PROGRAMMKONSTRUKTION UND SIMULATION

Stochastische und ereignisorientierte Simulation

(Skriptum zur Lehrveranstaltung)

Prof. Dr. Timm Grams
Fachhochschule Fulda
Fachbereich Elektrotechnik und Informationstechnik

16.09.2008 (Erstausgabe: 15.08.2004)

Beschreibung der Lehrveranstaltung

Ziel: Die Teilnehmer lernen die Modellierung von Systemen und Prozessen und das Durchführen von Berechnungsexperimenten mit dem Computer. Die mathematischen und methodischen Grundlagen der *stochastischen* und der *ereignisorientierten Simulation* sollen das Verständnis für die Leistungen und die Leistungsgrenzen existierender Simulationssoftware schaffen und dazu befähigen, mit leicht zugänglichen Werkzeugen Simulationsmodelle direkt zu programmieren (Tabellenkalkulation, Java Software Development Kit). Zu den Anwendungsbeispielen gehören Entscheidungsprozesse in Produktion und Lagerhaltung, Werkstattfertigung und Rechnernetze.

Grundsätze: Die Entwicklung der Techniken der digitalen Simulation ist traditionell eng verknüpft mit der Entwicklung der Programmiersprachen und der Programmiermethodik[1]. Deshalb stehen im Zentrum der Lehrveranstaltung das Lernen und das Einüben der Methoden und der Vorgehensweisen der Softwaretechnik im Rahmen von Modellbildung und Simulation.

Wert gelegt wird auf ein mehrstufiges Vorgehen: Analyse, Spezifizierung, Entwurf, Implementierung und Validierung. Modellierungssprache ist die *Unified Modeling Language* (UML). Programmiert wird in der objektorientierten Sprache Java. Die grafische Ergebnisdarstellung und Aufbereitung geschieht mit Tabellenkalkulation.

Voraussetzungen: Die für die Programmierung benötigten Elemente der objektorientierten Programmiersprache Java werden im Laufe der Lehrveranstaltung an Hand von Beispielen erläutert. Vorausgesetzt werden gute Kenntnisse der Programmiersprache C.

Zur Typographie: Programme und insbesondere deren Variablen werden grundsätzlich nicht kursiv geschrieben. `Schreibmaschinenschrift` wird für Programmtexte verwendet, wenn es der Übersichtlichkeit dient: Verbesserung der Unterscheidung zwischen Programmteilen und beschreibendem Text, Verdeutlichung der Programmstruktur durch Einrücken. Kursiv stehen alle Variablen und Funktionsbezeichner, die nicht Bezeichner in einem Programm sind, also insbesondere die Variablen im mathematischen Sinn oder Abkürzungen und Bezeichner für Programmteile. Kursivschrift dient im Fließtext der Hervorhebung - beispielsweise beim erstmaligen Auftreten wichtiger Stichwörter.

Begleitmaterial zur Lehrveranstaltung: Begleitmaterial, Links sowie die Problem- und Aufgabenstellungen zu den Projekten sind auf der Web-Seite

`http://www.fh-fulda.de/~grams/simulati.htm`

zu finden.

Aufbau und Ablauf der Lehrveranstaltung: Die Lehrveranstaltung ist in zwölf *Lektionen* gegliedert. Jede Lektion besteht aus *Vorlesung*, *Übung* und *Praktikum*. In den Übungsstunden werden die Übungen des Skriptums bearbeitet und besprochen. Außerdem dienen die Übungsstunden dazu, die Arbeiten des Praktikums vorzubereiten. Im Rahmen des Praktikums werden mehrere kleine *Simulationsstudien* und ein größeres *Simulationsprojekt* nach folgendem Muster bearbeitet:

1. *Pflichtenblatt*. Das Pflichtenblatt ist Ergebnis der Analysephase.

---

[1] Der an historischen Details Interessierte sei hier auf die Quellen verwiesen (Dahl, Hoare, 1972). In dem inzwischen klassischen Artikel beschreiben die Schöpfer der Programmiersprache SIMULA 67 deren wesentlichen Sprachelemente. Diese dienten der eleganten Modellierung und Simulation von Wartesystemen („ereignisorientierte Systeme“). Die zu Grunde liegenden Strukturierungsprinzipien wurden Jahrzehnte später unter dem Etikett „objektorientierte Programmierung“ Allgemeingut.

2. *Entwurf*. Erstellung von Objekt-, Klassen- und Zustandsdiagrammen. Beschreibung der Programmstrukturen der imperativen Teile mit Mitteln der Programmiersprache (Grobentwurf).
3. *Implementierung*. Spezifikation der im Grobentwurf benannten Programmteile. Programmkonstruktion, Verifizierung und Validierung: Erstellung von Prognosen. Überprüfung des Modells an Spezialfällen. Durchführung der Experimente. Erfassung und kritische Würdigung der Ergebnisse.

Jede Phase wird auf maximal zwei DIN A 4 Seiten dokumentiert und im Seminar präsentiert (soweit die Zeit dafür reicht). Eine Phase darf erst bearbeitet werden, wenn die vorhergehende Phase abgeschlossen und testiert worden ist. Gruppenarbeit in Gruppen zu je drei wird empfohlen. Die Testate werden im Rahmen von kurzen Fachgesprächen gegeben. Rollenverteilung: Der Lehrgangsleiter ist *Auftraggeber*. Die Arbeitsgruppe ist *Auftragnehmer*.

Gliederung

Literaturverzeichnis

Lehrbücher zur ereignisorientierten Simulation

Bratley, P; Fox, B. L.; Schrage, L. E.: A Guide to Simulation. Springer, New York 1987. *Ein solides Grundlagenwerk zum Thema Simulation, wie es hier aufgefasst wird.*

Fishwick, P. A.: Simulation Model Design and Execution. Prentice Hall, Englewood Cliffs, New Jersey 1995

Law, A. M.; Kelton, W. D.: Simulation Modeling & Analysis. McGraw-Hill, New York 1991

Page, B.: Diskrete Simulation. Eine Einführung mit Modula-2. Springer, Berlin, Heidelberg 1991. *Deckt das Stoffgebiet ab. Anstelle von Java wird hier mit Modula-2 gearbeitet.*

Weiterführende Literatur

Ameling, W. (Hrsg.): Buchreihe "Fortschritte der Simulationstechnik". Herausgegeben im Auftrag der Arbeitsgemeinschaft Simulation (ASIM). Vieweg, Braunschweig/Wiesbaden

Tavangarian, D. (Hrsg.): Simulationstechnik. 7. Symposium des ASIM-Fachausschusses 4.5 Simulation der Gesellschaft für Informatik, Hagen 1991. Vieweg, Braunschweig 1991.

Nachschlagewerke

Fisz, M.: Wahrscheinlichkeitsrechnung und mathematische Statistik. DVW Berlin 1976

Gnedenko, B. W.: Lehrbuch der Wahrscheinlichkeitsrechnung. Verlag Harri Deutsch, Thun 1980

Kleinrock, L.: Queuing Systems. Vol. 1: Theory. Wiley-Interscience 1975. *Diese zwei Bände von Kleinrock sind die Standardreferenz zu Wartesystemen. Der Anwendungsschwerpunkt liegt auf den Rechnernetze.*

Kleinrock, L.: Queuing Systems. Vol. 2: Computer Applications. Wiley-Interscience 1976

Knuth, D.: The Art of Computer Programming. Vol. 2: Seminumerical Algorithms. Addison-Wesley, Reading Mass. 1981. *Hier findet man (fast) alles, was man an Mathematik der reellen Zahlen in der Informatik braucht, insbesondere Zufallszahlengeneratoren und numerisch einwandfreie Formeln für die Berechnung statistischer Kenngrößen.*

Programmiersprachen und Programmiermethodik

Balzert, H: Lehrbuch Grundlagen der Informatik. Konzepte und Notationen in UML, Java und C++, Algorithmik und Software-Technik. Anwendungen. Spektrum Akademischer Verlag, Heidelberg, Berlin 1999. *Für die objektorientierte Software-Entwicklung verwenden wir das Werkzeug GO (enthalten auf der CD zum Buch). Verbindung: Softwaretechnik an der Ruhr-Universität Bochum.*

Dahl, O.-J.; Hoare, C. A. R.: Hierarchical Program Structures. In: Dahl/Dijkstra/Hoare: Structured Programming. Computer Science Classics. Academic Press, London 1972. *Hier erfährt man von den Software-Pionieren, wie es zu den objektorientierten Sprachen gekommen ist und was sie mit der Simulation zu tun haben.*

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Mass. 1995. *Im Package EventSim erkennt man das Entwurfsmuster der Template Method. Spezielle Ausprägungen der Entwurfsmuster Singleton und Wrapper findet man in der SimulationThread-Klasse wieder. Bei etwas komplizierteren Datenstrukturen stößt man schnell auf die Notwendigkeit, das Entwurfsmuster Iterator (Cursor) zu realisieren.*

Gosling, J.; Joy, B.; Steele, G.: The Java™ Language Specification. Version 1.0. Addison-Wesley, Reading, Mass. First printing: August 1996. ISBN 0-201-63451-1. *Der Java-Standard.*

Kreutzer, W.: Grundkonzepte und Werkzeugsysteme objektorientierter Systementwicklung - Stand der Forschung und Anwendung. Wirtschaftsinformatik, 32(Juni 1990)3, 211-227

Meyer, B.: Object-oriented Software Construction. Prentice Hall 1988

Oestereich, B.: Die UML-Kurzreferenz für die Praxis - kurz, bündig, ballastfrei. Oldenbourg, München 2002

Multithreading

Arnold, Ken; Gosling, James: The Java™ Programming Language. Addison-Wesley, Reading, Mass. 1996. *Eine Einführung in die Sprache Java mit besonderer Berücksichtigung der Thread-Programmierung.*

Oaks, S.; Wong, H.: Java Threads. O'Reilly, Cambridge 1997. *Leicht fassliche und umfassende Darstellung des Arbeitens mit Java-Threads. Auflistung der System-Threads und der Threads von GUI-Anwendungen.*

Petzold, Charles: Programming Windows. 5th ed. Microsoft Press 1999

Hintergrundmaterial

Eibl-Eibesfeldt, I.: Die Biologie des menschlichen Verhaltens. Piper, München 1984

Grams, T.: Denkfallen und Programmierfehler. Springer, Heidelberg 1990

Grams, T.: Grundlagen des Qualitäts- und Risikomanagements. Zuverlässigkeit, Sicherheit, Bedienbarkeit. Vieweg Praxiswissen, Braunschweig, Wiesbaden 2001

Krech, Crutchfield u. a.: Grundlagen der Psychologie. Band 4 Kognitionspsychologie. Beltz, Weinheim 1992

McGeoch, C.: Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups. ACM Computing Surveys. 24 (June 1992) 2, 195-212

Sasieni, M.; Yaspan, A.; Friedman, L.: Methoden und Probleme der Unternehmensforschung. Physica-Verlag Würzburg, Wien 1965

1 Einführung in die Simulation

Zweck und Werkzeuge

Mittels Simulation lassen sich schwer durchschaubare Zusammenhänge spielerisch erkunden. Sie ist eine ideale Lernhilfe. Sie erlaubt ungefährliche und preiswerte Experimente mit komplexen oder experimentell unzugänglichen Systemen (Fahrzeuge, Kraftwerke, Ökosysteme, Volkswirtschaften). Experimente lassen sich wiederholen und in beliebigen räumlichen und zeitlichen Maßstäben darstellen (Lupe, Zeitlupe).

Die vorliegende kurze Einführung in die Simulation stellt exemplarisch die grundlegenden Simulationstechniken vor. Die folgenden Ziele werden verfolgt:

- Darstellung der Möglichkeiten und Grenzen der Simulation auch für denjenigen, der ein bereits fertig vorliegendes Simulationsprogramm anwenden will.
- Orientierungshilfe und Einstieg in die wichtigsten Techniken für den angehenden Entwickler eines Simulationssystems.
- Hilfestellung für das Erstellen kleinerer Simulationsanwendungen mit allgemein verfügbaren Werkzeugen.

Programmiersprachen und Simulationswerkzeuge

Zu den allgemein verfügbaren Werkzeugen zählen

- die Tabellenkalkulation
- die imperativen (Pascal, C), und
- die objektorientierten Programmiersprachen (Delphi, Java).

Spezielle Simulationswerkzeuge sind

- anwendungsorientierte Simulationsprogramme (Spice, Simulink), und
- Simulatoren (ICE-Simulator, Kraftwerkssimulatoren).

Simulation ist eine interdisziplinäre Aufgabe: Neben den wissenschaftlichen Grundlagen des Anwendungsgebiets finden sich in der Simulationstechnik eine Reihe von weit entwickelten Fachgebieten: Systemanalyse, Softwaretechnik, Interface-Design, Angewandte und Numerische Mathematik, Qualitätssicherung, Daten- und Versionsverwaltung, Dokumentation.

Simulation ist alltäglich

Warum simulieren wir? Fragen nach Sinn und Zweck unserer Handlungen stellt man am besten dem Verhaltensforscher. Er meint: "Angetrieben von seiner Neugier setzt sich der Mensch von frühester Kindheit an aktiv mit seiner Umwelt auseinander; er sucht nach neuen Situationen, um daraus zu lernen. Er manipuliert die Gegenstände seiner Umwelt auf vielerlei Art, und seine Neugier endet erst, wenn ihm das Objekt oder die Situation vertraut wird oder wenn er die Aufgabe, die sich ihm stellte, gelöst hat" (Eibl-Eibesfeldt, 1984).

Informationsgewinn durch Experimente

Das natürliche Verfahren zur Befriedigung der Neugier - zur Informationsgewinnung also - ist das Experiment mit dem zu erforschenden System: Will ich wissen, ob das Hemd passt, ziehe ich es probeweise an.

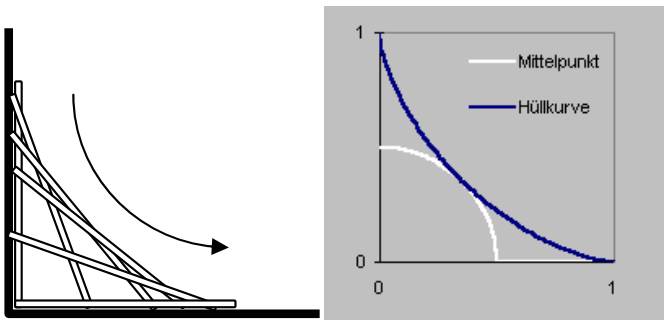
Aber manchmal geht das nicht so einfach. Dann heißt es, nachdenken! Die folgende kleine Denksportaufgabe soll klar machen, was dabei passiert.

Die "rutschende Leiter"

Aufgabe: Eine Leiter steht zunächst senkrecht an der Wand. Dann wird ihr Fußende langsam von der Wand weggezogen, bis die Leiter ganz auf dem Boden liegt. Auf welcher Kurve bewegt sich dabei der Mittelpunkt der Leiter? Ist die Kurve konkav oder konvex - also nach unten oder nach oben gekrümmt?

Lösungsversuche: Intuitiv wird meist eine konvexe, tangential an Wand und Boden anliegende Kurve vermutet. Richtig aber ist, dass sich der Mittelpunkt der Leiter, die an der Wand herabrutscht, genau so bewegt wie der Mittelpunkt einer Leiter, die einfach kippt, bei der also der Fußpunkt unverändert an der Wand bleibt.

Dass das so ist, ergibt sich aus folgendem Gedankenexperiment: Man stelle sich die beiden Leitern in der Mitte verbunden vor, wie eine Schere. Beim Öffnen dieser "Schere" rutscht die eine Leiter an der Wand entlang und die andere kippt. Der Verbindungspunkt beider ist ihr jeweiliger Mittelpunkt. Der Mittelpunkt bewegt sich also auf einem Viertelkreis, dessen Enden auf Wand und Boden senkrecht stehen; er beschreibt eine konkave Kurve.



Wie kommt es zu der ursprünglichen intuitiven und falschen Antwort? Man kann sich den Irrtum etwa so erklären: Wir stellen uns vor, wie die Leiter fällt. Dazu bilden wir im Kopf Modelle der Gegenstände und bewegen sie probeweise.

Denken als Simulation im Kopf

Dieses Probehandeln im vorgestellten Raum ist eine Art *Simulation im Kopf*. Albert Einstein sieht darin - wie zuvor bereits Sigmund Freud - das Wesen des Denkens: "Die geistigen Einheiten, die als Elemente meines Denkens dienen, sind bestimmte Zeichen und mehr oder weniger klare Vorstellungsbilder, die 'willkürlich' reproduziert und miteinander kombiniert werden können ... dieses kombinatorische Spiel scheint die Quintessenz des produktiven Denkens zu sein" (Krech, Crutchfield u.a., 1990).

Wie so oft, wurde auch hier ein Mechanismus dadurch entdeckt, dass er zu einem Irrtum führte. Der Irrtum hat seine eigentliche Ursache aber nicht im Probehandeln. Unsere Beobachtung und Auswertung wurde durch die sogenannte Prägnanztendenz fehl geleitet: Die prägnante Hüllkurve (Einhüllende, Enveloppe) der Leiterbewegung drängt die schwer zu verfolgende Kurve der Mittelpunktsbewegung in den Hintergrund.

System, Modell, Experiment

Experimente mit dem realen Objekt sind nicht immer durchführbar: Versuchsobjekte können unzugänglich sein (zu groß und zu weit weg wie die Planeten - oder auch zu klein wie die Moleküle); die Dauer der Experimente übersteigt manchmal unsere Geduld oder gar unsere Lebensdauer (bei Evolutionsprozessen beispielsweise); manches (wie ein elektrischer Vorgang) geht einfach zu schnell; der Versuch kann zu gefährlich sein (wie das in der Chemie und in der Kerntechnik der Fall ist); mit dem Wetter oder einer Volkswirtschaft zu experimentieren, verbietet sich von selbst.

Simulation ist das Experimentieren mit Modellen

Einen Ausweg bietet das *Experimentieren mit einem Modell*. Und genau das nennen wir Simulation. Dabei spielt es zunächst keine Rolle, aus welchem Material die Modelle sind. Sie können

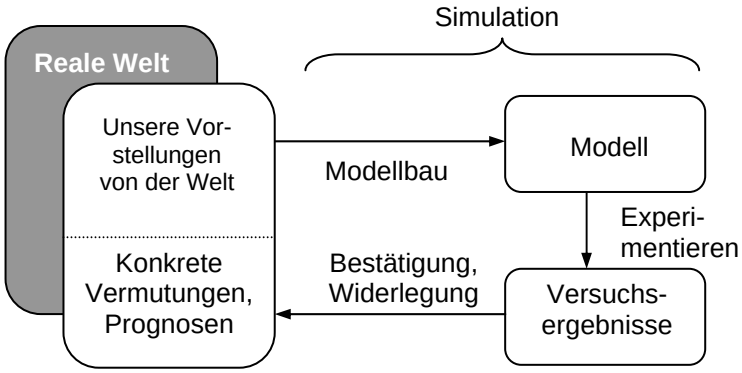
- aus Pappe, Holz, Blech und manch anderem Material sein;
- auf dem Papier in Form von Zeichnungen und Berechnungen vorliegen;
- im Computer als Programm existieren;
- in unserem Kopf vorhanden sein.

Digitale Simulation ist die Durchführung von Berechnungsexperimenten mit dem Computer

Hier geht es speziell um Berechnungsexperimente mit dem Computer, um die *Digitale Simulation.*

Was genau will man mit einer Simulation erreichen? Dem Naturwissenschaftler geht es um das Überprüfen von Theorien: Er baut ein Modell nach den Vorgaben der Theorie und untersucht, ob die Vorhersagen des Modells mit der Wirklichkeit zusammenpassen. Der Betriebswirtschaftler hingegen sucht nach einer optimalen unternehmerischen Entscheidung durch experimentelles Durchspielen von Alternativen. Der Ingenieur benötigt Unterstützung beim Entwurf und bei der Optimierung technischer Systeme. Weitere Simulationszwecke sind: die Wettervorhersage, Prognosen zur Entwicklung der Umwelt oder des Marktes, das computerbasierte Training (Beispiele: ICE-, Kraftwerks-, Flugsimulator).

Simulation erweitert Denkfähigkeit



Über eines müssen wir uns bei der Simulation immer im Klaren sein: Simulationsmodelle sind keine Abbildungen oder bloße Abstraktionen der realen Welt. Sie sind Abbildungen unserer Vorstellungen von der Welt. Und unsere Vorstellungen sind nur Anpassungen an die Welt. Ihre Güte wird daran gemessen, inwieweit sie uns helfen, in der Welt zurechtzukommen. "Simulation is really only an extension of human intellect, not the way things behave in nature" meint Eugene Miya dazu (The Risk Digest, ACM, 23.4.86).

Skepsis ist die rationale Einstellung gegenüber unseren Programmen und gegenüber dem Computer. Insbesondere bei der Simulation heißt es, immer wieder kritisch zu prüfen, ob die Theorie korrekt in ein Computerprogramm übertragen worden ist und ob die Theorie wirklich richtig und der Sache angemessen ist.

Fehler drohen bei Modellbildung und Versuchsauswertung

Irrtümer drohen nicht nur bei der Beobachtung und der Auswertung von Experimenten, wie wir oben gesehen haben. Wir haben große Chancen, unsere Irrtümer bereits in die Modelle einzubauen.

Vor allem sollten wir nie Experimente mit dem Computer durchführen, wenn wir von den Resultaten noch gar keine Ahnung haben. Unser Wahrnehmungsapparat, der ganz auf Sinnsuche in der Welt eingestellt ist, wird sonst nämlich auch in absurde Ergebnisse noch Zusam-

menhänge hineinkonstruieren und irgendetwas Verwertbares sehen. Wir verspielen so die Chance, den Unsinn zu bemerken.

Simulation planen und erwartetes Ergebnis vorab formulieren

Die Gefahr, dass uns der Computer fehl leitet, lässt sich verringern. Grundsätzlich wird man die Experimente planen. Noch vor der Durchführung wird eine - vielleicht zunächst nur grobe - Erwartung hinsichtlich des Ergebnisses gebildet und festgehalten. Erst dann folgt der Versuch.

Aus Fehlern lernen

Ein von der Erwartung abweichendes Simulationsergebnis sollte uns freuen, denn in genau diesem Fall können wir etwas hinzulernen. Das gelingt uns aber nur, wenn wir die Abweichung genau untersuchen und ihrer Ursache nachgehen. Zunächst ist zu prüfen, ob die Abweichung auf einen Programmierfehler oder etwas Ähnliches zurückgeht. Ist das ausgeschlossen, kann es an unserem schlechten Verständnis des simulierten Gegenstands liegen. Eine Fehleranalyse zeigt uns, wo wir falsch liegen und wie wir zu besseren Prognosen kommen können.

Die negative Methode

Da wir nicht Bestätigung suchen, sondern uns über entdeckte Fehler freuen, sprechen wir auch von der *negativen Methode* (Grams, 1990). Sie ist die Methode der Wissenschaft.

Vorgehen bei der Simulation

Bei der Durchführung von Simulationen lassen sich die Phasen Problemformulierung, Modellerstellung, Experimentieren und Dokumentieren unterscheiden. Hier einige der Fragen, die in den einzelnen Abschnitten zu beantworten sind..

PROBLEM formulieren: Was ist das Ziel? Welche Art von Ergebnissen wird von der Simulation erwartet? In welcher Form werden die Ergebnisse erwartet?

MODELL erstellen: Wie sehen die innere Struktur und des Wirkungsgefüges des zu untersuchenden Systems aus? Was sind die Parameter des Modells? In welchen Größenordnungen bewegen sie sich? Ist das Modell eine gültige Repräsentation des Untersuchungsgegenstands? Gibt es analytische Lösungen für Spezialfälle? Werden diese durch das Simulationsmodell gut reproduziert? Werden bekannte und durch Daten belegte Effekte gut genug modelliert (Validation)? Welche Programmiersprache ist geeignet? Welches Vorgehen ist bei der Programmkonstruktion zu wählen? Ist das Programm korrekt (Verifikation[2])?

EXPERIMENTE planen, durchführen, und auswerten: Für welche Zeit- und Parameterbereiche sollen die Berechnungen durchgeführt werden? Welche Daten sind für eine Weiterverarbeitung aufzuheben? Hinweis: Man beginnt die genaue Untersuchung des Gegenstands erst dann, wenn das "Fernrohr" grob eingestellt ist.

ERGEBNISSE darstellen: Sind die Ergebnisse allgemeinverständlich und prägnant zusammengefasst? Ist Wiederauffindbarkeit der Ergebnisse zu einem späteren Zeitpunkt sichergestellt? Welchen Aufbau hat das Dokumentationssystem?

Das Vorgehensmodell der Simulation mit den Schritten PROBLEMFORMULIERUNG, MODELLERSTELLUNG, EXPERIMENTE und ERGEBNISDARSTELLUNG stimmt leider

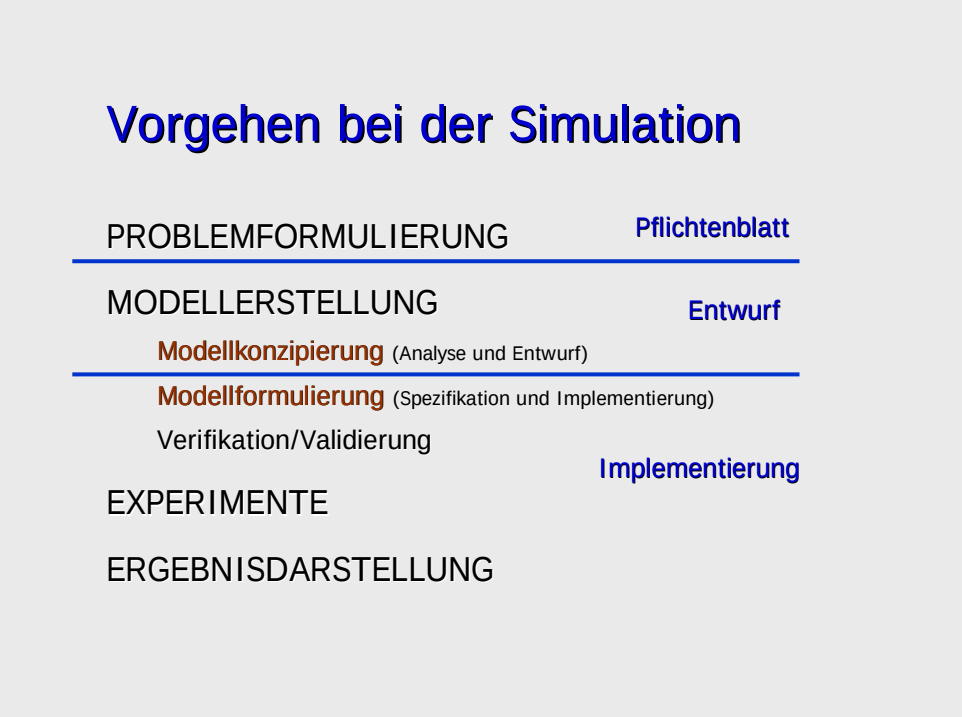
---

[2] Zu den Begriffen: Die Verifikation sagt, ob wir das System richtig gebaut haben, und die Validierung sagt, ob wir das richtige System gebaut haben.

nicht mit den üblichen Vorgehensmodellen der Software-Entwicklung überein. Die *Software-Konstruktion* geschieht in den Hauptschritten

- *Pflichtenblatt* (einschließlich Spezifikation)
- *Entwurf* und
- *Implementierung* (Realisierung des Programms einschließlich Verifikation und Validation).

