Aufgabenblatt 1

Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 10.11.2017 13:00 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben und während der Übung präsentieren können.
- Ihre Programme müssen kompilierbar und ausführbar sein.
- Ändern Sie bitte nicht die Dateinamen und die vorhandene Ordnerstruktur.
- Bei manchen Aufgaben finden Sie Zusatzfragen. Diese Zusatzfragen beziehen sich thematisch auf das erstellte Programm. Sie müssen diese Zusatzfragen in der Übung beantworten können.
- Verwenden Sie, falls nicht anders angegeben, für alle Ausgaben `System.out.println()`.
- Verwenden Sie für die Lösung der Aufgaben keine Aufrufe (Klassen) aus der Java-API, außer diese sind ausdrücklich erlaubt.

Aufgabe 1

Erweitern Sie die main-Methode um folgende Funktionalität:

- Implementieren Sie ein Programm, welches das in Abbildung 1 gezeigte Bild ausgibt.
- Im Bild sind alle Maßangaben vorhanden, die notwendig sind, um dieses Bild zu erzeugen.
- Das gesamte Bild hat eine Größe von 500×500 Pixel und der Punkt $P(x = 0, y