Das folgende Schaubild stellt den Zusammenhang zwischen den *Vorgehensmodellen* her.



Musterpflichtenblatt für Simulationsprojekte

Vorbemerkung

Zweck: Das Musterpflichtenblatt bezieht sich auf Aufgabenstellungen im Rahmen der Lehrveranstaltung Simulation. Beim Pflichtenblatt handelt es sich um eine Vereinbarung zwischen Auftraggeber (das ist die Rolle des Lehrers) und Bearbeitern (Rolle der Studierenden). Zur Erhöhung der Zuverlässigkeit der Kommunikation wird das Pflichtenblatt von den Bearbeitern erstellt.

Struktur des Pflichtenblattes: Dem Musterpflichtenblatt liegt das *allgemeine Vorgehen bei der Simulation* zu Grunde (vorhergehender Abschnitt).

Inhalt des Pflichtenblatts: Es beschreibt für jeden der aufgeführten Punkte den Wissensstand und die Lücken (Probleme), die im Verlauf des Projekts zu schließen sind (Ziele). Es enthält Hinweise, wie das geschehen kann (Methoden).

Vorgehensmodell: Das Musterpflichtenblatt spiegelt die Vorgehensweise heutiger objektorientierter Softwareentwicklung wieder (Gamma, Helm, Johnson, Vlissides, 1995; Meyer, 1988). Die objektorientierte Software-Entwicklung wird auf der Grundlage der *Unified Modeling Language* (UML) mit einem frei verfügbaren Werkzeug (beispielsweise GO der Ruhr-Universität Bochum) durchgeführt (Balzert, 1999).

Struktur des Pflichtenblatts

Kopfteil: Thema, Bearbeiter

PROBLEMFORMULIERUNG

Es soll klar werden, dass tatsächlich am richtigen Problem gearbeitet wird. Kontrollfragen: Was ist der Untersuchungsgegenstand? Was ist über ihn bekannt? Was soll in Erfahrung gebracht werden? Wenn nur eine einfache Aufgabe zu bearbeiten ist, steht hier die (präzisierte) Aufgabenstellung

MODELLERSTELLUNG.

Modellkonzipierung (Analyse- und Entwurfsphase): Beschreibung der inneren Struktur und des Wirkungsgefüges des zu untersuchenden Systems. Darstellungsmittel: Blockschaltbilder und Systemgleichungen, Ablauf- und Zustandsdiagramme. Identifizierung von Entwurfsmustern (Design Patterns). Beschreibung der Programmstruktur auf hoher Abstraktionsebene:

Beschreibung der imperativen Teile mit den Mitteln der Programmiersprache (Pseudo-Code): Beschreibung der Programmstruktur (Sequenz, Auswahl, Schleife) und Spezifizierung der Programmabschnitte in Form von Kommentaren. (Wie das geht, wird im Skriptum zur Lehrveranstaltung "Programmieren" gezeigt.)

Beschreibung objektorientierter Strukturen mittels Objekt- und Klassendiagrammen in der Unified Modeling Language (UML).

Festlegungen zur *Ein-* und *Ausgabe:* Auflistung aller Parameter und Daten, mit denen das Programm zu versorgen ist. Beschreibung der Datenerfassung, der Datenaufbereitung und Darstellungsformen.

Entwurf und Strukturierung der *Bedienoberfläche:* Festlegungen zur *Konsolenversion* (Texteingabe und -ausgabe über Tastatur und Bildschirm), zur *grafischen Bedienoberfläche* (Graphical User Interface, GUI) und zur *interaktiven Steuerung* des Programmablaufs.

Festlegung der Strukturen der *Ein-* und *Ausgabedateien:* Eingabesprache zur Beschreibung der Systemstruktur, der Parameter und der Steuerung des Programmablaufs. Schnittstellen zu anderen Programmen. Weiterverarbeitung und Darstellung der Daten (mit einem Tabellenkalkulationsprogramm beispielsweise).

Datenerfassung: Abschätzung der Parameter des Modells und der Anfangsbedingungen.

Modellformulierung (Spezifikations- und Implementierungsphase):

- *Spezifikation* des Modells und seiner Subsysteme mittels Vor-/Nachbedingungen und Invarianten. Programming by Contract.
- *Definition* der Ein- und Ausgabe: Bedienoberfläche und Datei-Strukturen.
- *Validierung der Spezifikation.* Überprüfung der Spezifikation anhand analytischer Lösungen für Teilaspekte und Grenzfälle.
- Auswahl der Programmiersprache.
- Programmkonstruktion.

Verifikation des Programms. Programmbeweis für kritische Teile des Modells.

Validierung des Gesamtsystems anhand realer Daten. *Kalibrierung* des Modells.

EXPERIMENTE

Es ist festzulegen, für welche Zeit- und Parameterbereiche die Berechnungen durchgeführt werden sollen und welche Daten zur Weiterverarbeitung aufzuheben sind. Vor der Durchführung der Experimente ist eine *Prognose* der Ergebnisse zu erstellen. Man beginnt die genaue Untersuchung des Gegenstands erst dann, wenn das "Fernrohr" grob eingestellt ist.

ERGEBNISDARSTELLUNG

Dokumentation: Allgemeinverständliche und prägnante Zusammenfassung der Ergebnisse. Sicherstellung der Wiederauffindbarkeit im Rahmen eines Dokumentationssystems.

Schlussteil: Unterschriften, Ort, Datum

Übung

Beginnen Sie mit der Erstellung der Pflichtenblätter für die Simulationsstudien und Simulationsprojekte.

2 „Imperative“ (nicht objektorientierte) Programme mit Java

Zur Programmierung

Dieser Kurs setzt keine Kenntnis der Programmiersprache Java voraus. Andererseits führt er auch nicht so in die Sprache ein, wie man es von einem Programmierkurs erwarten könnte. Die Elemente der Programmiersprache werden an Programmierbeispielen erläutert, und zwar durch Bezugnahme auf die Programmiersprache C.

Die genauen Definitionen sollten im Zweifelsfall immer in der aktuellen Sprachdefinition (Java Language Specification) nachgeschlagen werden. Was für C die Standardbibliothek von Funktionen ist (Standard Library), das ist für Java die Java™ 2 Platform API Specification. Dabei steht API für Application Programming Interface.

Beide Dokumente sind Bestandteil des *Java Software Development Kit* (Java SDK) der Firma Sun Microsystems (http://www.javasoft.com). Hier finden Sie alles, was Sie für die Entwicklung von Simulationssoftware im Umfang dieser Lehrveranstaltung benötigen.

Wenn Sie in der API herumstöbern, werden Sie gleich einen fundamentalen Unterschied zwischen C und Java entdecken: In der C-Standardbibliothek stehen *Funktionen,* denn Funktionen sind die wesentlichen Strukturelemente von C. Bei Java finden Sie statt dessen *Klassen.* So wie C-Programme sich aus Funktionen zusammensetzen, so setzen sich Java-Programme aus Klassen zusammen.

Klassen sind eine Verallgemeinerung der struct-Datentypen in C. Der class-Type in Java kann nicht nur *Member* (Felder) für Variablen enthalten, sondern auch solche für Funktionen. Variablen werden in der objektorientierten Programmierung *Attribute* genannt, und Funktionen heißen *Methoden*.

Nehmen wir als Beispiel eine Klasse für Zahlen. Neben dem Attribut `x` für die Zahl möge es noch die Methode `norm()` für die Bildung des Betrags geben:

```
class Number {
  double x;
  double norm(){return x<0?-x:x;}
}
```

Mit „`Number z = new Number();`“ wird eine *Referenz*[3] `z` auf ein Objekt der Klasse Number deklariert. Außerdem wird diese Referenz mit der Referenz auf ein neu erzeugtes *Objekt* dieser Klasse initialisiert. Nun können wir dem Attribut dieses Objekts beispielsweise mittels der Zuweisungsanweisung „`z.x=-3;`“ den Wert `-3` zuweisen. Der Methodenaufruf `z.norm()` liefert als Rückgabewert den Betrag der Zahl `z.x`.

Noch eins muss man wissen, bevor man die API durchstöbert: Klassen lassen sich durch den Mechanismus der Vererbung erweitern. Anstelle eine Klasse für Zahlenpaare von Grund auf neu zu definieren, kann man das Attribut x der Klasse `Number` bereits als gegeben annehmen. Man führt nur eine Typerweiterung mit Hilfe des Schlüsselworts `extends` ein. So kommt man zur Klasse der Zahlenpaare:

```
class Pair extends Number {
  double y;
  double norm(){return Math.sqrt(x*x+y*y);}
}
```

---

[3] Eine Referenz in Java ist so etwas wie ein Pointer (Zeiger) in C. Da Klassenvariablen in Java grundsätzlich nur dynamisch angelegt werden, entfällt in Java die Unterscheidung zwischen einer Feldauswahl über Pointer mittels Pfeil „->“ und über Variable mittels Punktnotation. Deshalb wird in Java für die Feldauswahl grundsätzlich die Punktnotation verwendet.

Für die neue Klasse muss die Methode `norm()` *überschrieben* werden. Was das genau heißt, wird später geklärt. Jedenfalls sind nach der Erzeugung eines Paares in der Deklaration „`Pair z = new Pair();`“ jetzt die folgenden Zuweisungen möglich: `z.x=3; z.y=4;` und der Aufruf `z.norm()` liefert nun die euklidische Norm des Zahlenpaares, im Beispiel den Wert 5.

Durch die Möglichkeit der *Typerweiterung* stehen Klasse zueinander in einer hierarchischen Beziehung. Die ursprüngliche Klasse ist in dieser Beziehung die *Oberklasse*. Die erweiternde Klasse ist in dieser Beziehung die *Unterklasse.* Ober- und Unterklasse heißen - in enger Anlehnung an den Java-Sprachgebrauch - auch *Superklasse* bzw. *Subklasse*.

In der Java-Dokumentation wird die Vererbungshierarchie folgendermaßen sichtbar gemacht (hier am Beispiel der Klasse Pair):

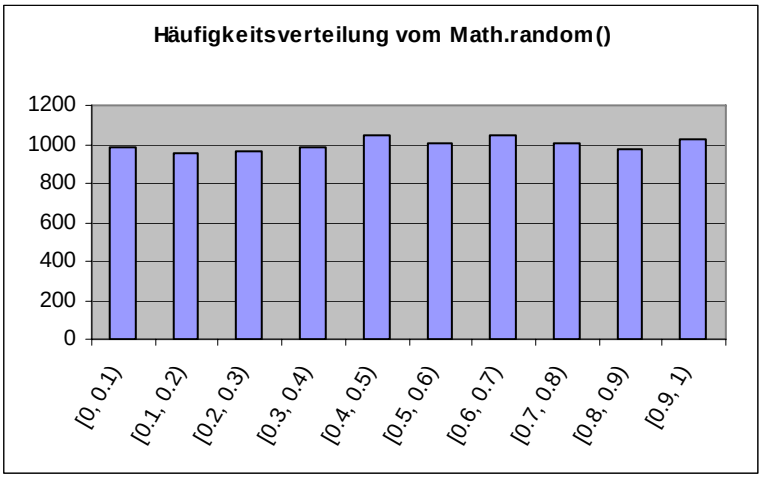
```
java.lang.Object
  |
  +--Number
        |
        +--Pair
```

Dabei fällt auf, dass die Klasse `Number` offenbar auch eine Superklasse hat, die Klasse `Object`. Sie bildet die Wurzel des Vererbungsbaums. Jede Klasse erbt also grundsätzlich die Methoden der Object-Klasse.

In Java werden die Klassen entsprechend ihren Aufgaben zu *Packages* gebündelt. Die für die Spache Java fundamentalen Klassen sind im Package `java.lang` zusammengefasst. Für die Ein-/Ausgabe gibt es das Package `java.io`.

Mit diesen Kenntnissen sollte es Ihnen gelingen, das folgende Programmierbeispiel nachzuvollziehen und die benötigen Sprachelemente in der API-Dokumentation aufzuspüren.

Ein Beispiel zum Einstieg



Aufgabe: Schreiben Sie ein Java-Programm, das ein Histogramm der Häufigkeitsverteilung des Zufallszahlengenerators Math.random() erzeugt. *Eingabe:* Auf der Kommandozeile soll das Programm die Anzahl der Zufallswerte und die Anzahl der Klassen des Histogramms entgegennehmen. Die *Ausgabe* des Histogramms geschieht in eine Textdatei als zweispaltige Tabelle. Die erste Spalte erhält die Überschrift „Klasse“. Sie enthält die den Klassen entsprechenden Wertebereiche der Zufallsgrößen (Klassenbreite). Die zweite Spalte erhält die Häufigkeiten unter der Überschrift „Häufigkeit“. Die Datenformatierung ist so zu wählen, dass die Textdatei problemlos in ein Excel-Programm für die grafische Darstellung eingegeben werden kann.

Das Java-Programm

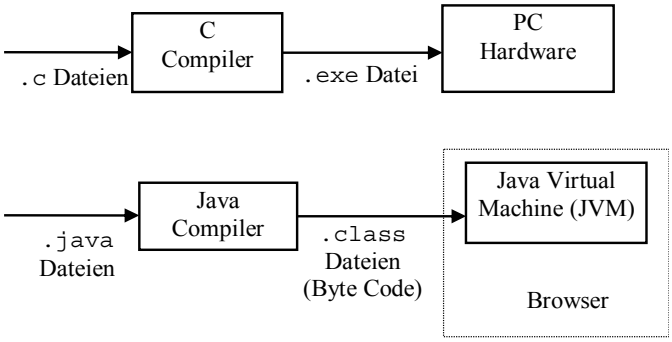
```
import java.io.*;
import java.text.*;
import java.util.*;

public class Histogramm {
```

```
  public static void main(String args[]) {
    if (args.length!=2) System.out.println(
      "\nEINGABE: Histogramm <long Stichprobenumfang> <long Klassenanzahl>"
    ); else {
      /*Eingabe*/
      long n= Long.parseLong(args[0]);  //Stichprobenumfang
      int k= Integer.parseInt(args[1]); //Klassenanzahl
      long[] h= new long[k]; //Histogramm
      System.out.println("ECHO DER KOMMANDOZEILEN-EINGABE");
      System.out.print("  Stichprobenumfang= "); System.out.println(n);
      System.out.print("  Klassenanzahl= "); System.out.println(k);

      /*Verarbeitung*/
      for (long i=0; i<n; i++) ++h[(int)Math.floor(k*Math.random())];

      /*Ausgabe*/
      DataOutputStream out;
      try {
        out= new DataOutputStream(new FileOutputStream("Histogramm.dat"));
        out.writeBytes("HISTOGRAMM\n");
        out.writeBytes("Klasse\tHaeufigkeit\n");
        for (int i=0; i<k; i++)
        out.writeBytes(i+"\t"+h[i]+"\n");
        out.close();
      } catch(IOException e) {
        System.out.println("\nFEHLER: Ausgabedatei ...");
      }
    }
  } //main
} //class
```

Die öffentliche (public) Histogramm-Klasse muss in einer Datei stehen, die Histogramm.java heißt.

Programm übersetzen und laufen lassen

Java-Programme werden normalerweise in ein Bytecode-Format übersetzt, die von der Java Virtual Machine (JVM) interpretiert werden kann. Die JVM kann in einen Brouwser eingebunden sein. Auf diese Weise können die Java-Applets von Web-Seiten angezeigt und aktiviert werden.



Die nebenstehenden Blockschaltbilder zeigen die Übersetzungs- und Interpretationsvorgänge für C und Java im Vergleich.

Der Java-Compiler heißt javac und die JVM hat den Bezeichner java. Beide liegen als exe-Dateien vor (javac.exe, java.exe).

Wenn Sie das Programm „zum Laufen bringen“ wollen, sind die folgenden Dinge zu tun:

1. Setzen der `classpath`-Variablen. Compiler und JVM müssen Zugriff auf Ihre Dateien haben, deshalb muss die `classpath`-Variable auf die entsprechenden Verzeichnisse gesetzt werden. Zum Beispiel so: `set classpath=.;c:\eigene~1\java;`
2. Übersetzen des Programms mit dem Aufruf `javac Histogramm.java`
3. Starten des Programms mit dem Aufruf `java Histogramm 1000 10`

4. Wenn alles geklappt hat, ist nun die Textdatei `Histogramm.dat` neu erzeugt worden. Schauen Sie sich die Datei an und erstellen Sie mit dem Programm Excel eine Grafik des Histogramms.
5. Variieren Sie die Eingabeparameter und wiederholen Sie die Schritte 3 und 4.

Übung

Ergründen Sie die Arbeitsweise des Programms in folgenden Schritten.

1. Lesen Sie das Programm durch und versuchen Sie zunächst, jeden Schritt zu verstehen. Sie werden sehen: Objektorientierte Programme lassen sich recht flüssig lesen und intuitiv verstehen. Die größten Verstöße gegen die natürlichen Sprache hat Java von C übernommen. Aber das kennen Sie ja.
2. Ordnen Sie die im Programm auftretenden Klassen den importierten Java-Packages zu. Klassen lassen sich daran erkennen, dass sie mit großem Anfangsbuchstaben geschrieben werden. Arbeiten Sie mit dem Java SDK.
3. Schauen Sie sich nun die einzelnen Methodenaufrufe an und lesen Sie im SDK nach, welche Wirkungen sie haben.
4. Schreiben Sie das Programm ab, übersetzen Sie es und lassen Sie es laufen.
5. Variieren Sie den Programmtext und erkunden Sie weiter die Semantik der Java-Sprachelemente.

3 Grundlagen der Wahrscheinlichkeitsrechnung

Elementare Wahrscheinlichkeitsrechnung

Wir betrachten Zufallsergebnisse, wie sie unter anderem beim Wurf eines Würfels auftreten. Das Zufallsergebnis bezeichnen wir einmal mit ζ. Die Menge der Werte, die die Variable ζ annehmen kann, ist der Ereignisraum Ω. Im Falle des Würfels besteht der Ereignisraum aus den Zahlen 1 bis 6, also $\Omega = \{1, 2, 3, 4, 5, 6\}$. Die Teilmengen des Ereignisraums heißen Ereignisse, die Elemente des Ereignisraums sind die Elementarereignisse.

Seien *A, B, C, ...* Ereignisse. Diese werden durch logische Bedingungen oder die explizite Angabe der Mengen dargestellt: $\zeta < 5$ steht im Falle des Würfels beispielsweise für die Menge $\{1, 2, 3, 4\}$. Eine Bedingung wird also mit ihrer *Erfüllungsmenge* – das ist die Menge der Werte, die diese Bedingung erfüllen - gleich gesetzt. Für die Vereinigung zweier Ereignisse A und B wird $A \cup B$ geschrieben und ihr Durchschnitt ist AB.

Die Ereignisse treten mit gewissen Wahrscheinlichkeiten ein. Die Elementarereignisse eines Wurfs mit einem gerechten Würfel haben alle die Wahrscheinlichkeit 1/6. Die tatsächlichen Wahrscheinlichkeiten für einen realen Würfel wird man über eine große Anzahl von Versuchen näherungsweise aus den relativen Häufigkeiten der einzelnen Punktzahlen ermitteln. Für die Wahrscheinlichkeit eines Ereignisses A schreiben wir $P(A)$.

Aus der Häufigkeitsdefinition der Wahrscheinlichkeiten lassen sich direkt die fundamentalen Regeln für das Rechnen mit Wahrscheinlichkeiten herleiten:

1. Für jedes Ereignis A gilt $0 \le P(A) \le 1$.
2. Die Wahrscheinlichkeit des unmöglichen Ereignisses ist gleich null: $P(\{\}) = 0$.
3. Die Wahrscheinlichkeit des sicheren Ereignisses ist gleich eins: $P(\Omega) = 1$.
4. Für zwei einander ausschließende Ereignisse A und B, wenn also $AB = \{\}$ ist, gilt die Regel der Additivität: $P(A \cup B) = P(A) + P(B)$.

Aus diesen Regeln lassen sich weitere gewinnen. Beispielsweise die folgende für zwei sich nicht notwendigerweise ausschließende Ereignisse A und B: $P(A \cup B) = P(A)+P(B)-P(AB)$.

Die bedingte Wahrscheinlichkeit des Ereignisses A unter der Bedingung B ist definiert durch

$$P(A|B) = \frac{p(AB)}{p(B)}.$$

Daraus folgt unter anderem die Formel

$$\frac{p(A \mid B)}{p(A)} = \frac{p(B \mid A)}{p(B)}.$$

Zufallsvariable

Zufallsvariable sind reellwertige Zufallsergebnisse. Sie sind durch ihre *Verteilungsfunktion* charakterisiert. Die Verteilungsfunktion F einer Zufallsvariablen X ist definiert durch

$$F(x) = P(X < x).$$

Dabei ist $P(X < x)$ die Wahrscheinlichkeit dafür, dass der Wert der Variablen kleiner als x ist. Die Verteilungsfunktion ist monoton wachsend und es gilt $F(-\infty) = 0$ und $F(\infty) = 1$. Die Wahrscheinlichkeit, dass der Wert der Variablen in das Intervall $[a, b)$ fällt, ist gegeben durch

$$F(b) - F(a).$$

Sei X eine *diskrete Zufallsvariable*, die die Werte x_i ($i = 1, 2, 3, ..., n$) annehmen kann, und p_i die *Wahrscheinlichkeit* dafür, dass $X = x_i$ ist. Die Verteilungsfunktion ist dann gegeben durch

$$F(x) = \sum_{i|x_i<x} p_i \ .$$

Stetige Zufallsvariable zeichnen sich dadurch aus, dass es eine *Dichte f*(*x*) gibt, so dass die Verteilungsfunktion *F*(*x*) sich als deren Integral darstellen lässt:

$$F(x) = \int_{-\infty}^{x} f(u)\,\mathrm{d}u \ .$$

Die wichtigsten Formeln und Kennzahlen seien hier für die diskreten Zufallsvariablen kurz in Erinnerung gerufen. Der *Erwartungswert* (*Mittelwert*) einer diskreten Zufallsvariablen *X* ist gegeben durch

$$\mu = E[X] = \sum_{i=1}^{n} x_i\, p_i \ .$$

Sei *g* eine reelle Funktion und *X* eine Zufallsvariable, dann definiert *g*(*X*) ebenfalls eine Zufallsvariable. Ihr Erwartungswert ist gegeben durch

$$E[g(X)] = \sum_{i=1}^{n} g(x_i) p_i \ .$$

Ist die Zufallsvariable stetig, ist ihr Erwartungswert gleich

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f(x)\mathrm{d}x \ .$$

Mit diesen Formeln lässt sich die *Varianz* (auch: *Streuung*) einer Zufallsvariablen ausrechnen. Die Varianz σ^2 einer Zufallsvariablen *X* ist definiert durch

$$\sigma^2 = E[(X - \mu)^2] = E[X^2] - \mu^2.$$

Der Wert σ ist die *Standardabweichung* der Zufallsvariablen.

Hat die Zufallsvariable *X* den Erwartungswert μ und die Standardabweichung σ, dann hat die durch die *lineare Transformation* $Y = aX + b$ definierte Zufallsvariable *Y* den Erwartungswert $a\mu + b$ und die Standardabweichung $|a|\sigma$.

Seien nun *X* und *Y* zwei Zufallsvariable, die die Werte x_i ($i = 1, 2, ..., n$) bzw. y_j ($j = 1, 2, ..., m$) mit den Wahrscheinlichkeiten p_i bzw. q_j annehmen. Die Wahrscheinlichkeit dafür, dass $X = x_i$ und $Y = y_j$ bezeichnen wir mit p_{ij}.

Offensichtlich bestehen zwischen den Wahrscheinlichkeiten die Beziehungen $p_i = \sum_j p_{ij}$ und $q_j = \sum_i p_{ij}$. Sind die Zufallsvariablen *statistisch unabhängig*, dann gilt: $p_{ij} = p_i\, q_j$.

Der Erwartungswert der Summe von Zufallsvariablen ist gleich der Summe der Erwartungswerte: $E[X + Y] = \Sigma_{ij}\,(x_i + y_j)\, p_{ij} = \Sigma_{ij}\, x_i\, p_{ij} + \Sigma_{ij}\, y_j\, p_{ij} = \Sigma_i\, x_i\, p_i + \Sigma_j\, y_j\, q_j = E[X] + E[Y]$.

Bei unabhängigen Zufallsvariablen gilt eine entsprechende Formel auch für das Produkt: $E[X \cdot Y] = E[X] \cdot E[Y]$. In diesem Fall ist die Varianz additiv: $\sigma^2_{X+Y} = \sigma^2_X + \sigma^2_Y$.

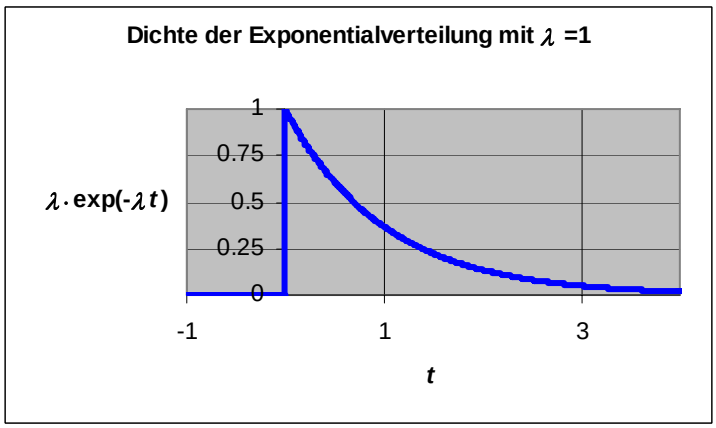
Verteilungen

Normalverteilung: Die Dichte der *Normalverteilung* ist gegeben durch

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \ .$$

Die Parameter μ und σ in dieser Formel haben zugleich die Bedeutung von Erwartungswert und Standardabweichung. Eine Zufallsvariable mit dieser Verteilung bezeichnet man kurz als

(μ, σ)-normalverteilt. Es handelt sich bei $f(x)$ um die berühmte Glockenkurve, deren Maximum bei μ liegt. Die Wendepunkte befinden sich jeweils im Abstand σ von der Maximumstelle.



Exponentialverteilung: Zufallsvariable ist eine Zeit T bis zum Eintritt eines Ereignisses, das unvorhersehbar ist, wie beispielsweise der Zerfall eines Atoms. Eine solche zufällige Zeit heißt *exponentialverteilt* wenn sie die folgende Eigenschaft besitzt: Die Wahrscheinlichkeit für das Auftreten des Ereignisses in einem Intervall [t, t+dt) - unter der Bedingung, dass es bis dahin noch nicht aufgetreten ist - ist unabhängig von der bis dahin verstrichenen Zeit t. Außerdem ist diese Wahrscheinlichkeit für hinreichend kleine Intervalle proportional zur Intervallänge dt.

Bezeichnet man die angesprochene Proportionalitätskonstante mit λ, so ist die Verteilungsfunktion der Exponentialverteilung gegeben durch

$F(t) = 0$ für $t < 0$, und

$F(t) = 1 - e^{-\lambda t}$ für $0 \le t$

Die zugehörige Verteilungsdichte ist gleich

$f(t) = \lambda\, e^{-\lambda t}$ für $0 \le t$.

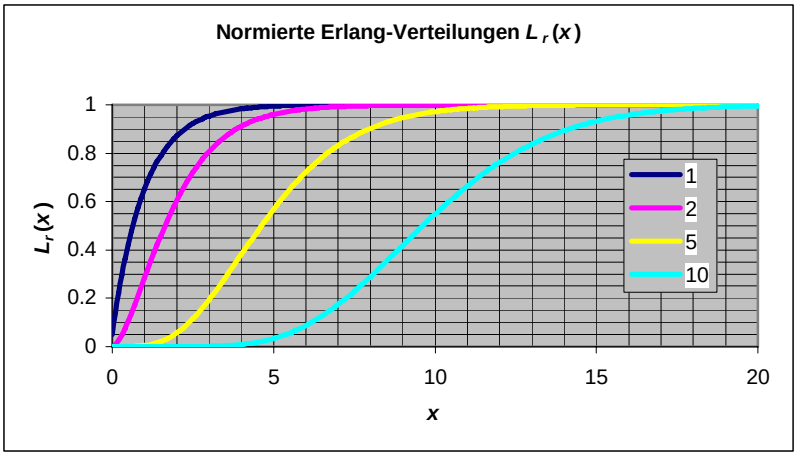
Erwartungswert und Standardabweichung dieser Verteilung sind jeweils gleich $1/\lambda$.

Dass die Exponentialverteilung tatsächlich die eben angesprochene Eigenschaft hat, soll kurz nachgewiesen werden: Die Wahrscheinlichkeit, dass das Ereignis in das - als hinreichend klein angenommene - Intervall [t, t+dt) fällt, ist annähernd gleich $f(t)\cdot$dt, also gleich $\lambda\cdot e^{-\lambda t}\cdot$d$t$. Die Wahrscheinlichkeit dafür, dass das Ereignis bis zum Zeitpunkt t noch nicht eingetreten ist, ist gleich $1\text{-}F(t)$, also gleich $e^{-\lambda t}$. Die bedingte Wahrscheinlichkeit dafür, dass das Ereignis im genannten Intervall eintritt, unter der Bedingung dass es vorher noch nicht eingetreten ist, ist gleich dem Quotienten $f(t)\cdot$d$t/(1\text{-}F(t))$ dieser Werte, also gleich $\lambda\cdot$dt. Das war zu zeigen: Die fragliche Wahrscheinlichkeit ist unabhängig von t und proportional zu dt.

Erlang-Verteilung: Die exponentialverteilten Zufallsvariablen T_1, T_2, ... T_r seien voneinander statistisch unabhängig. Die Verteilungen sind gleich, das heißt, sie haben alle denselben Wert des Parameters λ. Wir betrachten die Summe T dieser Zufallsvariablen $T = T_1 + T_2 + \ldots + T_r$. Die Verteilung dieser Zufallsvariablen wird als *Erlang-Verteilung mit dem Parameter r* bezeichnet. Ihr Erwartungswert ist das r-fache des Erwartungswerts eines jeden der Summanden: $E[T] = r/\lambda$. Die Verteilungsfunktion der Erlang-Verteilung lässt sich aus dem Markoff-Modell für den Poisson-Strom herleiten (Grams, 2001):

$$P(T<t) = 1 - \sum_{i=0}^{r-1} \frac{(\lambda\cdot t)^i}{i!} e^{-\lambda\cdot t} = \int_0^{\lambda\cdot t} \frac{u^{r-1}}{(r-1)!}\cdot e^{-u}\,\mathrm{d}u = L_r(\lambda\cdot t)$$

Wir setzen $x = \lambda\cdot t$ und sehen uns die so normierten Verteilungsfunktionen $L_r(x)$ für die Parameterwert $r = 1, 2, 5$ und 10 an.



Binomialverteilung: Bei der *Binomialverteilung* gehen wir von einem Versuchsschema aus, das aus einer Folge von n Versuchen besteht. Jeder der Versuche hat eines von zwei möglichen Ergebnissen, und zwar tritt das Ereignis A ein oder das dazu komplementäre Ereignis B. Die Wahrscheinlichkeit für das Ereignis A sei p. Die Wahrscheinlichkeit des Ereignisses B ist dann gleich $1 - p$. Die Ergebnisse der Versuche seien unabhängig voneinander.

Mit X_i bezeichnen wir die Zufallszahl, die den Wert 1 annimmt, wenn im i-ten Versuch Ereignis A eingetreten ist; im Fall des Ereignisses B hat sie den Wert 0. Mit X bezeichnen wir die Summe der Zufallsvariablen vom ersten bis zum n-ten Versuch: $X = X_1 + X_2 + ... + X_n$. Die Wahrscheinlichkeit dafür, dass das Ereignis A in der Versuchsserie genau k mal eintritt (und das Ereignis B demzufolge n-k mal), ist gleich

$$p_k = P(X = k) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} .$$

Die Verteilungsfunktion der (n, p)-Binomialverteilung ist

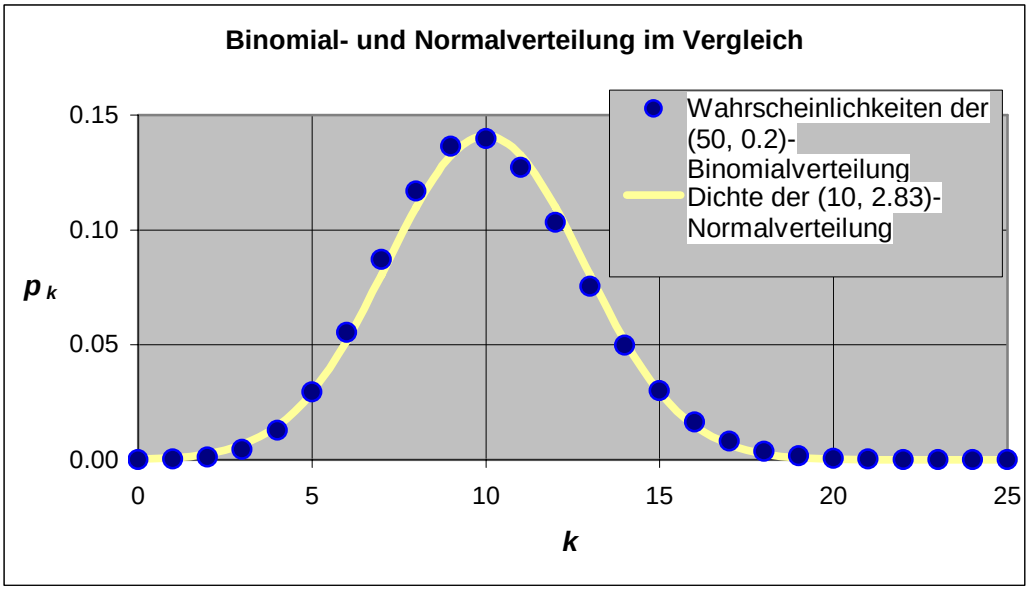
$$F(x) = P(X < x) = \sum_{k<x} \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} .$$

Der Erwartungswert der (n, p)-Binomialverteilung ist gegeben durch $\mu = n \cdot p$ und die Standardabweichung durch $\sigma = \sqrt{n \cdot p \cdot (1-p)}$.

Gleichverteilung: Die Verteilungsdichte der *Gleichverteilung* ist folgendermaßen definiert: $f(x) = 1$ für $x \in [0, 1)$ und $f(x) = 0$ sonst. Für die Verteilungsfunktion F gilt dann: $F(x) = 0$ für $x < 0$, $F(x) = x$ für $0 \le x < 1$, und $F(x) = 1$ für $1 \le x$. Erwartungswert und Streuung sind gegeben durch $\mu = 1/2$ und $\sigma^2 = 1/12$. Die Bedeutung der Gleichverteilung liegt darin, dass sie einfache Möglichkeiten zur Erzeugung beliebiger Verteilungen eröffnet.

Der zentrale Grenzwertsatz

Die herausragende Bedeutung der Normalverteilung rührt daher, dass viele Zufallsgrößen, die in der Natur beobachtet werden können, tatsächlich näherungsweise normalverteilt sind. Das lässt sich darauf zurückführen, dass diese Zufallsvariablen aus der Überlagerung vieler einzelner, weitgehend voneinander unabhängiger Einflüsse entstehen. Und eine Summe von vielen unabhängigen Zufallsvariablen gleicher Größenordnung ist tatsächlich annähernd normalverteilt. Die Näherung ist umso genauer, je größer die Anzahl der Summanden ist. Diesen Sachverhalt bezeichnet man als den *zentralen Grenzwertsatz* der mathematischen Statistik.



Beispiel Binomialverteilung: Die Binomialverteilung ist die Summe unabhängiger Zufallsvariablen. Die (10, 0.2)-Binomialverteilung wird wegen des zentralen Grenzwertsatzes durch die Normalverteilung mit demselben Erwartungswert von 10 und derselben Standardabweichung von 2.83 gut angenähert.

Übung

Demonstrieren Sie, dass die Summe aus zwölf auf dem Intervall [0, 1)-gleichverteilten Zufallszahlen eine Zufallszahl ergibt, die (6, 1)-normalverteilt ist. Legen Sie dazu ein Tabellenkalkulationsblatt an, mehr brauchen Sie nicht. Machen Sie eine Statistik über hundert Stichprobenwerte und zeigen Sie in einer Grafik den Vergleich mit der Normalverteilung. Eine Folge von Experimenten erhalten Sie durch wiederholte Neuberechnung des Blattes, die mit der Taste <F9> angestoßen werden kann.

4 Grundlagen der stochastischen Simulation

Methode der stochastischen Simulation

Die stochastische Simulation (auch: simuliertes Stichprobenverfahren) wird dann eingesetzt, wenn gewisse Parameter eines Modells oder auch Eingangsgrößen nicht fest gegeben, sondern zufallsbedingt sind.

Der Zufall kann an zwei Stellen des Systems eingreifen:

- Die Parameterwerte des Systems streuen. Beispiel aus der Fertigung technischer Systeme: Die Parameterwerte sind zwar für eine bestimmte Systemrealisierung konstant. Aber von Realisierung zu Realisierung derselben Art können sie durchaus verschieden sein, da die Bauelemente gewissen Fertigungstoleranzen unterliegen.
- Die Eingangsgrößen sind zufällige (stochastische) Prozesse, beispielsweise ein von Rauschen überlagertes Signal am Eingang eines Rundfunkempfängers.

Unter einer zufälligen Zeitfunktion versteht man eine Zeitfunktion $x(t)$ aus einer Schar (einem Ensemble) von möglichen Realisierungen. Die Gesamtheit (das Ensemble) der Zeitfunktionen bildet einen stochastischen Prozess $X(t)$. Zu jedem festen t ist $X(t)$ eine Zufallsvariable.

Sowohl bei stochastischen Parametern als auch bei stochastischen Eingangsfunktionen ergeben sich für die Zeitfunktionen der Ausgangsgrößen des Systems zufällige Zeitfunktionen. Sie müssen als Realisierungen stochastischer Prozesse aufgefasst werden.

Die Aufgabe kann nun beispielsweise lauten, die statistischen Eigenschaften einer Ausgangsgröße $Y(t)$ zu bestimmen. Dazu erzeugt man eine *Stichprobe* der Ausgangsgröße, indem man eine Reihe von Realisierungen für die stochastischen Eingangsgrößen und Parameter bestimmt und mit jeder der Realisierungen einen (deterministischen) Simulationslauf durchführt.

Die Stichprobe der Ausgangsgröße wird ausgewertet, indem man beispielsweise Schätzwerte für den Mittelwert und die Standardabwiechung oder die Häufigkeitsverteilung der Ausgangsgröße zu einem Zeitpunkt t (oder auch mehreren) ermittelt.

Im einfachsten Fall sind nur statistische Kenngrößen der Ausgangsfunktionen gefragt - beispielsweise der Erwartungswert. Oft können die Ausgangsprozesse als stationär angesehen werden. Der Erwartungswert der Ausgangsgröße ist dann für alle Zeiten gleich. Es geht dann also nur noch darum, einen Zahlenwert zu bestimmen.

Im Zusammenhang mit der stochastischen Simulation treten immer wieder folgende Teilaufgaben auf:

1. *Erzeugung von Zufallszahlen* zu vorgegebenen Verteilungen. Diese Aufgabe fällt bei der Zufallsauswahl der Eingangsgrößen und Parameter je Simulationslauf an.
2. *Ergebnisbeurteilung*. Hier geht es darum, zu bestimmen, wie genau der durch die Stichprobe ermittelte Mittelwert dem tatsächlichen Mittelwert entspricht.

Auf diese beiden Teilaspekte der stochastischen Simulation wird in den folgenden beiden Unterabschnitten eingegangen.

Zufallsgeneratoren

Echte Zufallszahlen lassen sich auf Digitalrechnern wegen der Determiniertheit der Automaten (Algorithmen) nicht erzeugen. Wegen des begrenzten Speichers ist außerdem der Zustandsraum endlich und die erzeugte Zahlenfolge periodisch. Gute Zufallszahlengeneratoren

erzeugen Zahlenfolgen mit großer Periode und guten statistischen Eigenschaften (Fisz, 1976; Knuth, 1981).

In jeder Programmiersprache ist es auf relativ einfache Weise möglich, Zufallswerte zu generieren, die auf dem Intervall [0, 1) als gleichverteilt angesehen werden können. Die Zufallsvariable dieses Zufallszahlengenerators sei hier G genannt.

Nun werden Verfahren besprochen, mit denen sich Zufallszahlen erzeugen lassen, die eine bestimmte vorgegebene Verteilung haben.

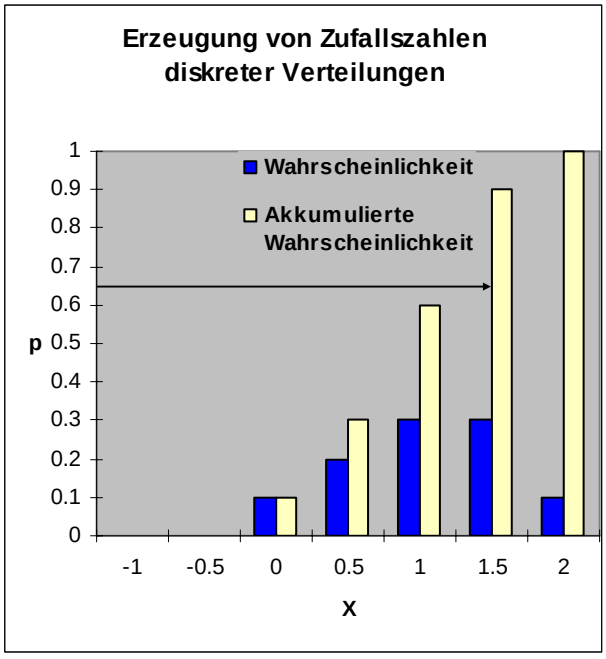
Allgemeine Methode zur Erzeugung von Zufallszahlen

Die Verteilungsfunktion der zu erzeugenden Zahlen sei F. Die Umkehrfunktion F^{-1} dieser Funktion möge existieren. Dann erhält man Zufallswerte mit der gewünschten Verteilung, indem man die gleichverteilten Werte in die Umkehrfunktion einsetzt: $X = F^{-1}(G)$. Beweis: Die Wahrscheinlichkeit von $X < x$ bzw. $F^{-1}(G)<x$ ist genau so groß, wie die des Ausdrucks $G<F(x)$. Und diese Wahrscheinlichkeit ist, da G gleichverteilt ist, gleich $F(x)$.

Beispiel: Die Verteilungsfunktion F der *Exponentialverteilung* ist gegeben durch $F(t) = 1 - e^{-t/\tau}$. Daraus folgt $F^{-1}(G) = -\tau \cdot \ln(1-G)$. Das heißt: Man erhält exponentialverteilte Zufallszahlen, indem man im Intervall (0, 1] gleichverteilte Zufallszahlen logarithmiert und den Betrag dieser Werte mit dem Erwartungswert τ der Exponentialverteilung multipliziert.

Methode 1 für diskrete Verteilungen

Auch bei den praktisch wichtigen empirischen Verteilungen lässt sich die Methode (leicht modifiziert) anwenden. Die *diskrete Verteilung* möge in Form eines Balkendiagramms der Summenfunktion (akkumulierte Wahrscheinlichkeiten) vorliegen.



Die Erzeugung von Zufallszahlen X, die dieser diskreten Verteilung entsprechen, geschieht durch Umkehrung der Summenfunktion. Ausgehend von einem Wert G der gleichverteilten Zufallsgröße erzeugt man den zugehörigen Wert X folgendermaßen: Man trägt den Wert von G auf der Ordinate ab (eingezeichnet ist $G = 0.75$) und geht horizontal nach rechts, bis man auf den ersten Balken der Summenfunktion trifft. Der zugehörige Abszissenwert ist die gesuchte Zufallszahl X (hier: X=1.5).

Methode 2 für diskrete Verteilungen

Ein sehr einfaches Verfahren ergibt sich, wenn sich die Wahrscheinlichkeiten p_i einer diskreten Verteilung - näherungsweise - als rationale Zahlen k_i/K mit nicht zu großem K darstellen lassen. Dann wird man einen K-dimensionalen Vektor definieren, in dem der Wert x_i, dessen Wahrscheinlichkeit gleich k_i/K ist, genau k_i mal als Komponentenwert erscheint. Die Erzeugung der Zufallszahl geschieht dann dadurch, dass man unter den Komponenten des Vektors eine zufällige Auswahl trifft, mit einer für alle Komponenten gleichen Wahrscheinlichkeit.

Erzeugung normalverteilter Zufallszahlen

Normalverteilte Zufallsvariable kann man nach einer Methode erzeugen, die den zentralen Grenzwertsatz der Statistik ausnutzt. Dieser Satz beinhaltet, dass die Summe von N voneinander unabhängigen Zufallsvariablen X_i (i = 1, 2, ..., N), die alle dieselbe Verteilung besitzen, näherungsweise normalverteilt ist. Die Normalverteilung wird umso besser erreicht, je größer N ist.

Seien Erwartungswert und Standardabweichung der X_i gleich μ bzw. σ. Die Summe $Y_N = X_1 + X_2 + ... + X_N$ hat dann den Mittelwert $N\mu$ und die Streuung $N\sigma^2$.

Durch lineare Transformation der Zufallsvariablen Y_N kann man folglich näherungsweise normalverteilte Zufallsvariablen mit vorgegebenen Parametern (Mittelwert und Standardabweichung) bekommen.

Eine einfacher Zufallszahlengenerator arbeitet folgendermaßen: Er addiert zwölf gleichverteilte Zufallszahlen und zieht von dieser Summe die Zahl 6 ab. Die so ermittelten Zufallszahlen sind näherungsweise (0, 1)-normalverteilt.

Exakt und dabei auch noch effizienter sind die Zufallszahlengeneratoren von Box und Muller. Eigentlich liefern diese Generatoren in jedem Berechnungsgang sogar zwei voneinander unabhängige Zufallszahlen. Wir wollen hier nur jeweils einen der beiden Werte nutzen. Auch dann noch sind die Generatoren mehr als zweimal schneller als der einfache Generator.

Eine Version des Algorithmus wird im Buch von Bratley, Fox und Schrage (1987, S. 161 f.) hergeleitet. Sie beruht auf Methode der Umkehrfunktion. Die Verteilungsfunktion der Normalverteilung ist formelmäßig nicht invertierbar. Der Weg geht über die Verteilungsdichte zweier unabhängiger (0, 1)-normalverteilter Zufallsvariablen X und Y. Mit dx und dy bezeichnen wir hier einmal die (endlichen) Längen von hinreichend kleinen Abschnitte der x- und der y-Achse. Für die Wahrscheinlichkeit, dass die zweidimensionale Zufallsvariable (X, Y) in ein Flächenstück der Kantenlängen dx und dy fällt, gilt näherungsweise die Formel

$$P(x \le X < x + \mathrm{d}x,\ y \le Y < y + \mathrm{d}y) = \frac{1}{2\pi} \cdot e^{-\frac{x^2+y^2}{2}}\,\mathrm{d}x\mathrm{d}y\,.$$

Wir stellen jeden Zufallspunkt der Ebene jetzt durch seine – ebenfalls zufälligen – Polarkoordinaten dar: $X = R \cos \Phi$, $Y = R \sin \Phi$. Ein kleines Flächenstück mit dem Radiusabschnitt dr und der Winkeldifferenz dφ im Abstand r vom Ursprung hat die Fläche $r\,\mathrm{d}r\,\mathrm{d}\varphi$. Die Wahrscheinlichkeit dafür, dass der Zufallspunkt in dieses Flächenstückchen fällt, ist gleich

$$P(r \le R < r + \mathrm{d}r,\ \varphi \le \Phi < \varphi + \mathrm{d}\varphi) = \frac{1}{2\pi} r \cdot e^{-\frac{r^2}{2}}\,\mathrm{d}r\mathrm{d}\varphi\,.$$

Offenbar sind die beiden Variablen R und Φ ebenfalls unabhängig voneinander. R hat die Verteilungsdichte $r \cdot e^{-\frac{r^2}{2}}$ und Φ ist auf dem Intervall [0, 2π) gleichverteilt. Obwohl die Verteilungsdichte von R komplizierter aussieht als die von X, lässt sich jetzt die Verteilungsfunktion in geschlossener Form angeben. Die Verteilungsdichte besitzt die Stammfunktion $-e^{-\frac{r^2}{2}}$, wie man leicht nachrechnet. Das Integral der Verteilungsdichte über das Interval [0, r) ist damit gleich $1 - e^{-\frac{r^2}{2}}$. Das ist die Verteilungsfunktion von R. Nach der Methode der Umkehrfunktion erhält man mit der auf dem Intervall [0, 1) gleichverteilten Zufallsvariablen G die folgende Realisierungsmöglichkeit $R = \sqrt{-2\ln(1-G)}$. Wenn keine Gefahr besteht, dass der Zufallszahlengenerator eine null fabriziert, kann man 1-G noch durch G ersetzen.

Die Zufallsvariable X lässt sich mit den zwei gleichverteilten und voneinander unabhängigen Zufallsvariabeln G_1 und G_2 demnach folgendermaßen erzeugen:

$$X = \cos(2\pi G_1) \cdot \sqrt{-2\ln(1-G_2)}$$

Für die Variable Y ist nur cos durch sin zu ersetzen. Damit erhalten wir eine erste Java-Version eines Generators für (0, 1)-normalverteilte Zufallszahlen.

```
static public double drawing1() {
  return Math.cos(2*Math.PI*Math.random())
         *Math.sqrt(-2*Math.log(1-Math.random()));
}
```

Die Berechnung der Cosinusfunktion lässt sich umgehen (Knuth, 1981, S. 117 f.). Dazu besorgt man sich eine zweidimensionale Zufallsvariable (V_1, V_2), die auf dem Einheitskreis gleichverteilt ist. Das geht einfach, indem man zwei auf dem Intervall [-1, 1) gleichverteilte Zufallsvariablen betrachtet und auf x- und y-Achse anträgt. Dann sondert man alle die Werte aus, die nicht in das Innere des Einheitskreises fallen (Zurückweisungstechnik). Man braucht also einen kleinen Überschuss an Zufallszahlen. Die Anzahl der benötigten Zufallszahlen vergrößert sich bei dieser Variante des Algorithmus um den Faktor $4/\pi$; das ist ein Mehr von etwa 27 %. Die Zufallsvariable $S = V_1^2 + V_2^2$ ist auf dem Intervall [0, 1) gleichverteilt. Diesen Wert können wir anstelle von G_2 in die obige Formel einsetzen. Anstelle des Wertes $\cos(2\pi G_1)$ nehmen wir $V_1 / \sqrt{S}$. Das liefert eine zweite Java-Version des Zufallzahlengenerators.

```
static public double drawing2() {
  double V1, V2, S;
  do {
    V1=2*Math.random()-1;
    V2=2*Math.random()-1;
    S=V1*V1+V2*V2;
  } while (1<=S);
  return V1*Math.sqrt(-2*Math.log(1-S)/S);
}
```

Eine Zeitmessung erbrachte (in einer bestimmten Laufzeitumgebung) eine um etwa 14 % verbesserte Zeiteffizienz gegenüber der ersten Version.

Ergebnisbeurteilung: Vertrauensintervalle

Wir haben für die Ergebnisvariable X die N Stichprobenwerte $x_1, x_2, ..., x_N$ erhalten. Wir interessieren uns für den Erwartungswert $\mu = E[X]$ der Ergebnisvariablen und nehmen das arithmetische Mittel m der Stichprobenwerte als Schätzwert für μ:

$$m = (\sum_{i=1}^{N} x_i) / N$$

Es fragt sich nun, inwieweit man damit rechnen kann, dass $m \approx \mu$ ist, und welchen Einfluss der Stichprobenumfang auf die Güte des Schätzwerts hat. Präziser ausgedrückt geht es darum, den Stichprobenumfang N zu bestimmen, so dass mit einer vorgegebenen *Sicherheit* Q (in %) der absolute Fehler nicht größer als die *Fehlergrenze* E ist:

$$|m - \mu| \le E$$

Der Schätzwert m ist selbst eine Zufallsvariable: Wir fassen die x_i als Realisierungen von Zufallsvariablen X_i auf, die alle dieselbe Verteilung wie X besitzen, und die voneinander statistisch unabhängig sind.

Der Erwartungswert der Zufallsvariablen m ist gleich μ. Der Schätzwert hat demnach wenigstens den richtigen Erwartungswert. Die Standardabweichung ist gleich $\sigma / \sqrt{N}$. Sie nimmt

mit wachsendem N ab. Das heißt, dass die Schätzwerte umso weniger streuen, je größer die Stichprobe ist. Die Sicherheit, dass m in der Nähe von μ liegt, nimmt mit dem Stichprobenumfang zu.

Diese Aussage lässt sich folgendermaßen quantifizieren. In der Praxis ist N meist genügend groß, so dass aufgrund des zentralen Grenzwertsatzes die Zufallsvariable $\sqrt{N}(m-\mu)/\sigma$ als näherungsweise (0, 1)-normalverteilt angesehen werden kann. Die Dichtefunktion der Normalverteilung mit dem Erwartungswert 0 und der Standardabweichung 1 ist gegeben durch

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2}} .$$

Der Wert t wird zu einem vorgegebenen Q so bestimmt, dass

$$Q = \int_{-t}^{t} \Phi(x)\mathrm{d}x$$

gilt. Die wichtigsten Wertepaare t und Q sind in der Tabelle 2.1 aufgelistet. Die genannten Zusammenhänge führen zur Formel

$$P(|m-\mu| \le t\,\sigma / \sqrt{N}) = Q.$$

Der absolute Fehler $|m-\mu|$ ist mit der Wahrscheinlichkeit Q durch $E = t\,\sigma/\sqrt{N}$ begrenzt. Das heißt: Der korrekte Wert μ liegt mit der *Sicherheit* (auch: Vertrauenswahrscheinlichkeit) Q im Intervall $[m - E, m + E]$. Dieses zufällige Intervall heißt *Konfidenzintervall* oder *Vertrauensintervall*. Für das Konfidenzintervall schreiben wir manchmal auch kurz $m \pm E$.

Gebräuchliche Werte für Q und t

Q	t
60 %	0.84
68.27 %	**1**
80 %	1.28
90 %	1.64
95 %	1.96
95.5 %	**2**
98 %	2.33
99 %	2.58
99.7 %	**3**

Da $\sigma/\sqrt{N}$ gleich der Standardabweichung der Schätzgröße m ist, bezeichnet man das Konfidenzintervall $m \pm t\,\sigma/\sqrt{N}$ auch als $t\sigma$-Bereich. Zur statistischen Sicherheit von 95.5 Prozent gehört demnach der 2σ-Bereich.

Die Methode der Bestimmung des Vertrauensintervalls hat noch einen Schönheitsfehler: Wir kennen σ im Allgemeinen nicht. Wir verschaffen uns für σ einen Schätzwert s, indem wir dieselbe Stichprobe wie zur Bestimmung von μ benutzen. Wir setzen

$$s^2 = \frac{1}{N-1} \cdot \sum_{i=1}^{N} (x_i - m)^2 = \frac{1}{N-1} \cdot (\sum_{i=1}^{N} {x_i}^2 - Nm^2)$$

und stellen fest, dass s^2 den Erwartungswert σ^2 besitzt.

Eigentlich ist s auch nur ein Schätzwert für σ. Erneut stellt sich die Frage nach dem Vertrauensintervall. Ein Teufelskreis kündigt sich an. Wie man ihm entkommen kann, findet man in den Lehrbüchern der Statistik. Hier nur so viel: Bei Stichprobengrößen ab etwa 40 darf man s anstelle von σ in die Formel für die Fehlergrenze einsetzen: $E = t\,s/\sqrt{N}$ für $N \ge 40$.

Für die Genauigkeitsbetrachtung dieses Abschnitts wurde vorausgesetzt, dass

- die Stichprobenwerte voneinander statistisch unabhängig sind,
- allen Stichprobenwerten dieselbe Verteilung zugrunde liegt,
- der Schätzwert m (näherungsweise) normalverteilt ist, und
- s ein guter Schätzwert für σ ist.

Die letzten beiden Voraussetzungen sehen wir als erfüllt an, wenn $N \ge 40$.

Varianzreduktion

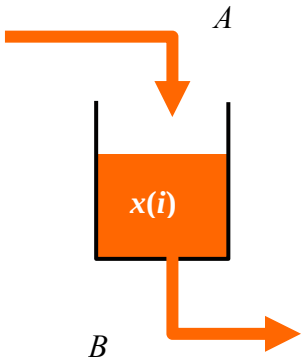
Techniken der Varianzreduktion dienen dazu, die Genauigkeit des Ergebnisses bei gleicher Stichprobenzahl zu erhöhen.

Man wählt solche Zielgrößen aus, die bei unverändertem Maßstab eine möglichst kleine Varianz σ^2 haben. Dann wird das Konfidenzintervall entsprechend klein.

Eine gute Gelegenheit, die Varianz zu reduzieren, ergibt sich beispielsweise, wenn es darum geht, zwei Ergebnisvariablen X_1 und X_2 miteinander zu vergleichen. Wenn diese in etwa in derselben Weise von den zufälligen Einflüssen abhängen, dann geht man am besten zur Differenz dieser Größen über und wählt als Zielgröße die Variabel $X = X_1 - X_2$. Diese hat oftmals eine deutlich geringere Varianz als die ursprünglichen Größen. Diese Methode läuft darauf hinaus, dass man in den Simulationsläufen zur Ermittlung der Stichprobenwerte für X_1 und X_2 gemeinsame Zufallszahlen zugrunde legt (Bratley, Fox und Schrage, 1987), (McGeoch, 1992).

Die Methode der Stapelmittelwerte

Beispiel: Der Lagerbestand eines bestimmten Gutes am Ende eines Tages ist die zu untersuchende Größe. Die Folge $x(0)$, $x(1)$, $x(2)$, ... dieser Werte bildet einen stochastischen Prozess.



Die je Tag eintreffende Menge ist die Realisierung einer Zufallsvariablen A. Die täglich nachgefragten Güter werden durch die Zufallsvariable B beschrieben.

Bei dem betrachteten Prozess hängen zeitlich eng benachbarte Werte stärker voneinander ab als weiter voneinander entfernte. Die Einzelwerte kann man also nicht als statistisch unabhängige Stichprobenwerte ansehen.

Dennoch lässt sich die Genauigkeitsbewertung mit Konfidenzintervallen auch auf diesen und ähnlich gelagerte Fälle übertragen.

Wir bilden die Mittelwerte $y(i)$ von jeweils b aufeinanderfolgenden Werten der ursprünglichen Folge. Für $y(i)$ werden die Werte ab dem Index ib genommen: $x(ib)$, $x(ib+1)$, $x(ib+2)$, ..., $x((i+1)b-1)$. Die $y(i)$ heißen *Stapelmittelwerte,* und b ist die Stapelgröße:

$$y(i) = \frac{1}{b} \sum_{j=ib}^{(i+1)b-1} x(j)$$

Sei a die Anzahl der Stapel. Es werden also insgesamt $n = a \cdot b$ Werte des stochastischen Prozesses benötigt: $x(0)$, $x(1)$, $x(2)$, ..., $x(n-1)$. Die a Stapelmittelwerte $y(0)$, $y(1)$, ..., $y(a-1)$ gelten nun als Stichprobenwerte. Das arithmetische Mittel dieser Stapelmittelwerte ist natürlich dasselbe wir das der ursprünglichen Folge.

Das Konfidenzintervall für den Mittelwert μ wird nun auf der Basis der Stichprobenwerte $y(i)$ ermittelt. Dabei wird unterstellt, dass bei ausreichend großem b die Abhängigkeit der Stichprobenwerte $y(i)$ voneinander vernachlässigbar gering ist. Es ist sinnvoll, a und b etwa gleich groß zu wählen.

Die Methode der Stapelmittelwertbildung und weitere Techniken der Ergebnisbeurteilung sind im Buch von Bratley, Fox und Schrage (1987) zu finden.

Übungen

Übung 1: Schreiben Sie eine öffentliche Klasse Statistics. Gegeben sind ein Auszug aus der Java-Dokumentation und die Spezifikationen (Funktionsbeschreibungen) der Konstruktoren und Methoden. Legen Sie für Ihre Klasse ein eigenes Verzeichnis namens `drawings` an.

Dort können Sie alle Klassen unterbringen, die mit Statistiken und mit der Erzeugung von Zufallszahlen zu tun haben, beispielsweise auch die bereits konstruierte Klasse `Erlang`. Zur Abgrenzung des Namensraumes fassen Sie die Klassen des Verzeichnisses zu einem `package` gleichen Namens zusammen. Dazu fügen Sie in den Kopf jeder Datei die Paket-Deklaration „`package Drawings;`" ein. Es folgt ein Auszug aus der Java-Dokumentation der Klasse Statistics.

```
java.lang.Object
  |
  +--Drawings.Statistics
```

Constructor Summary	
Statistics()	
Statistics(int b)	

Method Summary	
void	**add**(double z)
double	**meanVal**()
int	**nBatches**()
double	**stanDev**()

`new Statistics()` liefert die Referenz auf ein neues Statistik-Objekt und setzt dessen Stapelgröße auf den Wert 1.

`new Statistics(b)` liefert die Referenz auf ein neues Statistik-Objekt und setzt dessen Stapelgröße auf den Wert b.

`add(z)` erfasst den nächsten Wert z. Verwaltet die Stapel und aktualisiert die Akkumulatoren für die Berechnung der statistischen Kenngrößen.

`meanVal()` ist der Mittelwert aller bis dahin in kompletten Stapeln erfassten Werte.

`stanDev()` ist die Standardabweichung des geschätzten Mittelwerts.

`nBatches()` ist die Anzahl aller bis dahin komplettierten Stapel.

Übung 2: Schreiben Sie ein Simulationsprogramm für die Lagersimulation und wenden Sie bei der Ergebnisbeurteilung die Methode der Stapelmittelwerte an. Experimentieren Sie mit verschiedenen Stapelgrößen und notieren Sie Ihre Beobachtungen. Wählen Sie folgende Verteilungsfunktionen für die angelieferten und die ausgelieferten Gütermengen: Die Variable *A* sei (10, 2)-normalverteilt und *B* sei (11, 3)-normalverteilt. Der Kern des Programmes könnte so aussehen:

```
L=0;
for(i=0; i<n; i++){
  L+=a-b;
  if(L<0)L=0;
  x[i]=L;
}
```

Übung 3: Schreiben Sie eine öffentliche Klasse `Erlang` zur Erzeugung erlangverteilter Zufallszahlen. Bei Aufruf der Methode `Erlang.drawing(r, m)` soll eine Zufallszahl zurückgegeben werden, die erlangverteilt ist mit dem Parameter `r` und die den Erwartungswert

`m` besitzt. Der Fall `r=0` steht für die deterministische Verteilung. In diesem Fall soll also der Wert `m` zurückgegeben werden.

```
public class Erlang {
  static public double drawing(int r, double m) { ... }
}
```

5 Objektorientierte Software-Entwicklung

An einem Beispiel wird in die objektorientierte Software-Konstruktion eingeführt. Gegenstand der Lektion sind

- grundlegende Java-Sprachelementen sowie
- grundlegende Elemente der Unified Modeling Language (UML).

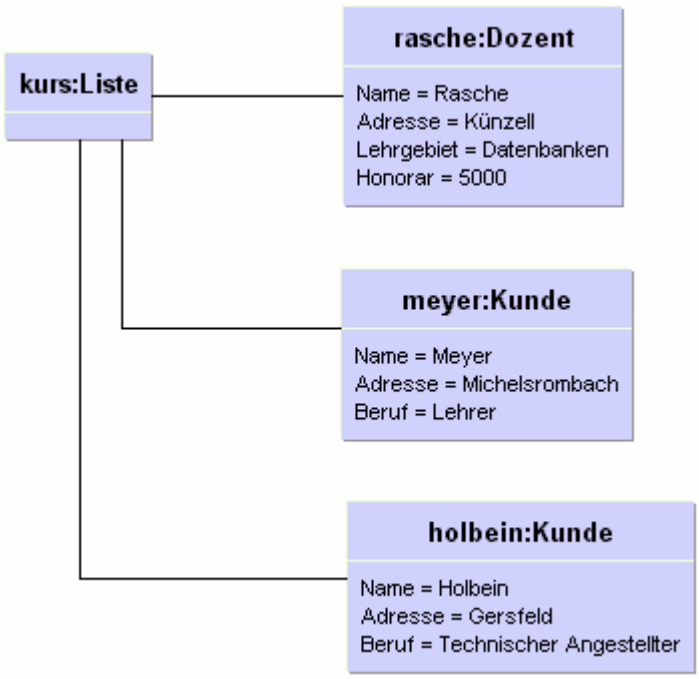
Wie beim einführenden Beispiel ist der Lernende aufgefordert, das Beispiel nachzuvollziehen und explorativ zu manipulieren.

Die Aufgabenstellung: Lehrgangsverwaltungsprogramm

Das zu erstellende Programm dient der Verwaltung der Daten von Teilnehmern (Kunden) und Dozenten eines Lehrgangs. Funktion: Erfassung der Daten von Kunden, das sind die Teilnehmer des Kurses, und Dozenten. Nach Beendigung der Eingabe soll die Liste der Kunden und Dozenten auf dem Bildschirm ausgegeben werden. (Siehe auch die Einführung "Grundlagen der objektorientierten Software-Entwicklung" auf der CD zum "Lehrbuch Grundlagen der Informatik" von Helmut Balzert, 1999).

Analyse: Objektdiagramm

Die Ergebnisse der Analyse und des Designs sind Objekt- und Klassendiagramme, die nach den Regeln der Unified Modeling Language (UML) gestaltet werden. Die Diagramme werden mit dem Werkzeug GO, das dem oben angesprochenen Lehrbuch beigefügt ist, erstellt.



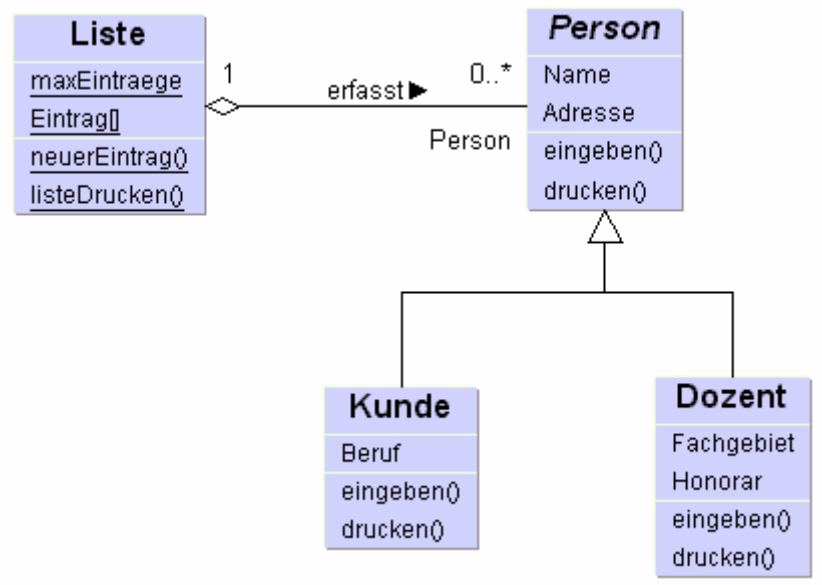
Das Objektdiagramm stellt beispielhaft dar, welche *Attribute* den *Objekten* des Systems zukommen und welche Beziehungen zwischen den Objekten bestehen. Das nebenstehende Objektdiagramm zeigt die Objekte meyer und holbein vom Typ Kunde sowie das Objekt rasche vom Typ Dozent. Das Objekt kurs vom Typ Liste enthält Bezüge auf diese Objekte.

Objektbezeichner beginnen mit einem Kleinbuchstaben, Klassenbezeichner mit einem Großbuchstaben.

Analyse: OOA-Klassendiagramm

Das OOA-Klassendiagramm ist eine Darstellung der Objekttypen (Klassen), und deren Beziehungen zueinander. Die Klassen sind als dreigeteilte Rechtecke gezeichnet. Darin stehen jeweils

1. der Name der *Klasse*,
2. die *Attribute* (Variablen) und
3. die *Methoden* (Funktionen).



Unterstrichen sind die *statischen* (static) Attribute und Methoden. Diese gehören zur Klasse und sind auch ohne Objekt existent und erreichbar.

Kursiv geschriebene Klassennamen bezeichnen *abstrakte* Klassen. Das sind Klassen ohne eigene Realisierungen (Objekte).

Im Klassendiagramm bleiben die Beziehungen zwischen den Objekten sichtbar. An den Verbindungslinien stehen Angaben über die mögliche Anzahl der Objekte und die Art der Verbindung. Eine offene Raute zeigt eine *Aggregation* an; sie ist auf der Seite desjenigen Objekttyps angebracht, der die "Verantwortung für das Ganze" trägt.

Verbindungen mit einem offenen Dreieck stehen für eine Vererbungsbeziehung. Das Dreieck zeigt auf die *Oberklasse*. *Unterklassen erben* die Attribute und Methoden der Oberklasse und fügen ihnen eventuell weitere hinzu. Methoden der Unterklassen *überschreiben* Methoden der Oberklasse, vorausgesetzt sie haben dieselbe *Signatur* (identischen Namen und dieselben Parametertypen) und denselben Typ des Rückgabewertes.

Den Referenzen oder Zeigern (Pointern) der Klasse Liste ist nicht anzusehen, ob sie gerade auf einen Kunden oder auf einen Dozenten verweisen. Neben dem durch die Deklaration festgelegten statischen Typ haben solche Referenzen noch einen *dynamischen Typ*. Dieser ist gegeben durch den Typ des Objekts, auf das gerade verwiesen wird; er ist veränderlich. Dieser *Polymorphismus* von Referenzen ist von zentraler Bedeutung für die objektorientierte Programmierung.

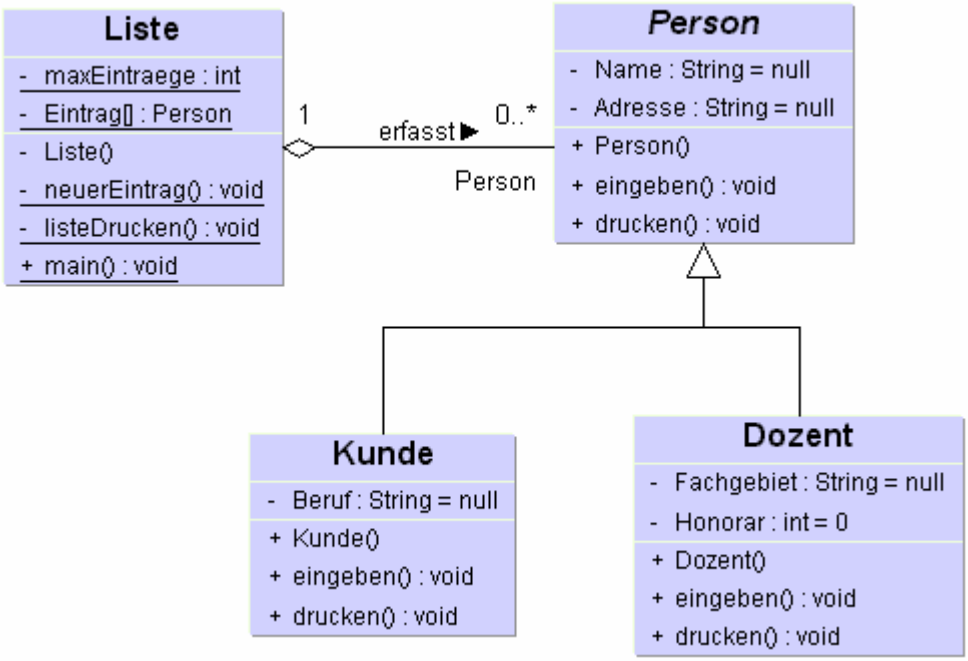
Der Aufruf überschriebener Methoden richtet sich grundsätzlich nach dem dynamischen Typ der Referenz. Anders ausgedrückt: Der Methodenaufruf einer überschriebenen Methode hängt vom Typ des Objekts ab und nicht etwa vom statischen Typ der Referenz.

Design: OOD-Klassendiagramm

Das OOD-Klassendiagramm enthält weitere Präzisierungen. Das sind insbesondere

1. die Typen der Attribute und deren Initialisierung,
2. die Typen der Parameter und Rückgabewerte der Methoden, und
3. die Festlegungen gemäß *Geheimnisprinzip*: Private Attribute und Methoden sind mit einem Minuszeichen markiert. Öffentliche (public) Elemente erhalten ein Pluszeichen. Private Elemente sind nur innerhalb einer Klasse und innerhalb deren Objekte sichtbar. Die öffentlichen Elemente bilden die Schnittstelle nach außen.

4. Konstruktoren unterscheiden sich von den Methoden dadurch, dass es keine Rückgabewerte gibt. Sie werden bei der Generierung von Objekten aufgerufen und dienen hauptsächlich der Initialisierung der Attribute des Objekts. Konstruktoren werden durch den Klassennamen bezeichnet: Der Konstruktor der Klasse Kunde beispielsweise ist Kunde()[4]. Konstruktoren werden nicht vererbt und können folglich auch nicht - im Sinne des Polymorphismus - überschrieben werden.



Java-Programmtexte

```
IO.java
import java.io.*;

public class IO {
  static byte buf[]=new byte[129];

  public static String inStr() {
    int i=0;
    try {i= System.in.read(buf);}
    catch(IOException e){
      System.out.println(e.toString());
      e.printStackTrace();
    }catch(NumberFormatException e){
      System.out.println(e.toString());
      e.printStackTrace();
    }
    return new String(buf, 0, i);
  }

  public static double inDbl() {return new Double(inStr()).doubleValue();}
  public static void outStr(String s) {System.out.print(s);}
  public static void outInt(int s) {System.out.print(Integer.toString(s));}
```

[4] Im Beispiel hätte man die Konstruktoren nicht hinschreiben müssen, da sie nichts besonderes zu tun haben. Der Compiler fügt diese Default-Konstruktoren selbsttätig hinzu. Bei Abarbeitung eines Konstruktors wird als erstes der Konstruktor der Superklasse aufgerufen. Fehlt ein solcher Aufruf, wird der Default-Konstruktor der Superklasse aufgerufen. Demnach kann die Konstruktohr-Deklaration „public Kunde() {}“ weggelassen werden oder aber auch ausführlicher „public Kunde() {super();}“ geschrieben werden.

```
}

// Person.java
public abstract class Person {
  private String Name = null;
  private String Adresse = null;

  public  Person() {}

  public void eingeben() {
    IO.outStr("? Name: ");
    Name=IO.inStr();

    IO.outStr("? Adresse: ");
    Adresse=IO.inStr();
  }

  public void drucken() {
    IO.outStr("! Name: "); IO.outStr(Name);
    IO.outStr("! Adresse: "); IO.outStr(Adresse);
  }

}

// Kunde.java
public class Kunde extends Person {
  private String Beruf = null;

  public  Kunde() {}
  public void eingeben() {
    super.eingeben();
    IO.outStr("? Beruf: ");
    Beruf=IO.inStr();
  }
  public void drucken() {
    super.drucken();
    IO.outStr("! Beruf: "); IO.outStr(Beruf);
  }
}

// Dozent.java
public class Dozent extends Person {
  private String Fachgebiet = null;
  private int Honorar = 0;

  public  Dozent() { }

  public void eingeben() {
    super.eingeben();
    IO.outStr("? Fachgebiet: "); Fachgebiet=IO.inStr();
    IO.outStr("? Honorar: "); Honorar=(int)IO.inDbl();
  }

  public void drucken() {
    super.drucken();
    IO.outStr("! Fachgebiet: "); IO.outStr(Fachgebiet);
    IO.outStr("! Honorar: "); IO.outInt(Honorar); IO.outStr("\n");
  }

}

/** Liste.java, Timm Grams, 5.9.2001
Hauptklasse der Java-Version des Programms zu "Tutorial - ein
Lehrgangsverwaltungssystem".

Das Projekt dient der Demonstration von Grundkonzepten der
```

```
objektorientierten Programmierung.
*************************************************************************/
public class Liste
{
  static private int maxEintraege=200;

  /** Vektor der Listeneintraege */
  static private Person[] Eintrag=new Person[maxEintraege];

  static private void neuerEintrag(char c) {
   int i=0;  Person p;
   while (i<maxEintraege&&Eintrag[i]!=null) i++;
   switch (c) {
     case 'd': Eintrag[i]=p=new Dozent(); p.eingeben(); break;
     case 'k': Eintrag[i]=p=new Kunde(); p.eingeben(); break;
   }
  }

  static private void listeDrucken() {
    int i;
    for (i=0; i<maxEintraege; i++) if (Eintrag[i]!=null)
    Eintrag[i].drucken();
  }

  public static void main (String[] args) {
    char c;
    IO.outStr("\nLehrgangsverwaltung - ein Tutorial der ooP");
    do {
      IO.outStr("\n? <D>ozent, <K>unde, <S>chluss ");
      c=Character.toLowerCase(IO.inStr().charAt(0));

      if (c=='k'|| c=='d') neuerEintrag(c);
      else if (c!='s') IO.outStr("\nVertippt?");
    } while(c!='s');
    IO.outStr("\n\n! Ausgabe der Liste\n");
    listeDrucken();
  }
}
```

Übung

Erweitern Sie das Lehrgangsverwaltungsprogramm, indem Sie als Unterklasse zur Klasse `Kunde` die Klasse `Sonderkunde` einführen. Sonderkunden bekommen Rabatt. Als zusätzliches Attribut haben Sie den Ihnen zustehenden Rabatt. Demonstrieren Sie die Funktionsfähigkeit Ihres Programmes.

6 Grundlagen der Wartesysteme

Wozu wird die Theorie der Wartesysteme gebraucht? Für die Validierung, denn: die rationale Einstellung zum Computer ist grundsätzliches Mißtrauen.

Die Theorie gestattet die Überprüfung von Sonderfällen und von bestimmten (der symbolischen Berechnung zugänglichen) Kenngrößen. Wenn das Modell für Sonderfälle und für diese ausgezeichneten Kenngrößen stimmt, so ist das noch keine Garantie für die Korrektheit. Aber umgekehrt bieten diese Tests die Chance, Fehler im Modell zu finden.

Bezeichnungen und Kenngrößen für einfache Wartesysteme

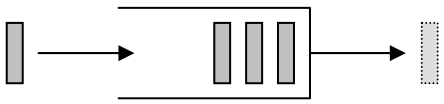
A/B/m bezeichnet ein einfaches Wartesystem mit einer durch A spezifizierten Verteilung der *Zwischenankunftszeit* und mit einer durch B spezifizierten Verteilung der *Bediendauer*. Die Zahl der Bedienungseinheiten ist m. Kennbuchstaben für tatsächliche Verteilungen sind

M Markoffscher (poissonscher) Prozess

D Deterministische (konstante) Zwischenzeiten

G Generelle Verteilung der Zwischenzeiten

E_r Erlang-Verteilung mit dem Parameter r



M/D/1 steht demnach für ein Wartesystem mit poissonschem Ankunftsprozess, konstanter Bediendauer und einer Bedienungseinheit.

Bezeichnungen:

t_n Zwischenankunftszeit zwischen dem Kunden n-1 und n.

T Zufallsvariable der Zwischenankunftszeiten (allen Zwischenankunftszeiten gemeinsam)

x_n Bediendauer für den n-ten Kunden

X Zufallsvariable der Bediendauer (allen Bediendauern gemeinsam)

u_n n-te Bilanz: $u_n = x_n - t_{n+1}$

U Bilanzvariable: $U = X-T$

w_n Wartezeit des n-ten Kunden

W_n Zufallsvariable der Wartezeit des n-ten Kunden

Kenngrößen:

a mittlere Zwischenankunftszeit: $a = E[T]$

$1/a$ Ankunftsrate

b mittlere Bediendauer: $b = E[X]$

σ_b Standardabweichung der Bediendauer

w mittlere Wartezeit: $w = \lim_{n \to \infty} E[W_n]$

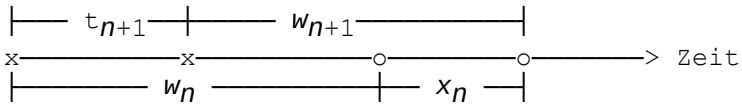
N mittlere Anzahl wartender Kunden

p_0 Wahrscheinlichkeit für freien Schalter

ρ Auslastung: $\rho = b/a$ (Zeitanteil, zu dem der Bedienschalter belegt ist)

Einige Sätze aus der Theorie der Wartesysteme

Die grundlegenden Zusammenhänge zwischen den Zeiten zeigt das folgende Diagramm für das G/G/1-Wartesystem. (Mit x sind Ankunftsereignisse und mit o sind die Ereignisse des Abfertigungsbeginns markiert.)



Es gilt die *Rekursionsbeziehung für Wartezeiten:*

$$w_{n+1} = \max(0, w_n + x_n - t_{n+1}) = \max(0, w_n + u_n).$$

Es gilt der *Satz von Little* (Kleinrock, I, 1975, S. 10 ff.):

$$N = w/a.$$

Den Satz von Little macht man sich am besten so klar: Die mittlere Anzahl N der Kunden in der Warteschlange ist gleich der Anzahl der Kunden, die im Mittel während der Verweilzeit eines Kunden in der Warteschlange (Mittelwert: w) eintrifft. Deren Ankunftsrate ist gleich $1/a$. Das Produkt aus Ankunftsrate und mittlerer Wartezeit ist folglich gleich der mittleren Anzahl wartender Kunden.

Die *Auslastung* ρ ist - auf lange Sicht gesehen - der Zeitanteil, zu dem der Schalter belegt ist. Das ist zugleich die (stationäre) Wahrscheinlichkeit dafür, dass der Schalter belegt ist. Die Wahrscheinlichkeit für einen freien Schalter ist dann gleich $p_0 = 1 - \rho$.

Die mittlere Wartezeit für das *M/G/1-Wartesystem* ergibt sich zu

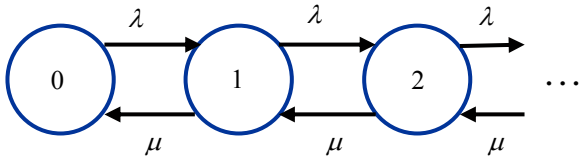
$$w = \frac{b}{2}\left(1+\frac{\sigma_b^2}{b^2}\right)\frac{\rho}{1-\rho}$$

Das ist die *Pollaczek-Khinchin-Mittelwertformel* (Kleinrock, 1975, S. 187, S. 308; Gnedenko, 1980, S. 337 ff.).

Diese Formel[5] reduziert sich für das *M/M/1-Wartesystem* auf $w = b\ \rho/(1-\rho)$ und für das *M/D/1-Wartesystem* auf $w = b/2\ \rho/(1-\rho)$.

Übung

Leiten Sie die Pollaczek-Khinchin-Mittelwertformel für das M/M/1-Wartesystem her. Legen Sie ein Markoff-Modell zu Grunde: Das System ist im Zustand k genau dann, wenn sich k Kunden im System befinden. Im Zustand k warten k-1 Kunden. Das gilt für $0<k$. Wenn kein Kunde im System ist, wartet auch keiner. Die Übergangsraten λ und μ sind die Kehrwerte von mittlerer Zwischenankunftszeit und mittlerer Bediendauer: $\lambda=1/a$ und $\mu=1/b$. Berechnen Sie die stationären Zustandwahrscheinlichkeiten p_k.



[5] Diese Formeln hat Gunter Dueck in seiner Kolumne mit Leben gefüllt (Informatik Spektrum 27 (April 2004) 2, S. 186-191): „Alles am Limit. Alles ist so organisiert, dass alle unaufhörlich arbeiten. Es geht schließlich um den Auslastungskoeffizienten. Die anderen müssen warten. Beim Arzt warten also mehr als in der ganzen Praxis arbeiten. Mehr krank als weiß. Wir zahlen 14 Prozent des Volkseinkommens für unsere Gesundheit und warten also noch einmal für 14 Prozent?“

Hilfe: Wir machen uns zunächst einmal die Systemgleichungen klar. Sei das System im Zustand k. Die Wahrscheinlichkeit, dass das System im als klein angenommenen Intervall Δt in den Zustand k+1 übergeht, ist in etwa gleich $\Delta t \cdot \lambda$, und die Wahrscheinlichkeit, dass es in den Zustand k-1 übergeht, ist in etwa gleich $\Delta t \cdot \mu$. Das kommt daher, weil die Übergangszeiten jeweils mit den entsprechenden Parametern exponentialverteilt sind.

Durch Grenzübergang kommt man auf das folgende (unendliche) System von Differentialgleichungen:

$$\dot{p}_0 = -\lambda \cdot p_0 + \mu \cdot p_1$$

$$\dot{p}_k = \lambda \cdot p_{k-1} - (\lambda + \mu) \cdot p_k + \mu \cdot p_{k+1} \text{ für } k = 1, 2, 3, \ldots$$

Außerdem haben wir noch die Grundeigenschaft der Wahrscheinlichkeiten:

$$1 = \sum_k p_k$$

Im stationären Fall sind die Ableitungen gleich null und es ergibt sich ein (unendliches) algebraisches Gleichungssystem für die Wahrscheinlichkeiten, das sich mit der Direktansatzmethode auflösen lässt. Damit hat man auch die Wahrscheinlichkeitsverteilung für die Anzahl der Wartenden. Über Erwartungswertbildung und mit Hilfe des Satzes von Little kommt man schließlich zum Ziel.

7 Ereignisorientierte Simulation

Die *ereignisorientierte Simulation* wird anhand eines Beispiels in ihren Grundzügen entwickelt. In der folgenden Lektion wird das Beispiel im Rahmen eines kleinen Projekts weiter ausgebaut. Dann werden auch die Klassen für die ereignisorientierte Simulation komplett entwickelt (soweit sie für diese Lehrveranstaltung benötigt werden).

Methode und Programmierung

Die ereignisorientierte Simulation eignet sich für Systeme, deren Größen sich vornehmlich abrupt, zu möglicherweise zufälligen Zeitpunkten ändern wie beispielsweise beim Eintreffen eines Kunden in einer Warteschlange, und die in der Zwischenzeit konstant bleiben.

Bei der *ereignisorientierten Simulation* wird das System nur zu den Ereigniszeitpunkten betrachtet. Das unterscheidet diese Art der Simulation von der *zeitschrittorientierten*, die beispielsweise bei den deterministischen Systemen der Regelungs- und Nachrichtentechnik angewendet wird und auch bei ökologischen Simulationen.

Die aufgrund eines Ereignisses sich ergebenden Größen werden neu berechnet. Danach wird zum nächsten Ereignis übergegangen und die Simulationszeit entsprechend heraufgesetzt.

Jedes Ereignis erfordert Aktionen, die von der Art des Ereignisses abhängen: Tritt ein Kunde in die Schalterhalle der Bank, muss - bei leerer Warteschlange - ein freier Bankangestellter "aktiviert" werden, oder es erfolgt eine Einreihung in die Warteschlange. Wird ein Kundengespräch beendet, dann schaut der Angestellte, ob noch Kunden warten. Ist keiner da, wird die Bedienung vorübergehend eingestellt.

Das Verlangen, die Verkopplung von Ereignissen und Aktionen nachzubilden, war in den 60-er Jahren Anlass, mit Simula 67 einen neuen Typ von Programmiersprachen zu schaffen (Dahl, Hoare, 1972). Diese werden heute *objektorientiert* genannt. Dazu gehören - neben Simula - Smalltalk, Oberon, Object Pascal, Eiffel, C++ und Java.

Bei diesen objektorientierten Sprache werden Datentypen und die auf die Daten wirkenden Operatoren (Methoden, Prozeduren, Funktionen) zu Klassen zusammengefasst. Objekte sind die konkreten ("datenhaltigen") Realisierungen dieser Klassen. Ereignisse sind Objekte mit eingetragener Aktivierungszeit, die vom Simulationsprogramm zeitfolgerichtig aktiviert werden, und die die jeweiligen Handlungsanweisungen in Gestalt von Methoden selbst mitbringen - wie ja auch ein Kunde oder ein Bankangestellter selber weiß, was er zu tun hat.

Java-Klassen für die Simulation: Für die zeitfolgerichtige Aktivierung von Ereignissen wird ein allgemein verwendbares `package EventSim` definiert. Es enthält die `Event-` und die `Simulation`-Klasse.

Objekte von Unterklassen der abstrakten `Event`-Klasse wollen wir *ereignisfähige Simulationsobjekte*, *aktive* Simulationsobjekte oder einfach nur Simulationsobjekte nennen[6]. Die `Ereignis`-Klasse ist Unterklasse der `Link`-Klasse, die eine Verkettung der aktiven Objekte zu linearen Listen (Ereignisliste, Warteschlangen) ermöglicht.

Sowohl der Listenkopf einer linearen Liste als auch die Listenelemente haben als Basis die `Link`-Klasse. Der `Link`-Methodenaufruf `in(p)` hängt das Element `p` unmittelbar an das

---

[6] Vor dem WS 04 habe ich Ereignisse (von Event abgeleitete Objekte) als Prozesse bezeichnet und die Event-Klasse hieß Process. Das geschah in Analogie zu den Bezeichnungen in Simula. Die neue Bezeichnung trifft die Sache besser. Allerdings hat man nun für die aktiven Simulationsobjekte keine prägnante Bezeichnung mehr. Aufgegeben habe ich die Bezeichnung „Prozess“ auch, weil es einen Namenskonflikt mit der grundsätzlich importierten Process-Klasse des java.lang-Packages gibt. (Auch eine Event-Klasse gibt es noch anderswo, nämlich im java.awt-Package. In diesem Fall wirkt aber die Namensraum-Abschottung an den Package-Grenzen.)

aufrufende `Link`-Element an. Der Methodenaufruf `add(p)` hängt das Listenelement `p` hinten an die Liste an; `out()` entnimmt der Liste das nächste Element und liefert als Funktionswert die Referenz auf das entnommene Element. Die boolesche Methode `empty()` sagt, ob die Liste ab dem aufrufenden Element leer ist oder nicht.

Etwas vereinfacht sieht die `Event`-Klasse so aus:

```
abstract public class Event extends Link {
   double ti;
   public double t(){return ti;}
   abstract public void run();
   public void activate_at(double t0) {...}
}
```

Der Aktivierungszeitpunkt eines Ereignisses ist `ti`. Er wird über die Methode `t()` nach außen bekannt gemacht. Das Verhalten des Simulationsobjekts wird durch Überschreiben der `run`-Methode festgelegt. Die zeitfolgerichtige Eintragung eines Ereignisses in die Ereignisliste (`Simulation.sequence`) wird durch Aufruf der `activate_at`-Methode besorgt. Sie setzt `ti` auf `t0` oder (bei zu kleinem `t0`) auf die aktuelle Simulationszeit.

Die `Simulation`-Klasse ist die Basis (Superklasse) des Hauptprogramms der Simulation. Die Ereignisliste wird von der `run`-Methode der `Simulation`-Klasse abgearbeitet. Nacheinander aktiviert sie die Ereignisse der Ereignisliste durch Aufruf von deren `run`-Methoden. Die Simulationsuhr wird entsprechend mitgeführt.

```
public class Simulation {
   static Link sequence=new Link();                          //Ereignisliste
   static double SimTime;                                   //Simulationsuhr
   public static double time(){return SimTime;}                       //Zeit
   public static void run() {
      SimTime=0;
      while (!sequence.empty){
        Event p=(Event)sequence.out();
        SimTime=p.ti;
        p.run();
      }
   }
}
```

Beispiel: Ein einfaches Wartesystem

Aufgabe: Zu simulieren ist ein sehr einfaches Wartesystem (eine Bank) mit einem poissonschen Kundenstrom, einer unbegrenzten Zahl von Warteplätzen und einem Bankangestellten als Bedieneinheit. Die Zwischenankunftszeiten der Kunden sind exponentialverteilt mit dem Mittelwert *a*. Die Bediendauer *b* ist konstant. Die mittlere Wartezeit *w* der Kunden ist per Simulation zu ermitteln.

Das M/D/1-Simulationsmodell: Die Kunden werden mit der `Customer`-Klasse modelliert und der Bediener mit der `Teller`-Klasse. Die `Customer`-Klasse enthält die Kundenwarteschlange (`queue`) und die `Teller`-Klasse - als Referenz auf den Bediener - den Schalter (`counter`). Ist der Bediener frei (`counter!=null`), dann wird er augenblicklich aktiviert. Die negative Zeit in der Aktivierungsanweisung sorgt dafür, dass der Bediener ganz vorn in der Ereignisliste einsortiert wird.

Falls noch nicht alle Kunden erzeugt worden sind, wird mit dem Aufruf `new Customer(i+1).activate_at(...)` der nächste Kunde erzeugt und zu einem Zeitpunkt aktiviert, der sich aus der aktuellen Simulationszeit plus der zufälligen (exponentialverteilten) Zwischenankunftszeit (`InterArrivalTime`) ergibt. Schließlich begibt sich der Kunde in die Warteschlange.

Der Bediener, der durch die `Teller`-Klasse modelliert wird, schaut in der Kunden-Warteschlange nach dem nächsten Kunden. Ist die Warteschlange leer, besetzt er wieder den Schalter. Ansonsten "entnimmt" er den nächsten Kunden der Warteschlange, erfasst dessen Wartezeit und aktiviert sich selbst, sobald der Kunde bedient ist.

Das Hauptprogramm steckt in der `main`-Methode der Klasse `QueueMD1`. Dessen Hauptaufgabe ist, den ersten Kunden zu aktivieren und die Simulation mit der `Simulation.run`-Methode zu starten.

Die Attribute haben die folgenden Bedeutungen:

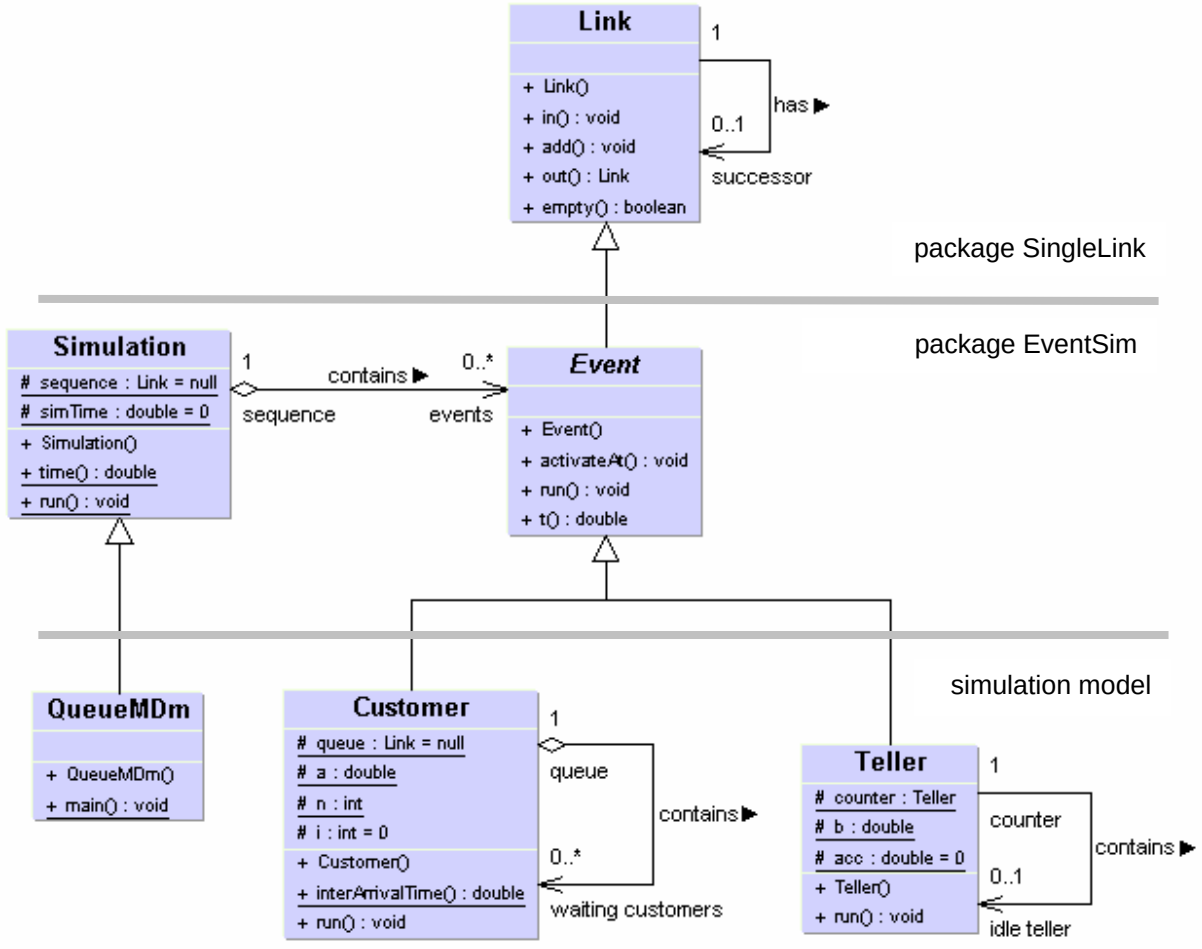
`Customer`

- `queue` ist der Kopf der Kundenwarteschlange (Customer)
- `a` ist die vorgegebene mittlere Zwischenankunftszeit
- `n` ist die Gesamtzahl der Kunden
- `i` ist die Nummer des Kunden (des Customer-Objekts)

`Teller`

- `counter` ist die Referenz auf den Bediener (das einzige Teller-Objekt), solange dieser untätig ist
- `b` ist die vorgegebene konstante Bediendauer
- `acc` ist der Akkumulator für die Wartezeiten der Kunden

Systemarchitektur



Erläuterung: Hier wird die gesamte Simulation auf der denkbar einfachsten Datenstruktur für Kollektionen aufgebaut, nämlich auf der einfach verketteten linearen Liste. Koppelelemente sind die Link-Objekte. Alle Elemente, die Kollektionen bilden müssen (in der System-Sequence, oder in Wartschlangen), sind Realisierungen (Objekte) von Unterklassen der Klasse Link. Die grundsätzliche Verkettbarkeit wird durch eine selbstbezügliche Assoziation angezeigt: Jedes Link-Objekt kann einen Nachfolger (successor) in der linearen Liste haben, der ebenfalls Link-Objekt ist.

Alle Kollektionen werden als lineare Listen aus Link-Elementen aufgebaut. Diese Strukturen werden ebenfalls durch Assoziationen repräsentiert, allerdings auf einer *höheren Abstraktionsstufe:* Die Assoziation verweist auf *alle* Elemente der Liste. Deswegen wird hier für die Kardinalität nicht mehr 0..1 sondern 0..* geschrieben.

Realisierung: Java-Klassen

Bei der folgenden Realisierung wird auf Allgemeinheit kein Wert gelegt. Am besten fasst man das System dieser Lektion als einen Prototyp auf, an dem sich die Grundzüge der ereignisorientierten Simulation studieren lassen..

```
class Customer extends Event {
   static Link queue= new Link();                    //Kundenwarteschlange
   static double a;                          //mittlere Zwischenankunftszeit
   static long n;                                 //Anzahl Kunden insgesamt
   long i;                                                   //Kundennummer
   Customer(long j){i=j;}                                     //Konstruktor
   double InterArrivalTime(){                       //Zufallszahlengenerator
      return -a*Math.log(1-Math.random());
   }
   public void run(){                                                //Aktionen
     if (Teller.counter!=null) {
       Teller.counter.activateAt(-1);
       Teller.counter=null;
     }
     if (i<n) new Customer(i+1).activateAt(
        Simulation.time()+InterArrivalTime()
     );
     queue.add(this);
   }
}

class Teller extends Event {
  static Teller counter=new Teller();                 //Der Bedienschalter
  static double b;                                          //Bediendauer
  static double acc=0;                         //Akkumulator der Wartezeiten
  public void run(){                                                //Aktionen
    if (Customer.queue.empty()) counter=this;
    else {
       Customer c=(Customer)(Customer.queue.out());
       acc+=Simulation.time()-c.t();
       activate_at(Simulation.time()+b);
     }
  }
}

public class QueueMD1 extends Simulation {
   ...
  public static void main (String[] args) {
     //Eingabe: Customer.a, counter.b und Customer.n
     ...
     //Verarbeitung
     new Customer(1).activate_at(0);
     run();

     //Ausgabe
```

```
        ...
        System.out.println(counter.acc/Customer.n);
    }
}
```

Übung

Machen Sie Vorschläge zu den folgenden Punkten.

- Die Unterbringung des Akkumulators in der Teller-Klasse ist fragwürdig. Welche Gesichtspunkte für die Unterbringung der Attribute gibt es. Wo würden Sie den Akkumulator unterbringen?
- Erweitern Sie das Modell so, dass die mittlere Wartezeit samt Vertrauensintervall ausgegeben wird. Verwenden Sie die Klasse `Statistics`.

8 Simulation des $E_r/E_r/m$-Wartesystems - Beispielprojekt

Pflichtenblatt

PROBLEMFORMULIERUNG

Es ist die Aufgabe gestellt, ein allgemeines Simulationsprogramm für G/G/m-Systeme zu erstellen. Um die Sache nicht zu kompliziert zu machen, werden anstelle der generellen Verteilungen Erlang-Verteilungen angenommen. Diese Einschränkung erscheint opportun auch im Hinblick auf die Tatsache, dass sich viele Verteilungen durch Erlang-Verteilungen recht gut approximieren lassen. Grenzfälle der Erlang-Verteilung:

r = 1: Exponentialverteilung

r = ∞: Deterministische Verteilung (Diese Verteilung wird in der `Erlang`-Klasse erzeugt, wenn der Parameter r auf null gesetzt wird.)

Der Simulationsablauf soll interaktiv steuerbar sein derart, dass jeweils nach festen (Simulations-)Zeitintervallen die Simulation für die Befehlseingabe angehalten wird. Dazu ist eine einfache *grafische Bedienoberfläche* (Graphical User Interface, GUI) bereitzustellen für die Parametereingabe, die Ergebnisausgabe und die Steuerbefehle. Auf Anforderung soll ein Histogramm der Wartezeiten ausgegeben werden.

MODELLERSTELLUNG

Modellkonzipierung (Analyse- und Entwurfsphase): Für das Simulationsmodell sind Objekt- und Klassendiagramme zu erstellen. Modellierungssprache ist die Unified Modeling Language (UML).

Allgemein verwendbares Package für die ereignisorientierte Simulation: Für die zeitfolgerichtige Aktivierung von Ereignissen wird ein allgemein verwendbares Package EventSim definiert. Es enthält die Event- und die Simulation-Klasse.

Alle aktiven Objekte des Simulationssystems sind Objekte von Unterklassen der abstrakten Event-Klasse. Diese wiederum ist selbst Unterklasse der Link-Klasse, die eine Verkettung von Ereignis-Objekten zu linearen Listen (Ereignisliste, Warteschlangen) ermöglicht.

Die Simulation-Klasse ist die Basis (Superklasse) des Hauptprogramms der Simulation. Sie enthält die Ereignisliste (sequence). Die Ereignisliste wird von der run-Methode der Simulation-Klasse abgearbeitet. Nacheinander aktiviert sie die Ereignisse der Ereignisliste durch Aufruf von deren run-Methoden. Die Simulationsuhr wird entsprechend mitgeführt.

Das Hauptprogramm ist in der main-Methode der Klasse EEm zu realisieren. Aufgaben des Hauptprogramms sind die Ein- und Ausgabe der Daten, die Aktivierung des ersten Kunden und der Start der Simulation durch Aufruf der Simulation.run-Methode.

Für statistische Auswertungen von Stichproben geschieht mit der allgemein verwendbaren Klasse Statistics. Es ist die Methode der Stapelmittelwerte zu Grunde zu legen.

Bedienung des Programms: Für die interaktive Bedienung des Simulationsprogramms ist eine einfache grafische Oberfläche zu entwickeln. Es enthält Textfenster für die *Eingabe:*

- Mittlere Zwischenankunftszeit
- Erlangparameter für die Zwischenankunftszeit
- Mittlere Bediendauer
- Erlangparameter für die Bediendauer
- Anzahl der Bedieneinheiten

- Stapelgröße für die statistische Erfassung
- Dauer des Simulationsintervalls (Abstand zwischen Haltepunkten)

Die *Ausgabe* des 2-Sigma-Intervalles der mittleren Wartezeite der Kunden geschieht ebenfalls in einem Textfenster. Für die *Ausgabe* der Histogramm-Datei wird ein Textfenster zur Eingabe des Dateinamens vorgesehen und außerdem ein Button für das Anstoßen der Ausgabe der Datei.

Die *Steuerung* der Simulation geschieht mit den Buttons für Start, Weiter und Stopp.

Datenerfassung: entfällt.

Modellformulierung (Spezifikations- und Implementierungsphase): Das System ist als Java-Anwendung zu realisieren. Für die ereignisfähigen Simulationsobjekte ist ein Zustandsdiagramm (Mealy-Automat) zu definieren, das die erlaubten Zustandsübergänge festlegt (Lieferanten-Kunden-Kontrakt). Dabei ist vor allem auf die korrekte Listenverwaltung zu achten: Link-Objekte dürfen nur dann in eine lineare Liste eingeordnet werden, wenn sie nicht gerade zu einer linearen Liste gehören.

Verifikation: Für die run-Methoden sämtlicher ereignisfähigen Simulationsobjekte ist der Nachweis zu erbringen, dass nur die Übergänge gemäß Zustandsdiagramm möglich sind und dass die Prozesse des Systems sich folglich stets in einem Zustand dieses Zustandsdiagramms befinden (Invariante).

Validierung: Als Grenz- und Sonderfall wird das M/D/1-System herangezogen, für das es analytische Resultate gibt.

EXPERIMENTE

Das M/D/1-System ist für die Parameter $a = 5$ und $b = 4$ zu simulieren. Die Ergebnisse dieses Systems sind mit den Ergebnissen des $M/E_5/1$-Systems und weiteren Varianten zu vergleichen.

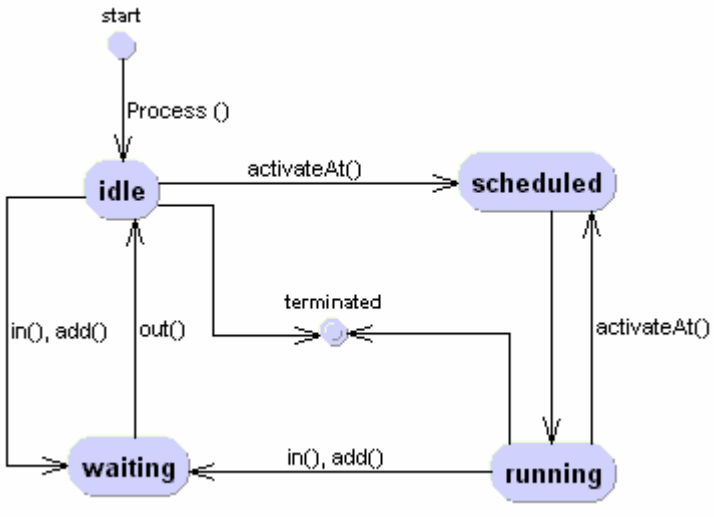
ERGEBNISDARSTELLUNG

Dokumentation: Pflichtenheft, OOA- und OOD-Modelle sowie das Programm werden in einem Bericht auszugsweise zusammengestellt und beschrieben. Dazu kommen exemplarische Ergebnisse.

Entwurf

Spezifikation

Wesentliche Korrektheitsforderungen sind, dass lineare Listen nicht durch unbedachtes Einfügen von Listenelementen zerstört werden, und dass es nicht zu baumelnden Zeigern (dangling references) kommt. Für die Simulationsobjekte werden Zustände definiert derart, dass diese Fehler ausgeschlossen werden können, solange sich jedes Simulationsobjekt in einem der Zustände befindet. Die Zustände sind folgendermaßen definiert:



idle: Prozess läuft nicht und er ist nicht in einer linearen Liste (aber erreichbar)

scheduled: Prozess befindet sich in der Ereignisliste (sequence) und läuft gerade nicht

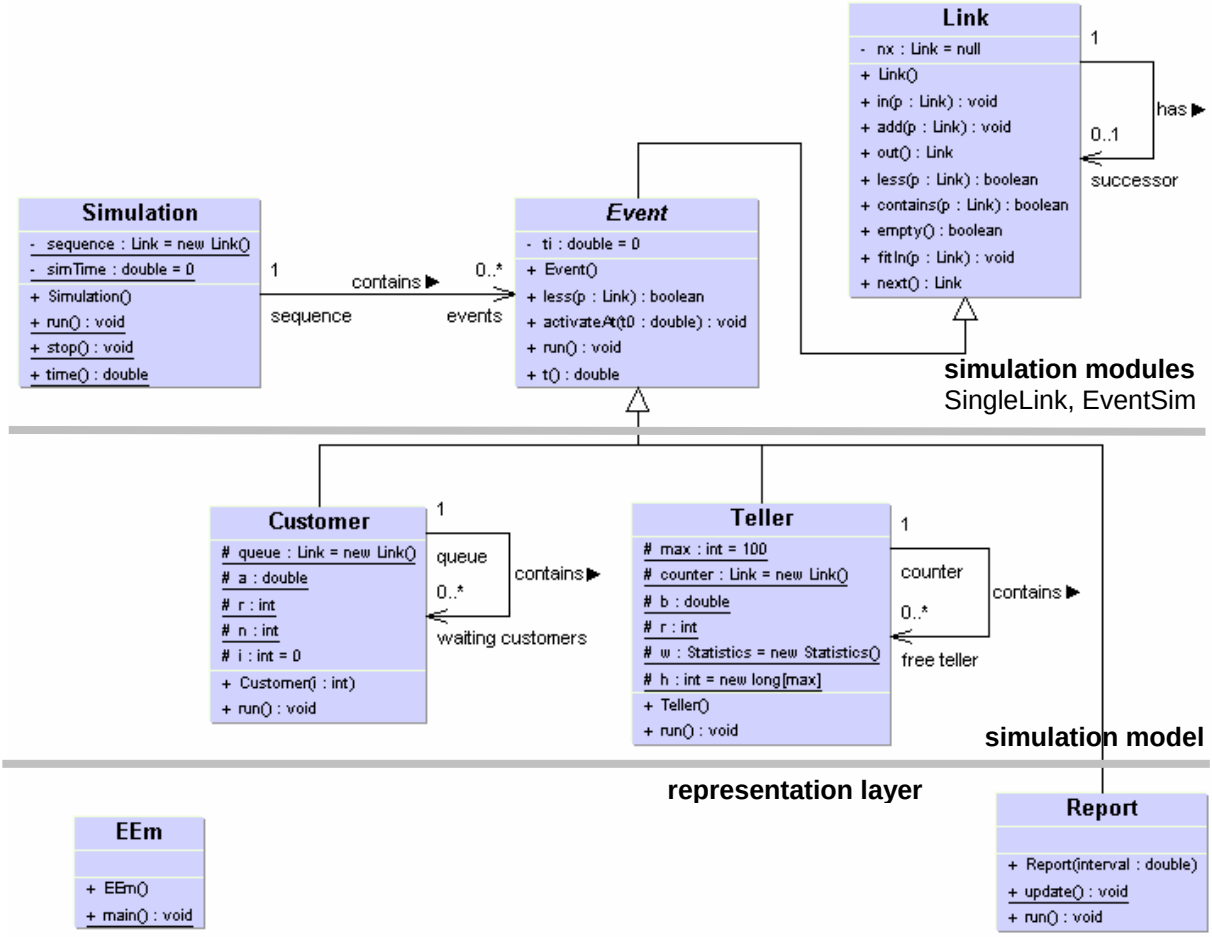
running: Prozess ist der Ereignisliste entnommen worden und läuft (nach Aufruf seiner run-Funktion)

waiting: Prozess wartet in *einer* linearen Liste (nicht Ereignisliste) und läuft gerade nicht

terminated: Prozess ist nicht mehr erreichbar und kann gelöscht werden

Wenn nur die im Zustandsdiagramm zugelassenen Übergänge aufgrund der an den Pfeilen notierten Ereignisse auftreten, wird es nicht zu den eingangs beschriebenen Fehlzuständen kommen.

Klassendiagramm



Mehrebenen-Modell

Der Entwurf folgt einem Mehrebenen-Modell:

1. *Simulationsmodule (simulation modules):* Diese Ebene umfasst alle die Klassen, die für die ereignisorientierte Simulation generell verwendbar sind. Sie sind in den Packages `SingleLink` und `EventSim` zusammengefasst.
2. *Simulationsmodell (simulation model):* Die Klassen des Simulationsmodells enthalten alles, was für das Verhalten des Modells relevant ist. Hier befinden sich insbesondere alle Klassen der Simulationsobjekte, in diesem Fall diejenigen der Kunden (Customer) und Schalterangestellten bzw. Bediener (Teller). Das Simulationsmodell ist frei von allen Darstellungsaspekten.

3. *Darstellungsschicht (representation layer):* Hier sind alle die Klassen zu finden, die für die Kommunikation mit dem Bediener zuständig sind.

Die klare Trennung von Simulationsmodell und Darstellungsschicht erlaubt die Entkopplung der Arbeiten: Zunächst kann man die Ein- und Ausgabe über Konsole vorsehen. Die Entwicklung des Simulationsmodells wird so entlastet von Entwurfsüberlegungen für die grafische Bedienoberfläche (Graphical User Interface, GUI). Und umgekehrt braucht man sich bei der Konstruktion der Bedienoberfläche nicht mehr um das Simulationsmodell zu kümmern.

Das ereignisfähige Report-Objekt

Das Report-Objekt wird als Simulationsobjekt modelliert. Es gehört aber nicht mehr zum Simulationsmodell. Es sorgt für die Unterbrechung der Simulation und es eröffnet die Möglichkeiten zur Ausgabe der Simulationsergebnisse und zur Steuerung des Simulations-Prozesses. Das Report-Objekt plant sich bei der Erzeugung selbst ein, und zwar nach dem vorgesehenen Simulationsintervall.

Wird es aktiv, so hält es den Simulationsprozess an. Wird der Simulationsprozess durch ein externes Ereignis wieder angestoßen, plant sich der Report um das Simulationsintervall später erneut ein. Bei der Konsolenversion realisiert man die Unterbrechung der Simulation einfach durch das Warten auf Eingabe.

Bei der grafischen Bedienoberfläche ist größerer Entwurfsaufwand erforderlich: Es gibt mehr oder weniger gute Lösungen - und eine Menge Fehlermöglichkeiten. Der GUI-Entwurf wird in der folgenden Lektion behandelt.

Implementierung: Java-Quelltexte

Link.java

```
/** Link.java, Timm Grams, Fulda, Februar 2000 (letzte Aenderung: 03.12.03)

Die Klasse Link ist Basisklasse fuer einfach verkettete Datenstrukturen
(lineare und zyklische Listen, Baeume in Parent-Darstellung). Die Methode
less definiert eine abstrakte partielle Ordnung.

Erlaeuterung: Die Methode less und die Klasse Link werden nicht als
abstrakt deklariert. Das ermoeglicht die Nutzung von Objekten der Link-
Klasse als Listenkoepfe. Die Deklaration und Initialisierung einer
linearen Liste mit dem Namen list sieht dann so aus:

  Link list= new Link();

*************************************************************************/

package SingleLink;
public class Link {
 int ref;       //Anzahl der nx-Referenzen auf das Link-Objekt
 Link nx;

 public Link(){ref=0; nx=null;}
 public void in(Link p) {
   if(p.nx==null){p.nx=nx; nx=p; p.ref++;
   }else throw new Error("Link: Link.in(p) failed");
 }
 public void add(Link p) {
   if (p==null); //geaendert: 16.03.00
   else if (nx!=null) nx.add(p); else {nx=p; p.ref++;}
 }
 public Link out() { //geaendert: 16.03.00
   Link p=nx;
   if (p!=null) {nx=p.nx; p.ref--; p.nx=null;}
```

```
  return p;
 }
 public Link bisect(){
   Link p=nx;
   if (p!=null){nx=null; p.ref--;}
   return p;
 }
 public Link next() {return nx;}
 public int nref() {return ref;}

 //Partielle Ordnung
 public boolean less(Link p){return false;};

 /*p wird in die Liste mit dem Kopf this unter Erhaltung der partiellen
 Ordnung eingefuegt. Das Element p wird - falls ein solches existiert - vor
 dem ersten groesseren Element platziert, ansonsten am Ende der Liste.
 ************************************************************************/
 public void fitIn(Link p){
   Link cur=this;
   while (cur.nx!=null) {if (p.less(cur.nx)) break; cur=cur.nx;}
   cur.in(p);
 }

 public boolean contains(Link p) {
   Link cur=this; //Aenderung am 16.03.00
   while (cur!=null&&cur!=p) cur=cur.nx;
   return cur==p;
 }

 public boolean empty() {return nx==null;}

 public void clear() {bisect();}

 public int cardinal() {
   int i=0;
   Link cur=nx;
   while (cur!=null) {cur=cur.nx; i++;}
   return i;
 }
}
```

Event.java

```
/** Event.java, Timm Grams, Fulda, Februar 2000 (20.08.04)

Die Klasse Event bildet die Basis fuer die Modellierung von Ereignissen.
Das Verhalten der Ereignisse wird durch die fuer jeden zu modellierenden
Ereignistyp (Nachkommen von Event) durch Ueberschreiben der run-Methode
festglelegt, die sich weiterer Methoden und Attribute bedienen kann (siehe
auch die Simulation class von Simula).

Die activateAt-Methode sorgt fuer die zeitfolgerichtige Auflistung der
Ereignisse.
************************************************************************
*/
package EventSim;
import java.lang.System;
import SingleLink.*;

abstract public class Event extends Link {
  double ti;
  public Event(){ti=0;}
  public double t(){return ti;}
  public boolean less(Link p) {return ti<((Event)p).ti;}
  public void activateAt(double t0) {
    if (nref()==0) {
      ti=t0;
      Simulation.sequence.fitIn(this);
```

```
      if (ti<Simulation.simTime) ti=Simulation.simTime;
    }else throw new Error("EventSim: Illegal state transition");
  }
  public abstract void run();
}
```

Simulation.java

```
/** Simulation.java, Timm Grams, Fulda
Turbo Pascal-Urversion EventSim: 24.8.90
Java: 01.03.00, 19.12.02, 31.08.04

Die Simulation-Klasse ist Kern der ereignisorientierten Simulation im
Package EventSim. Durch Aufruf der Ablaufmethode run wird die Simulation
gestartet. Die aktuelle Simulationszeit wird von der Methode time gelie-
fert.

Die Ablaufmethode Simulation.run ruft nacheinander die Prozesse auf, und
startet dann deren virtuelle Prozedur run. Simulation.run endet, wenn die
Ereignisfolge (sequence) leer ist. Am Ende der Simulation wird die Simu-
lationszeit zurueckgesetzt um einen problemlosen Neustart zu ermoeglichen.

Die Prozedur stop beendet die Simulation indem sie  die Ereignisfolge
sequence leert.
**************************************************************************/

package EventSim;
import SingleLink.*;

public class Simulation {
  static Link sequence=new Link();
  static double simTime;
  private Simulation() {}
  public static double time(){return simTime;}
  public static void run() {
    while (!sequence.empty()){
      Event p=(Event)sequence.out();
      simTime=p.ti;
      p.run();
    }
    simTime=0;
  }
  public static void stop() {sequence.clear();}
}
```

Customer.java

```
import EventSim.*;
import SingleLink.*;
import Drawings.*;

public class Customer extends Event {
  static Link queue= new Link();  //customer queue
  static double a;                //mean interarrival time
  static int r;                   //Erlang parameter
  static long n;                  //total number of customers
  long i;                         //customer id number

  Customer(long i){n=this.i=i;}
  public void run(){
    if (!Teller.counter.empty()) {
      ((Teller)Teller.counter.out()).activateAt(-1);
    }
    new Customer(i+1).activateAt(Simulation.time()+Erlang.drawing(r, a));
    queue.add(this);
  }
}
```

Teller.java

```
import EventSim.*;
import SingleLink.*;
import Drawings.*;

public class Teller extends Event {
  static final int max=100;
  static Link counter= new Link(); //List of idle tellers
  static double b;                 //mean service time
  static int r;                    //Erlang parameter
  static Statistics  w;            //statistics of waiting periods
  static long[] h= new long[max];
  public void run(){
    double s;  //Sample value
    int k;     //Histogram class
    if (Customer.queue.empty()) counter.in(this); else {
      Customer c=(Customer)(Customer.queue.out());
      w.add(s=Simulation.time()-c.t());
      k=(int)Math.round(s);
      h[k<max?k:max-1]++;
      activateAt(Simulation.time()+Erlang.drawing(r, b));
    }
  }
}
```

ReportC.java (Konsolenversion)

```
import java.io.*;
import EventSim.*;
import Drawings.*;

public class ReportC extends Event {
  static double interval;
  static byte buf[]=new byte[129];

  public ReportC(double interval) {
    this.interval=interval;
    activateAt(interval);
  }

  public static void update() {
    if (0<Customer.n){
      System.out.print("! w = ");
      System.out.print(Teller.w.meanVal());
      System.out.print(" +- ");
      System.out.print(2*Teller.w.stanDev());
      System.out.println(" (2-Sigma-Intervall)");
    }
  }

  public void run() {
    update();
    activateAt(Simulation.time()+interval);

    do {
      try {
        System.out.println("? <c>ontinue, save <h>istogram, <s>top");
        System.in.read(buf);
      }catch(IOException e){
        System.out.println(e.toString());
        e.printStackTrace();
      }
      if(buf[0]=='h') {
        FileOutputStream out;
        DataOutputStream dOut;
        try {
          dOut=new DataOutputStream(
```

```
            out=new FileOutputStream("Histogram.dat"));
          for (int i=0; i<Teller.max;i++)
            dOut.writeBytes(Teller.h[i]+"\n");
          out.close();
        } catch (IOException e) {
          System.out.println("\nERROR: data transfer failure");
        }
      } else if(buf[0]=='s') Simulation.stop();
    } while(buf[0]!='c'&&buf[0]!='s');
  }
}
```

EEmC.java (Hauptprogramm, Konsolenversion)

```
import EventSim.*;
import SingleLink.*;
import Drawings.*;
import java.io.*;

public class EEmC {
  static byte buf[]=new byte[129];
  static double dIn() {
    double x=0;
    try {
      int i=System.in.read(buf);
      return Double.valueOf(new String(buf, 0, i)).doubleValue();
    }catch(IOException e){
      System.out.println(e.toString());
      e.printStackTrace();
    }catch(NumberFormatException e){
      System.out.println(e.toString());
      e.printStackTrace();
    }
    return x;
  }

  public static void main (String[] args) {

    System.out.println("E/E/m QUEUING SYSTEM");

    System.out.print("? Mean interarrival time= ");
    Customer.a=dIn();
    System.out.print("? Erlang paramter= ");
    Customer.r=(int)dIn();
    System.out.print("? Mean service time= ");
    Teller.b=dIn();
    System.out.print("? Erlang parameter= ");
    Teller.r=(int)dIn();

    System.out.print("? Number of service units= ");
    int m=(int)Math.round(dIn());
    while(0<m--) Teller.counter.in(new Teller());

    System.out.print("? Simulation interval= ");
    new ReportC(dIn());

    System.out.print("? Batch size= ");
    Teller.w= new Statistics((int)dIn());

    new Customer(1).activateAt(0);

    Simulation.run();
  }
}
```

Verifikation

Der Nachweis, dass jedes Simulationsobjekt nur die zulässigen Übergänge durchläuft, ist für jedes Event-Objekt (Customer und Counter) erbracht worden (Datei EEmVerifikation.ppt).

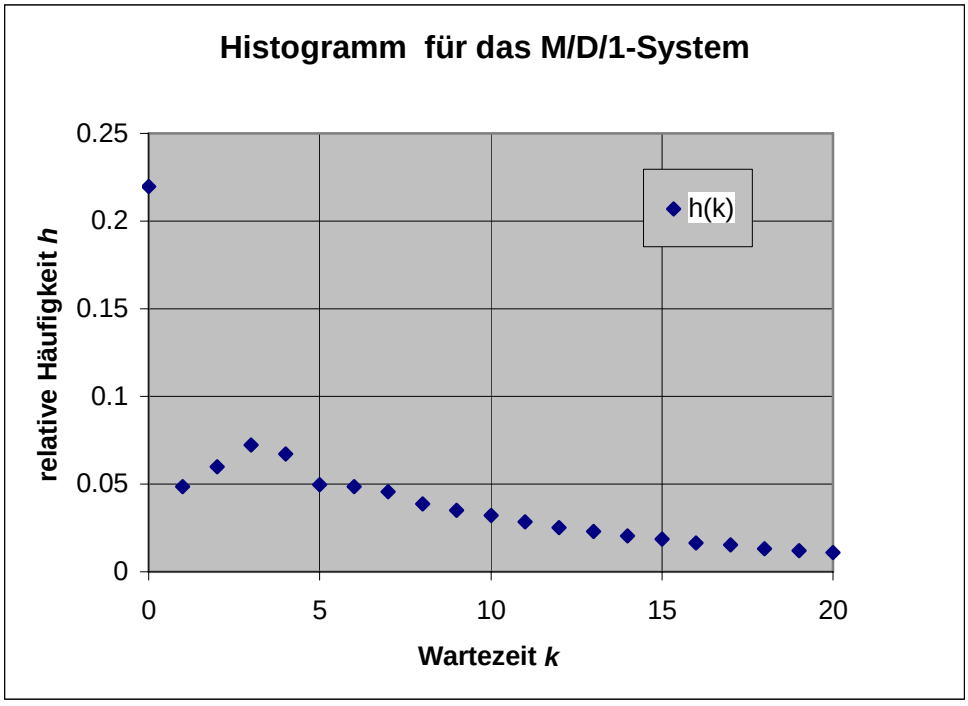
Validierung

Das Modell liefert für das M/D/1-System näherungsweise die vorhergesagten Ergebnisse. Beispielsweise lieferte ein Experiment mit dem Modell für a=1, b=0.8, m=1 und n=10^6 für die mittlere Wartezeit das 2σ-Intervall 1.61±0.02. Das Intervall schließt den analytisch ermittelten Wert 1.6 ein.

Ergebnisse

Für das M/D/1-System mit einer mittleren Zwischenankunftszeit von a = 5 und einer Bedien-dauer von b = 4 ergibt sich eine mittlere Wartezeit von w = 7.96 ± 0.12. Stichprobengröße: 10^6 Kunden. Zum Vergleich:

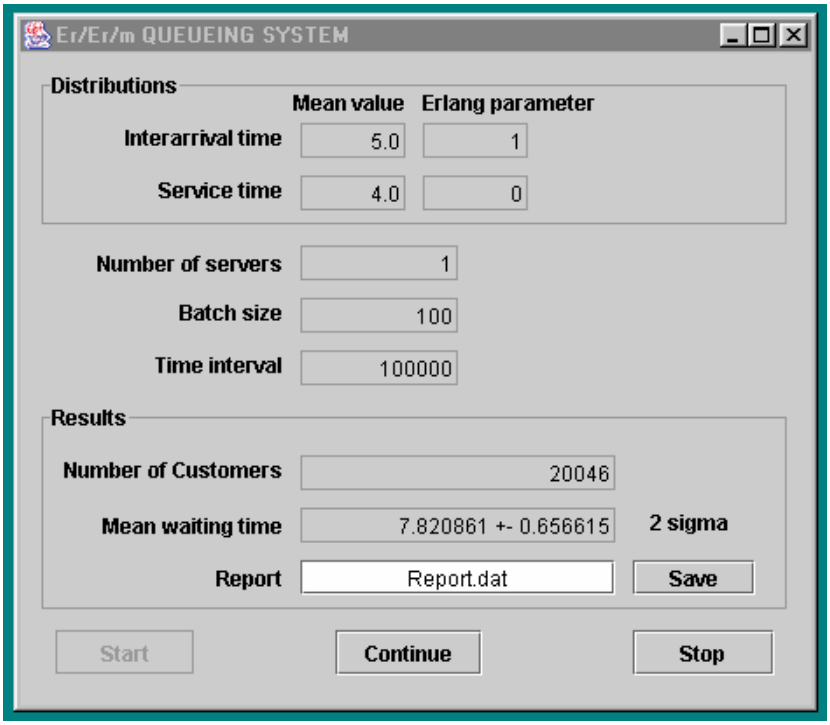
- M/E_5/1-System: w = 9.46 ± 0.15
- M/E_{10}/1-System: w = 8.83 ± 0.13
- M/E_{20}/1-System: w = 8.42 ± 0.13



9 Bedienoberfläche zur interaktiven Steuerung der Simulation

Eine einfache grafische Bedienoberfläche für das E_r/ E_r/m-Wartesystem

Es empfiehlt sich, für den Entwurf einer Bedienoberfläche eine *integrierte Entwicklungsumgebung* (Integrated Development Environment, IDE) zu nutzen, die eine Möglichkeit für das GUI-Editieren enthält. Die kostenlos erhältliche IDE „Sun ONE Studio - Community Edition“ erfüllt diese Anforderungen.



Für das folgende Layout sind nur fünf Java-Klassen des javax.swing-Packages erforderlich: JFrame (die Basis des Layouts) , JLabel, JButton, JTextField (für die Ein-/Ausgabe) und JPanel (zur Gruppierung der Elemente).

Die Klassen der Darstellungsschicht

EEm.java (Bedienoberfläche und Hauptprogramm)

```
import java.io.*;
import EventSim.*;
import Drawings.*;

public class EEm extends javax.swing.JFrame {

    public EEm() {initComponents();}

    private void initComponents() {//GEN-BEGIN:initComponents
        distributionsP = new javax.swing.JPanel();
        aT = new javax.swing.JTextField();
        bT = new javax.swing.JTextField();
        brT = new javax.swing.JTextField();
        arT = new javax.swing.JTextField();
        arL = new javax.swing.JLabel();
        meanL = new javax.swing.JLabel();
        aL = new javax.swing.JLabel();
        bL = new javax.swing.JLabel();
        mL = new javax.swing.JLabel();
        mT = new javax.swing.JTextField();
        tL = new javax.swing.JLabel();
        tT = new javax.swing.JTextField();
        resultsP = new javax.swing.JPanel();
        wL = new javax.swing.JLabel();
        reportT = new javax.swing.JTextField();
        saveB = new javax.swing.JButton();
        wT = new javax.swing.JTextField();
        sigmaL = new javax.swing.JLabel();
```

```
histogramL = new javax.swing.JLabel();
nT = new javax.swing.JTextField();
nL = new javax.swing.JLabel();
startB = new javax.swing.JButton();
contB = new javax.swing.JButton();
stopB = new javax.swing.JButton();
batchSizeL = new javax.swing.JLabel();
batchSizeT = new javax.swing.JTextField();

getContentPane().setLayout(null);

setTitle("Er/Er/m QUEUEING SYSTEM");
setName("Er/Er/m");
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
    }
});

distributionsP.setLayout(null);

distributionsP.setBorder(
    new javax.swing.border.TitledBorder("Distributions"));
aT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
aT.setText("5.0");

distributionsP.add(aT);
aT.setBounds(150, 30, 60, 20);

bT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
bT.setText("4.0");
distributionsP.add(bT);
bT.setBounds(150, 60, 60, 20);

brT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
brT.setText("0");
distributionsP.add(brT);
brT.setBounds(220, 60, 60, 20);

arT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
arT.setText("1");
distributionsP.add(arT);
arT.setBounds(220, 30, 60, 20);

arL.setText("Erlang parameter");
distributionsP.add(arL);
arL.setBounds(220, 10, 110, 16);

meanL.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
meanL.setText("Mean value");
distributionsP.add(meanL);
meanL.setBounds(140, 10, 70, 16);

aL.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
aL.setText("Interarrival time");
distributionsP.add(aL);
aL.setBounds(50, 30, 90, 16);

bL.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
bL.setText("Service time");
distributionsP.add(bL);
bL.setBounds(50, 60, 90, 16);

getContentPane().add(distributionsP);
distributionsP.setBounds(10, 10, 430, 90);

mL.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
```

```
mL.setText("Number of servers");
getContentPane().add(mL);
mL.setBounds(40, 110, 110, 20);

mT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
mT.setText("1");
getContentPane().add(mT);
mT.setBounds(160, 110, 90, 20);

tL.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
tL.setText("Time interval");
getContentPane().add(tL);
tL.setBounds(70, 170, 80, 16);

tT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
tT.setText("100000");

getContentPane().add(tT);
tT.setBounds(160, 170, 90, 20);

resultsP.setLayout(null);

resultsP.setBorder(new javax.swing.border.TitledBorder("Results"));
wL.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
wL.setText("Mean waiting time");
resultsP.add(wL);
wL.setBounds(30, 60, 110, 20);

reportT.setHorizontalAlignment(javax.swing.JTextField.CENTER);
reportT.setText("Report.dat");
resultsP.add(reportT);
reportT.setBounds(150, 90, 180, 20);

saveB.setText("Save");

saveB.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        saveBActionPerformed(evt);
    }
});

resultsP.add(saveB);
saveB.setBounds(340, 90, 70, 20);

wT.setEditable(false);
wT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
wT.setText("???");
resultsP.add(wT);
wT.setBounds(150, 60, 180, 20);

sigmaL.setText("2 sigma");
resultsP.add(sigmaL);
sigmaL.setBounds(350, 60, 50, 16);

histogramL.setHorizontalAlignment(
    javax.swing.SwingConstants.RIGHT);
histogramL.setText("Report");
resultsP.add(histogramL);
histogramL.setBounds(50, 90, 90, 20);

nT.setEditable(false);
nT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
nT.setText("???");
resultsP.add(nT);
nT.setBounds(150, 30, 180, 20);

nL.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
```

```
    nL.setText("Number of Customers");
    resultsP.add(nL);
    nL.setBounds(10, 30, 130, 16);

    getContentPane().add(resultsP);
    resultsP.setBounds(10, 200, 430, 120);

    startB.setText("Start");
    startB.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            startBActionPerformed(evt);
        }
    });

    getContentPane().add(startB);
    startB.setBounds(20, 330, 80, 26);

    contB.setText("Continue");
    contB.setEnabled(false);
    contB.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            contBActionPerformed(evt);
        }
    });

    getContentPane().add(contB);
    contB.setBounds(180, 330, 84, 26);

    stopB.setText("Stop");
    stopB.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            stopBActionPerformed(evt);
        }
    });

    getContentPane().add(stopB);
    stopB.setBounds(350, 330, 81, 26);

    batchSizeL.setHorizontalAlignment(
        javax.swing.SwingConstants.RIGHT);
    batchSizeL.setText("Batch size");
    getContentPane().add(batchSizeL);
    batchSizeL.setBounds(40, 140, 110, 16);

    batchSizeT.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
    batchSizeT.setText("100");
    getContentPane().add(batchSizeT);
    batchSizeT.setBounds(160, 140, 90, 20);

    pack();
    java.awt.Dimension screenSize =
        java.awt.Toolkit.getDefaultToolkit().getScreenSize();
    setSize(new java.awt.Dimension(460, 400));
    setLocation((screenSize.width-460)/2,(screenSize.height-400)/2);
}

private void stopBActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}

private void contBActionPerformed(java.awt.event.ActionEvent evt) {
     wT.setText("???");
     nT.setText("???");
     freeze=false;
}
private void saveBActionPerformed(java.awt.event.ActionEvent evt) {
    FileOutputStream out;
```

```
    DataOutputStream dOut;
    try {
      dOut=new DataOutputStream(out=new FileOutputStream(
        reportT.getText())
      );
      dOut.writeBytes("Er/Er/m QUEUING SYSTEM\na= " + aT.getText()
        +" (r= "+arT.getText()+"), b= "
        +bT.getText() +" (r= "+brT.getText()
        +")\nm= "+mT.getText()+", n= "+nT.getText()
        +"\n\nHistogram\n"
      );
      for (int i=0; i<Teller.max;i++)
          dOut.writeBytes(Teller.h[i]+"\n");
      out.close();
    } catch (IOException e) {
      System.out.println("\nERROR: Data transfer failure");
    }
}

private void startBActionPerformed(java.awt.event.ActionEvent evt) {
    Customer.queue.clear();
    Customer.a=new Double(aT.getText()).doubleValue();
    aT.setEditable(false);
    Customer.r=new Integer(arT.getText()).intValue();
    arT.setEditable(false);
    Teller.b=new Double(bT.getText()).doubleValue();
    bT.setEditable(false);
    Teller.r=new Integer(brT.getText()).intValue();
    brT.setEditable(false);
    int m= new Integer(mT.getText()).intValue();
    mT.setEditable(false);
    Teller.w= new Statistics(
        new Integer(batchSizeT.getText()).intValue());
    batchSizeT.setEditable(false);
    Teller.counter.clear();
    while (0<m--) Teller.counter.in(new Teller());
    new Customer(1).activateAt(0);
    new Report(new Double(tT.getText()).doubleValue());
    tT.setEditable(false);
    freeze=false;
    startB.setEnabled(false);
}

private void exitForm(java.awt.event.WindowEvent evt) {
    System.exit(0);
}

public static void main(String args[]) {

    (instance = new EEm()).show();

    while (freeze); freeze=true; //blocking (active wait)

    Simulation.run();
}


private javax.swing.JTextField batchSizeT;
public javax.swing.JButton contB;
private javax.swing.JLabel mL;
private javax.swing.JLabel histogramL;
public javax.swing.JTextField reportT;
private javax.swing.JTextField aT;
private javax.swing.JButton stopB;
private javax.swing.JLabel nL;
private javax.swing.JTextField bT;
private javax.swing.JLabel batchSizeL;
```

```
    private javax.swing.JPanel resultsP;
    private javax.swing.JLabel sigmaL;
    private javax.swing.JButton saveB;
    private javax.swing.JTextField mT;
    private javax.swing.JLabel meanL;
    public javax.swing.JTextField nT;
    private javax.swing.JTextField brT;
    private javax.swing.JPanel distributionsP;
    private javax.swing.JLabel wL;
    private javax.swing.JButton startB;
    private javax.swing.JLabel bL;
    private javax.swing.JLabel tL;
    private javax.swing.JLabel arL;
    public javax.swing.JTextField wT;
    private javax.swing.JLabel aL;
    private javax.swing.JTextField tT;
    private javax.swing.JTextField arT;

    public static boolean freeze = true;
    public static EEm instance;
}
```

Report.java

```
import java.io.*;
import java.util.*;
import java.text.*;
import EventSim.*;
import Drawings.*;

public class Report extends Event {
    static double interval;
    static NumberFormat nf= NumberFormat.getInstance(Locale.US);

    public Report(double interval) {
        this.interval=interval;
        nf.setMaximumFractionDigits(6);
        nf.setGroupingUsed(false);
        activateAt(interval);
    }

    public static void update() {
        EEm.instance.nT.setText(nf.format(Customer.n).toString());
        EEm.instance.wT.setText(nf.format(Teller.w.meanVal())+
          " +- " + nf.format(2*Teller.w.stanDev())
        );
    }

    public void run() {
        update();
        EEm.instance.contB.setEnabled(true);
        while (EEm.freeze); //blocking (active wait)
        EEm.freeze=true;
        EEm.instance.nT.setText("???");
        EEm.instance.wT.setText("???");
        EEm.instance.contB.setEnabled(false);
        activateAt(Simulation.time()+interval);
    }

}
```

10 Threads und Synchronisation

Nebenläufigkeit

Ein Thread kommt selten allein

Das Hauptprogramm[7] EEm.java beginnt mit „angezogener Handbremse“:

```
while (freeze); freeze=true; //blocking (active wait)
Simulation.run();
```

Das Programm geht also erst einmal in eine Warteschleife, aus der es durch Betätigung des Start-Buttons befreit wird: Durch die Methode `startBActionPerformed(...)` wird die Variable freeze gleich false gesetzt, die Schleife verlassen und die Simulation gestartet. Das sieht recht umständlich aus. Man könnte ja die Simulation gleich mit dem Start-Button aktivieren, also innerhalb der Methode `startBActionPerformed(...)` der `Action-Listener`-Klasse aufrufen. Solche Methoden werden im Folgenden auch Event-Handler genannt.

Gesagt, getan. Nach der Änderung lässt sich das Programm zwar starten. Aber danach macht die Bedienoberfläche keinen Mucks mehr. Bei näherem Hinsehen hat das Programm bis zur Aktivierung des Report-Ereignisses alles gemacht und auch die update-Methode abgearbeitet. Aber auf der Bedienoberfläche lässt sich das nicht mehr erkennen. Die gesamte grafische Repräsentation scheint abgeschaltet zu sein. Und genau so ist es.

Was ist passiert? Wir gehen normalerweise davon aus, dass der Programmlauf aus einer einzigen Folge von Anweisungen besteht, so wie wir dies im Programm aufgeschrieben haben. Aber genau unter dieser Annahme hätte unser ursprüngliches Programm gar nicht laufen können! Denn wie hätte es nach dem Start der Simulation überhaupt noch auf Tastatur- und Mausereignisse reagieren können? Die Ereignisse kommen im Simulationsprogramm nicht vor; sie treffen von außen ein.

Die Windows-Ereignisse werden in einer eigenen Programm-Ablaufsequenz abgearbeitet. Es gibt also mehrere solcher Ablaufsequenzen. Diese werden Threads genannt. Nur ein Thread kann den Prozessor gerade „haben“ und folglich laufen.

Aber der laufende Thread kann von einem Thread mit höherer Piorität verdrängt werden. Und ein Thread höherer Priorität ist es, der die Windows-Ereignisse des AWT (Abstract Window Toolkit) bearbeitet.

Wenn wir uns die Prioritäten der verschiedenen Threads einmal anschauen, mit denen wir es zu tun haben, wird verständlich, warum unser Programm in der ursprünglichen Version einwandfrei läuft. Zumindest diese Threads sind in einer GUI-Anwendung vorhanden[8]:

System thread group

- Clock handler
- Idle thread (Priorität 0)

---

[7] Im Programmtext sind die hier zu besprechenden Elemente durch fette Schrift hervorgehoben worden.

[8] Wenigstens die zwei folgenden Thread Gruppen erzeugt die JVM. Sie enthalten - außer den aufgeführten Threads - weitere Deamon-Threads (Oaks, Wong, 1997, p. 225). Deamon-Threads sind nur dazu da, Anwender-Threads zu unterstützen. Wenn es keine Anwender-Threads mehr gibt, haben diese Deamon-Threads keine Aufgabe mehr. Wenn nur noch Deamon-Thread vorhanden sind, beendet die JVM ihre Arbeit. (Oaks, Wong, 1997, p. 137).

- Garbage collector (Priorität 1)
- Finalizer thread (Priorität 1)

Main thread group

- Main thread (normale Priorität: 5): Das ist der Thread der mit der main-Methode gestartet wird.
- AWT-Input (höhere Priorität als normal[9]: 6): Hier werden die Ereignisse des zugrunde liegenden Window Systems behandelt und die notwendigen Aufrufe der Event-handling-Methoden des AWT (Abstract Window Toolkit) gemacht.
- AWT-Toolkit (plattformspezifisch)
- ScreenUpdater: Hier erfolgen die Aufrufe der `repaint`-Methode. Wenn ein Programm die `repaint`-Methode aufruft, wird eine `notify`-Anweisung an diesen Thread gesandt. Er kann dann die `update`-Methode aller betroffenen Komponenten aufrufen.

Multitasking und Multithreading

Mit dem so genannten *preemptive time-slicing multitasking* schafft es Windows, dem Anwender die Möglichkeit zu eröffnen, mit mehreren Programmen (Anwendungen) quasi gleichzeitig zu arbeiten. Multitasking ist die Fähigkeit des Betriebssystems (Operating System, OS), mehrere Programme nebenläufig auszuführen. Das Betriebssystem teilt jedem Prozess Zeitscheiben zu (time-slicing). Die Prozesse werden reihum zur Ausführung berechtigt (Staffelholzverfahren, round robin). Ist Zeit für die Weitergabe des Staffelholzes, wird nicht abgewartet, bis der laufende Prozess fertig ist. Er wird unterbrochen (preemptive multitasking).

„*Multithreading* ist Multitasking innerhalb eines Programms“ (Petzold, 1999, S. 1197). Ein *Thread (of Control)* ist definiert als eine Befehlsfolge, die unabhängig von anderen Threads of Control innerhalb eines einzigen Programms ausgeführt wird (Oaks, Wong, 1997, S. 3).

Das Standardverfahren der JVM für das Scheduling von Threads ist das *preemptive priority-based multithreading* (Oaks, Wong, 1997, S. 117). Es gibt also kein Zeitscheibenverfahren. Jeder aktuell laufende Thread kann nur durch einen Thread höherer Priorität unterbrochen werden.

Warten und dabei nicht blockieren

Synchronisation

Wir kehren zur ursprünglichen Lösung zurück.

Jetzt, wo wir gesehen haben, dass mehrere Threads auf dem Rechner laufen, stellt sich die Frage, ob diese Lösung auch angemessen ist. Die Sache funktioniert zwar, aber ziemlich schlecht: Solange das Programm auf Eingabe wartet, wird der Rechner blockiert. In dieser Zeit läuft keiner der anderen Java-Threads.

Es geht nun darum, dieses blockierende (aktive) Warten zu vermeiden. Dazu nutzen wir die Möglichkeit, die Ausführung eines Threads aus sich heraus zu unterbrechen. Das gelingt mit den `Object`-Methoden `wait()` und `notify()`, die in `synchronized`-Statements eingebettet sind.

Die Anweisungen für das nichtblockierende Warten `(wait)` und das Aufwecken `(notify)` sind prinzipiell unterbrechbar. Das heißt: wenn zwei Threads unterschiedlicher Priorität

---

[9] Ermittelt mit dem Aufruf `Thread.currentThread().getPriority()`

mit diesen Befehlen miteinander kommunizieren, könnte es grundsätzlich zur Unterbrechung inmitten eines solchen Befehls und damit zu unverhofftem Verhalten der Threads kommen. Deshalb ist dafür zu sorgen, dass diese Befehle in Programmabschnitten untergebracht sind, die sich wechselseitig ausschließen (mutual exclusion). Diesen wechselseitigen Ausschluss erzielt Java mit den `synchronized`-Statements und den `synchronized`-Methoden. Nur erstere wollen wir uns hier näher ansehen.

Jedes Objekt besitzt einen *Lock* (Verschluss). Mit dem `synchronized`-Statement erwirbt der Thread für die Dauer der Ausführung des Blockes den Lock für ein Objekt. Solange ein Thread den Lock hält, kann kein anderer Thread diesen Lock erwerben. Das heißt, dass sich auf diese Weise der wechselseitige Ausschluss von Anweisungsfolgen realisieren lässt.

Das `synchronized`-Statement hat folgende Struktur:

```
synchronized (Expression) { ... }
```

Der Ausdruck *Expression* muss vom Referenztyp sein, das heißt, er verweist auf ein Objekt. Mit dem `synchronized`-Statement erwirbt der ausführende Thread den Lock dieses Objekts sobald er frei ist. Erst wenn der Lock erworben worden ist, wird der Block ausgeführt. Nach Beendigung des Blockes wird der Lock zurückgegeben.

Die `Thread`-Methoden `wait` und `notify`

Die `wait`-Methode darf nur innerhalb eines synchronisierten Programmabschnitts aufgerufen werden. Bei Aufruf der `wait`-Methode hält der ausführende Thread also den Lock auf ein Objekt. Die `wait`-Methode bewirkt, dass sich der aktuelle Thread in die Wartschlange des Objekts einreiht und den Lock des Objekts zurückgibt. Der Thread schläft nun, bis er durch eine `notify`-Methode für dieses Objekt wieder aufgeweckt wird.

Der Aufruf der `notify`-Methode eines Objekts weckt einen wartenden Thread auf, falls überhaupt ein Thread wartet. Warten mehrere, wird einer ausgewählt. Der aufgeweckte Thread bewirbt sich wieder um den Lock des Objekts. Erst wenn er den Lock wieder hat, kann er sein `synchronized`-Statement fortsetzen und ordentlich - also mit Rückgabe des Locks - beenden.

Lösung mit nicht blockierendem Warten

Die `while`-Schleifen mit der `freeze`-Variablen ersetzen wir nun durch die passenden `synchronized`-Statements mit der Aufforderung zum Warten. Als Objekt für die Synchronisation wählen wir die `instance`-Variable der EEm-Klasse. Damit erhalten wir die folgenden Formulierungen für die `main`-Methode:

```
public static void main(String args[]) {

    (instance = new EEm()).show();
    synchronized (instance) {
        try{instance.wait();}
        catch(Exception e) {
            System.out.println(e.toString());
            e.printStackTrace();
        }
    }
    Simulation.run();
 }
```

Die `run`-Methode des Report-Ereignisses nimmt damit folgende Gestalt an:

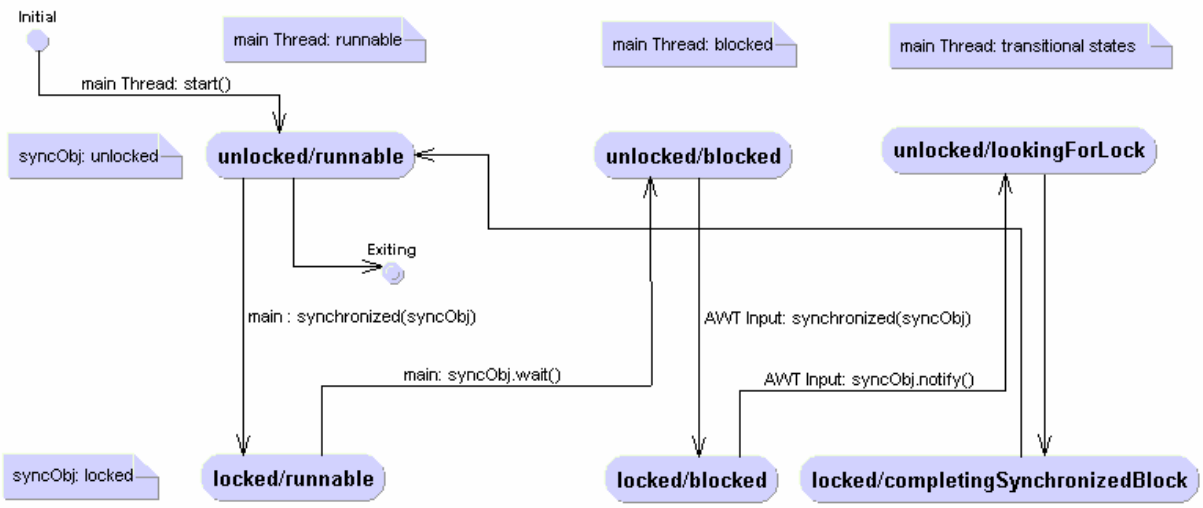
```
public void run() {
    update();
```

```
        EEm.instance.contB.setEnabled(true);
        synchronized (EEm.instance) {
           try{EEm.instance.wait();}
           catch(Exception e) {
              System.out.println(e.toString());
              e.printStackTrace();
           }
        }
        EEm.instance.nT.setText("???");
        EEm.instance.wT.setText("???");
        EEm.instance.contB.setEnabled(false);
        activateAt(Simulation.time()+interval);
    }
```

In den Event-Handlern für den Start-Button und den Continue-Button muss jetzt der main-Thread wieder „wachgeküsst“ werden. Die entsprechenden Synchronized-Statements sehen so aus:

```
synchronized(EEm.instance) {
    EEm.instance.notify();
}
```

In dem folgenden Zustandsübergangsdiagramm wird das Synchronisationsobjekt (im Beispiel EEm.instance) mit syncObj bezeichnet.



Ein eigener Simulations-Thread

Die main-Methode ist nach wie vor nicht sehr schön: Start mit anschließender Vollbremsung. Wir verfolgen die Idee weiter, die Simulation erst im Event-Handler des Start-Buttons aufzurufen. Wir schaffen dafür einen eigenen Thread für die Simulation. Den kann man dann mit einer passenden Priorität versehen. Aber es schadet auch nichts, wenn der Thread die Default-Priorität behält. Wenn dieser Thread im Event-Handler erzeugt wird, hat er von vornherein - also per Default (Unterlassung) - dieselbe Priorität wie der AWT-Input-Thread. Mit der eben beschriebenen Methode kann man diesen neu geschaffenen Thread in den Wartezustand schicken und so den Weg frei machen für die AWT-Events.

Es gibt mehrere Möglichkeiten, einen eigenen Thread für die Simulation zu generieren. Hier wird die Simulation-Klasse mit etwas mehr Fähigkeiten ausgestattet und zu einer Thread-Klasse gemacht, und zwar in Form einer Hüllklasse (Wrapper) für die Simulation-Klasse:

```
/** SimulationThread.java, Timm Grams, Fulda, 25.08.04
```

```
Die SimulationThread-Klasse ist eine Wrapper-Klasse. Sie macht aus der
Simulation-Klasse eine Thread-Klasse. Es gibt nur eine Realisierung, die
auf der instance-Variablen mitgeliefert wird (Singleton Entwurfsmuster).
**************************************************************************/

package EventSim;
import SingleLink.*;

public class SimulationThread extends Thread {
  public static SimulationThread instance= new SimulationThread();
  private SimulationThread() {}
  public void run() {Simulation.run();}
}
```

In der `startBActionPerformed`-Methode erfolgt der Start des `Simulation`-Threads mit der Anweisung „`SimulationThread.instance.start();`". Die Start-Methode des Threads ruft dann die `run`-Methode des `SimulationThread`-Objekts auf.

Die `main`-Methode von EEm kümmert sich jetzt nur noch um die Bereitstellung des GUI:

```
public static void main(String args[]) {

   (instance = new EEm()).show();
}
```

Unterbrechung und Fortsetzung der Simulation werden wie bisher durch die `run`-Methode des `Report`-Events und den Event-Handler `contActionPerformed` erledigt.

11 Diskrete Approximation des G/G/1-Waresystems

Einleitung

Gesucht ist eine weitere Möglichkeit zur Validierung und Überprüfung von Programmen der ereignisorientierten Simulation komplexer Wartesysteme.

Für einige einfache Wartesysteme kennt man Formeln für die mittlere Wartezeiten (Formeln von Pollaczek-Khinchin). Viele grobe Modellierungsfehler lassen sich bereits damit aufdecken.

Zweck der vorliegenden Studie ist, für einfache Wartesysteme (G/G/1-Wartesysteme) die Verteilungsfunktion der Wartezeit eines jeden Kunden direkt mittels Wahrscheinlichkeitsrechnung zu bestimmen.

Dazu werden die frei vorgebbaren Verteilungen der Zwischenankunftszeit und der Bediendauer diskretisiert. Über die Grundrelation für die Wartezeiten erhält man dann Formeln für die diskreten Verteilungen der Wartezeiten.

Verteilung der Wartezeiten

Die Verteilungen von Zwischenankunftszeit und Bediendauer werden als diskret vorausgesetzt. Die zufälligen Zeiten T und X nehmen o.B.d.A. nur die ganzzahligen Werte 0, 1, 2, 3, ... an. Das entspricht einer Diskretisierung mit der Schrittweite $\Delta t = 1$.

Die diskreten Verteilungen der Zwischenankunftszeiten und der Bediendauern sind gegeben durch die Folgen $a = (a_0, a_1, a_2, ...)$ und $b = (b_0, b_1, b_2, ...)$. Deren Element sind $a_i = P(T = i)$ und $b_i = P(X = i)$. Die diskrete Verteilung der Bilanzvariablen U ist gegeben durch $c_i = P(U = i) = P(X - T = i)$. Diese Folge erstreckt sich auch über negative Indizes i. Die Umkehrung einer Folge markieren wir mit einem Strich. Beispielsweise ist $c'_i = c_{-i} = P(U = -i) = P(-U = i) = P(T-X = i)$.

Die Verteilung der Wartezeit des n-ten Kunden oder Auftrags wird durch die Folge $p_n = (p_{n,0}, p_{n,1}, p_{n,2}, ...)$ dargestellt: $p_{n,i} = P(W_n = i)$. Aus der Formel für die Bilanzvariable und aus der *Rekursionsbeziehung für Wartezeiten*

$$w_{n+1} = \max(0, w_n + x_n - t_{n+1}) = \max(0, w_n + u_n)$$

erhält man die folgenden Zusammenhänge zwischen den Wahrscheinlichkeiten:

$$c_k = \sum_{i,j|j-i=k} a_i b_j = \sum_i a_i b_{k+i} = \sum_i a_{-i} b_{k-i} = \sum_i a'_i\, b_{k-i} \tag{0}$$

$$c'_k = c_{-k} = \sum_i a_i b_{i-k} = \sum_i a_{i+k} b_i \tag{0'}$$

$$p_{n+1,k} = \sum_{i,j|i+j=k} p_{n,i} c_j = \sum_i p_{n,i} c_{k-i} = \sum_i p_{n,i+k} c_{-i} = \sum_i p_{n,i+k} c'_i \quad \text{für } 0<k \tag{1}$$

$$p_{n+1,0} = \sum_{i,j|i+j\le 0} p_{n,i} c_j \tag{2}$$

$$p_{n+1,k} = 0 \quad \text{für } k<0 \tag{3}$$

Da die Wartezeit des ersten Kunden gleich null ist, gilt $p_{0,0} = 1$ und $p_{0,k} = 0$ für alle k ungleich 0. Aus den Gleichungen (1) bis (3) lassen sich die Verteilungen der Wartezeiten der weiteren Kunden rekursiv ermitteln.

Die Gleichungen (0) und (1) beinhalten im Wesentlichen Faltungsprodukte von Folgen. Diese lassen sich effizient mit der schnellen Fourier-Transformation berechnen.

Die Gleichung (2) ist entbehrlich. Den Wert $p_{n+1,\,0}$ kann man sich auch über die Bedingung verschaffen, dass die Summe der Wahrscheinlichkeiten der jeweiligen Verteilung eins ergeben muss.

Beispiel: M/D/1-Wartesystem

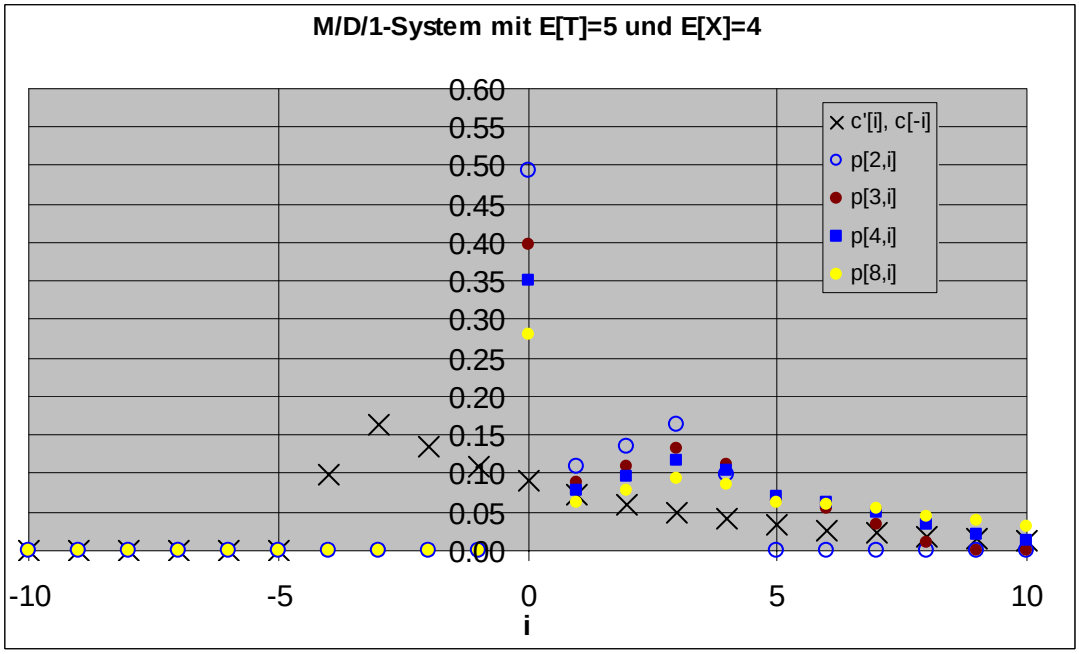
Für das M/D/1-Wartesystem mit der mittleren Zwischenankunftszeit 5 und der Bediendauer 4 werden die Wahrscheinlichkeitsverteilungen der ersten acht Kunden bestimmt. Die bei der Berechnung sich ergebenden Folgen sind (auszugsweise) in der folgenden Tabelle und im Diagramm dargestellt.

Die diskrete Verteilung der Zwischenankunftszeit ergibt sich so: Man stelle sich die kontinuierliche Zeitachse als eine Folge von Intervallen der Länge 1 vor. Jeder ganzzahlige Zeitwert i möge in der Mitte eines solchen Zeitintervalls liegen. Die Verteilungsdichte der Zwischenankunftszeit ist gleich 0 für $t<0$ und gleich $\frac{1}{5}e^{-t/5}$ für $0\le t$. Der Wert a_i wird gleich dem Wert der Verteilungsdichte zur Zeit $t = i$ gesetzt: $a_i = \frac{1}{5}e^{-i/5}$. Diese Formel ist für $0<i$ eine gute Wahl. In der Mitte des Intervalls [-0.5, 0.5] springt die Veteilungsdichte von 0 auf 0.2. Bei der Diskretisierung der Veteilungsdichte ist der halbe Funktionswert einzusetzen, also $a_0 = 0.1$. Für negative i sind die a_i gleich null.

Die anderen Folgen werden über die Formeln (1) bis (3) ermittelt (Excel Arbeitsblatt gg1.xls).

Tabelle: M/D/1-Wartesystem mit a=5 und b=4

i	a_i	b_i	c_{-i}	$p_{1,i}$	$p_{2,i}$	$p_{3,i}$	$p_{4,i}$	$p_{8,i}$
-4	0	0	0.10	0	0	0	0	0
-3	0	0	0.16	0	0	0	0	0
-2	0	0	0.13	0	0	0	0	0
-1	0	0	0.11	0	0	0	0	0
0	0.10	0	0.09	1	0.49	0.40	0.35	0.28
1	0.16	0	0.07	0	0.11	0.09	0.08	0.06
2	0.13	0	0.06	0	0.13	0.11	0.10	0.08
3	0.11	0	0.05	0	0.16	0.13	0.12	0.09
4	0.09	1	0.04	0	0.10	0.11	0.10	0.09
5	0.07	0	0.03	0	0.00	0.07	0.07	0.06
6	0.06	0	0.03	0	0.00	0.05	0.06	0.06
7	0.05	0	0.02	0	0.00	0.03	0.05	0.05
8	0.04	0	0.02	0	0.00	0.01	0.03	0.05
9	0.03	0	0.01	0	0.00	0.00	0.02	0.04
10	0.03	0	0.01	0	0.00	0.00	0.01	0.03



Beispiel einer analytischen Lösung

<<<DER ABSCHNITT MUSS NOCH ÜBERARBEITET WERDEN>>>

Verteilung der Zwischenankunftszeit: $a(k) = 1$ für $k=2$ und $a(k) = 0$ sonst

Verteilung der Bediendauer: $b(k) = 1/2$ für $k=0$ und $k=3$ und $b(k) = 0$ sonst

Daraus folgt für die diskrete Verteilung der Bilanzvariablen: $c(k) = 1/2$ für $k=-2$ und $k=1$ und $c(k) = 0$ sonst. Zu lösen ist die folgende Differenzengleichung:

$$p(k) = (p(k+2) + p(k-1))/2 \text{ für } k>0 \qquad (1')$$

Neben den Bedingungen (2) und (3) ist noch die Bedingung

$$p(0) = (p(0) + p(1) + p(2))/2 \qquad (2')$$

zu erfüllen. Der Lösungsansatz $p(k) = Az^k$ für $k>0$ führt auf die charakteristische Gleichung

$$z^3 - 2z + 1 = 0$$

Da nur solche Lösungen dieser Gleichung relevant sind, für die $|z|<1$ ist, bleibt nur die Lösung $z=(\sqrt{5} - 1)/2$ übrig. (Die ausgeschiedenen Lösungen sind $z=1$ und $z=-(\sqrt{5}+1)/2$.) Die Bedingung (4) wird zu

$$1 = p(0) + Az/(1-z) \qquad (4')$$

und (2') geht über in

$$p(0) = p(1) + p(2) = A(z + z^2) = A \qquad (2'')$$

(2'') und (4') liefern $p(0) = A = 1-z = (3- \sqrt{5})/2$. Damit ergibt sich für die mittlere Wartezeit das Ergebnis

$$w = \Sigma_{k>0}\, kp(k) = A\, \Sigma_{k>0}\, kz^k = Az/(1-z)^2 = 1{,}618$$

Zum Vergleich ein Simulationsergebnis mit dem Programm Queue für die Werte $k = 1000$ und $n = 1000$: $w = 1{,}61 \pm 0{,}02$ (2σ-Bereich).

12 Erzeugung nicht-homogener Poisson-Prozesse

Stationäre Verteilungen von Alter (Alterspyramide) und Restlebensdauer

Wir betrachten ein Objekt, das nach einer durch Zufall bestimmten Zeit verschwindet. Unmittelbar nach seinem Ende wird dieses Objekt durch ein gleichartiges neues ersetzt. Die Ereignisse der Erneuerung werden auf der Zeitachse markiert. Die Lebensdauern der Objekte - also die Intervalle zwischen den Ereignissen - mögen Realisierungen einer Zufallsvariablen X sein. Wir nehmen der Einfachheit halber an, dass die Zufallsvariable X diskret ist.

Bezeichnungen:

X Zufallsvariable der Lebensdauer eines Objekts. Zeit zwischen Ereignissen. Intervalllänge.

p_i Wahrscheinlichkeit, dass ein Objekt die Lebensdauer t_i hat:

$$p(X = t_i) = p_i$$

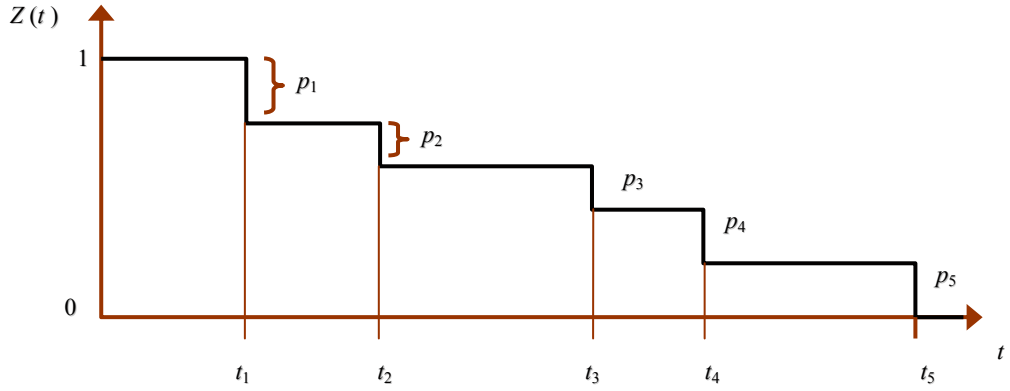
Die *Überlebenswahrscheinlichkeit* $Z(t)$ ist definiert durch

$$Z(t) = p(X \geq t) = \sum_{i|t_i \geq t} p_i = 1 - p(X < t) = 1 - F(t)$$

Hierin ist $F(t)$ die *Lebensdauerverteilung* $F(t) = \sum_{i|t_i < t} p_i$.

Es gilt

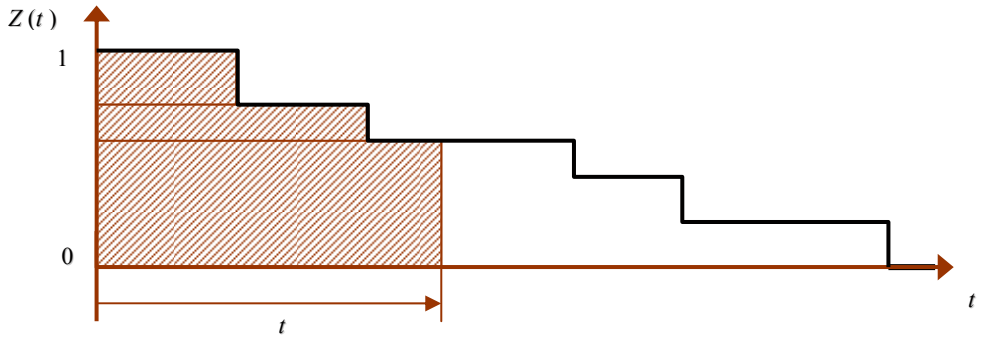
$$\int_0^t Z(t)\,\mathrm{d}t = \sum_{i|t_i<t} p_i \cdot t_i + \sum_{i|t\leq t_i} p_i \cdot t$$



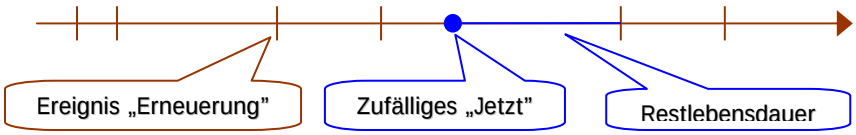
und die *mittlere Lebensdauer* τ der Objekte ist gegeben durch

$$\tau = \int_0^\infty Z(t)\,\mathrm{d}t = \sum_i p_i t_i \,,$$

wie man sich anhand der folgenden Grafik klar machen kann.



Aus dem schon lange andauernden *Erneuerungsprozess* wird nun ein Zeitpunkt rein zufällig ausgewählt. Das ist das zufällige „Jetzt“. Das Alter des gerade angetroffenen Objekts ist eine Zufallsvariable, sie erhält das Symbol T_A. Die Restlebensdauer wird mit T_R bezeichnet.



Nun geht es darum, die Verteilungsfunktionen $F_A(t)$ bzw. $F_R(t)$ dieser Zufallsvariablen zu ermitteln.

Mit q_i bezeichnen wir die Wahrscheinlichkeit, mit dem zufälligen „Jetzt“ auf ein Objekt der Lebensdauer t_i zu treffen. Für die Berechnung dieser Größe stellen wir uns einen großen Zeitraum vor, in dem schon viele Erneuerungen stattgefunden haben. Sei n_j die Anzahl aller Intervalle der Länge t_j und n die Gesamtzahl der Intervalle. Die Wahrscheinlichkeit, zufällig auf ein Objekt der Lebensdauer t_i zu treffen, ist gleich dem relativen Anteil dieser Intervalle am gesamten betrachteten Zeitraum.

Das gesuchte Verhältnis q_i ist gleich $n_i t_i/(\Sigma_j\, n_j t_j)$. Teilt man Zähler und Nenner durch n und nutzt man aus, dass für große n wenigstens näherungsweise $p_j = n_j/n$ ist, so erhält man für das gesuchte Verhältnis den Wert $p_i t_i/\tau$. Die Wahrscheinlichkeit, zufällig auf ein Intervall der Länge t_i zu stoßen, ist demnach gleich

$$q_i = p_i t_i/\tau.$$

Wegen der Zufälligkeit des Zeitpunkts, zu dem das Objekt angetroffen wird, kann das Alter des angetroffenen Objekts als gleich verteilt angenommen werden: Die Wahrscheinlichkeit, dass das angetroffene Objekt mit der Lebensdauer t_i ein Alter kleiner t hat, ist gleich t/t_i. (Das gilt unter der Voraussetzung, dass $t \le t_i$; andernfalls ist diese Wahrscheinlichkeit gleich 1. Im folgenden wird die Bedingung $t \le t_i$ unausgesprochen vorausgesetzt.) Die Wahrscheinlichkeit, dass das Objekt eine Restlebensdauer kleiner als t hat, ist gleich der Wahrscheinlichkeit, dass das Alter größer oder gleich t_i-t ist. Und diese Wahrscheinlichkeit ist das Komplement zu eins der Wahrscheinlichkeit, dass das Alter kleiner als t_i-t ist. Und die Letztere ist gleich $(t_i\text{-}t)/\, t_i$. Insgesamt ist die Wahrscheinlichkeit für eine Restlebensdauer kleiner t gleich $1\text{-}(t_i\text{-}t)/\, t_i$. Und dieser Wert ist wiederum gleich t/t_i. Man erhält also dasselbe Ergebnis wie für das Alter.

Der Wert t/t_i ist die Wahrscheinlichkeit dafür, dass das Alter des Objekts (oder seine Restlebensdauer) kleiner als t ist, unter der Bedingung, dass die Lebensdauer des Objekts gleich t_i ist. (Gilt, wie gesagt, für $t \le t_i$. Ansonsten ist diese bedingte Wahrscheinlichkeit gleich 1.)

Bereits jetzt ist zu erkennen, dass das Alter und die Restlebensdauer dieselbe Verteilung besitzen.

Die *Verteilungsfunktion des Alters* lässt sich nun angeben, indem man alle diese bedingten Wahrscheinlichkeiten mit den Wahrscheinlichkeiten für das Auftreten der jeweiligen Lebensdauer multipliziert und alle diese Produkte über alle möglichen Lebensdauern summiert:

$$p(T_A < t) = \sum_{i|t_i<t} q_i \cdot 1 + \sum_{i|t\le t_i} q_i \cdot t / t_i = (\sum_{i|t_i<t} p_i \cdot t_i + \sum_{i|t\le t_i} p_i \cdot t) / \tau$$

Der geklammerte Ausdruck ist gleich $\int_0^t Z(t)\,\mathrm{d}t$. Daraus folgt

$$F_A(t) = p(T_A < t) = \frac{1}{\tau} \cdot \int_0^t Z(t)\,\mathrm{d}t$$

Dementsprechend ist die *Verteilungsfunktion der Restlebensdauer* gleich

$$F_R(t) = p(T_R < t) = \frac{1}{\tau} \cdot \int_0^t Z(t)\,\mathrm{d}t$$

Die zugehörigen Verteilungsdichten sind $f_A(t) = f_R(t) = Z(t)/\tau$.

Das sind grundlegende Formeln der *Erneuerungstheorie* (Bratley/Fox/Schrage, 1987, S. 113 ff; Kleinrock, 1975, Band I, S. 169 ff.).

Beispiel: Mietwagenunternehmen

Ein Mietwagenunternehmen unterhält einen Park von komfortablen Wohnmobilen. Kein Fahrzeug bleibt länger als 2 Jahre im Einsatz. Der größere Teil ist sogar schon früher nicht mehr wirtschaftlich verwendbar, so dass laufend Ersetzungen erforderlich werden. Mehrjährige Beobachtungen ergaben die Wahrscheinlichkeitsverteilung für die wirtschaftliche Lebensdauer der Wohnmobile (Tabelle1 und Bild). Gesucht ist die Verteilung der Lebensdauern (Alterspyramide), die sich im Laufe der Zeit einstellt. Die Erneuerung findet immer am Anfang eines Zweimonatsintervalls statt. Zu dem Zeitpunkt wird die Statistik geführt, und zwar unmittelbar vor der Erneuerung.

Lösung: Die mittlere Lebensdauer beträgt 17 Monate. Die Alterspyramide (Verteilung des Alters) ist in Tabelle 2 und im Bild zu sehen. Literaturhinweise: Sasieni, Yaspan, Friedman (1965, S. 118 ff.).

Tabelle 1 Lebensdauerverteilung der Wohnmobile

Anzahl Monate Zeit bis zum Ersatz	Wahrscheinlichkeit
10	5 %
12	10 %
14	15 %
16	20 %
18	20 %
20	15 %
22	10 %
24	5 %

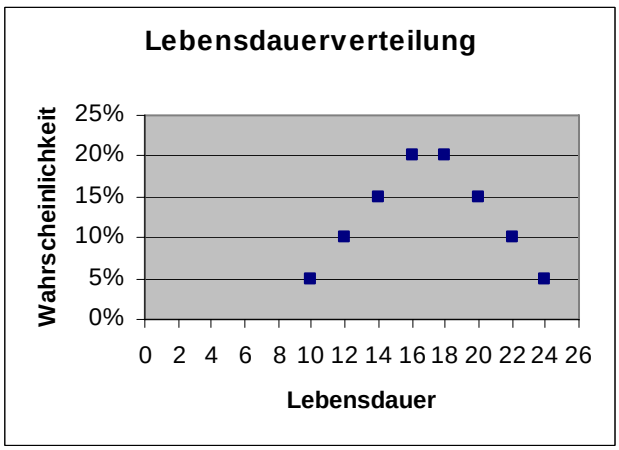
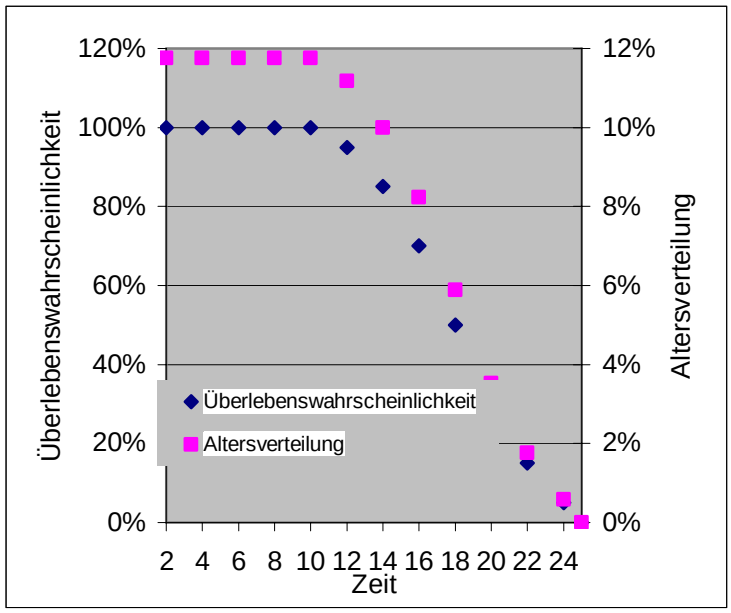


Tabelle 2 Altersverteilung der Wohnmobile

Lebensalter (Monate)	Wahr-scheinlich-keit
2	11.76 %
4	11.76 %
6	11.76 %
8	11.76 %
10	11.76 %
12	11.18 %
14	10.00 %
16	8.24 %
18	5.88 %
20	3.53 %
22	1.76 %
24	0.59 %



Simulation nichthomogener Poisson-Prozesse: Die Rücksetzmethode

Zu simulieren sei ein Poisson-Prozess. Unter einem Poisson-Prozess verstehen wir einen Erneuerungsprozess, dessen Intervalllängen (Lebensdauern) exponentialverteilt sind (Kleinrock, Band I, 1975, S. 24 ff. und S. 60 ff.):

$$F(t) = 1 - e^{-\lambda t}.$$

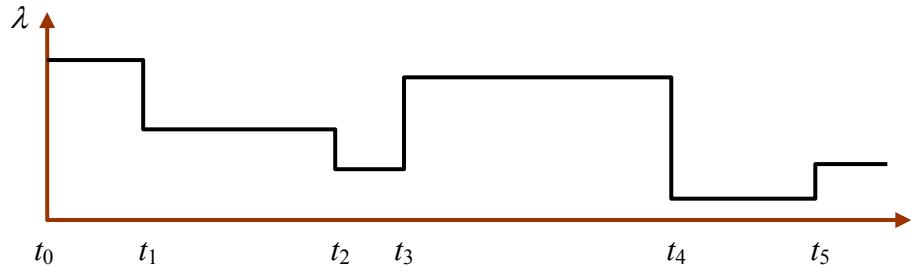
Dabei ist die Erneuerungsrate λ gleich dem Kehrwert der mittleren Lebensdauer:

$$\lambda = 1/\tau$$

Die Erneuerungsrate ist der Parameter des Poisson-Stroms. Er kann auch als Intensität des Ereignisstroms aufgefasst werden, da er die Anzahl der je Zeiteinheit zu erwartenden Ereignisse angibt.

Zur Simulation ermittelt man über die „Methode der Umkehrfunktion“ die Folge der Lebensdauern und damit die Folge der Erneuerungszeitpunkte.

Komplizierter wird’s, wenn der Parameter des Poisson-Prozesses nicht mehr konstant, sondern zeitabhängig ist: $\lambda = \lambda(t)$. Wir sprechen dann von einem inhomogenen Poisson-Prozess. Vorausgesetzt sei nun, dass $\lambda(t)$ eine Treppenfunktion ist.



Für die Zeit bis zur ersten Änderung ist $\lambda(t) = \lambda_1$, von da bis zur zweiten Änderung setzen wir $\lambda(t) = \lambda_2$. Also: $\lambda(t) = \lambda_i$ für $t_{i-1} \leq t < t_i$.

Die Aufgabe besteht nun darin, einen Poisson-Strom mit sich sprunghaft ändernden Erneuerungsraten zu simulieren. Das kann man nach den Methoden aus dem Buch von Bratley, Fox und Schrage (1987, S. 178 ff.) tun. Aber es geht einfacher. Nämlich nach einem *Rücksetzverfahren*:

Angefangen beim Zeitnullpunkt zieht man sukzessive Lebensdauern entsprechend der Exponentialverteilung mit dem Parameter λ_1. Die akkumulierten Lebensdauern liefern die entsprechenden Erneuerungszeitpunkte (Ereigniszeitpunkte). Das macht man solange, bis ein Ereignis die Zeitgrenze t_1 überschreitet. Dieses letzte Ereignis streicht man aus der Folge der Ereignisse wieder heraus und zieht die nächste Lebensdauer zum Parameter λ_2. Diese wird bei t_1 angesetzt (zu t_1 addiert) und so erhält man den nächsten Ereigniszeitpunkt, der anstelle des gestrichenen steht, usw.

Eine einfache Begründung für dieses Vorgehen liefert das diskrete Modell: Die Zeitachse wird in kleine Zeitschritte der Dauer Δt aufgeteilt. Die Wahrscheinlichkeit für das Eintreffen des Ereignisses in einem solchen Intervall ist gleich $\Delta t\lambda_i$, wobei λ_i der in diesem Intervall gerade gültige Parameter ist. Man „erwürfelt" nun für jeden dieser Abschnitte, ob das Ereignis eintritt oder nicht. Das „Rücksetzverfahren" entspricht nun dem Vorgehen, dass man zunächst für die Intervalle nach der Wahrscheinlichkeit $\Delta t\lambda_1$ würfelt, ob ein Ereignis im Intervall stattfindet oder nicht. Man hört erst auf, wenn das erste Ereignis nach Überschreiten der Zeitgrenze aufgetreten ist. Dann geht man zurück bis zur Zeitgrenze und fängt dort wieder mit dem Würfeln an; jetzt aber gemäß der Wahrscheinlichkeit $\Delta t\lambda_2$ für das Eintreten des Ereignisses.

Das ist aber - statistisch gesehen - offensichtlich dasselbe, als hätte man direkt an der Zeitgrenze die neue Statistik zu Grunde gelegt und nicht erst nach der alten Statistik weitergewürfelt. Letzteres ist aber genau das den sich ändernden Erneuerungsraten angemessene Vorgehen. Das heißt: Das Rücksetzverfahren ist korrekt.

Das „Paradoxon der Restlebensdauer"

In der Rücksetzmethode zur Simulation inhomogener Poisson-Prozesse scheint ein Widerspruch zu stecken. Angenommen es ist $\lambda_1 = \lambda_2 = \lambda$. Dann bedeutet das erneute Ansetzen an der Zeitgrenze t_1, dass die Zwischenankunftszeit der beiden Ereignisse, zwischen denen die Zeitgrenze liegt, systematisch vergrößert wird. Hätte man einen „durchgehenden" Poisson-Prozess, wäre dem „offensichtlich" nicht so.

Die zuletzt geäußerte Vermutung ist falsch. Tatsächlich entspricht die Situation dem so genannten „Paradoxon der Restlebensdauer" (Kleinrock, Band I, 1975, S. 169 ff.). Dieses Paradoxon löst sich auf, wenn man bedenkt, dass bei zufälliger Wahl eines Zeitpunkts auf der Zeitachse, dieser Zeitpunkt wohl bevorzugt in eines der längeren Intervalle fallen wird.

Diese Tatsache lässt sich mit den obigen Betrachtungen zur Alterspyramide präzisieren: Die Verteilungsdichte der ausgewählten Intervalle ist gleich $\lambda^2 te^{-\lambda t} = te^{-t/\tau}/\tau^2$. Der Erwartungswert der Länge der ausgewählten Intervalle ergibt sich daraus zu 2τ. Tatsächlich ist also - entgegen der zum Paradoxon führenden Vermutung - der Erwartungswert derjenigen Intervalle, in die eine Zeitgrenze fällt, doppelt so groß wie der Erwartungswert eines beliebigen Intervalls.

Die Verteilungsdichte der Restlebensdauer ergibt sich zu

$$f_R(t) = (1-F(t))/\tau = (1-(1-e^{-t/\tau}))/\tau = e^{-t/\tau}/\tau = f(t).$$

Das entspricht genau den Annahmen der Rücksetzmethode: Die Restlebensdauer ab der Zeitgrenzen besitzt dieselbe Verteilung wie die Lebensdauern insgesamt.

Sachverzeichnis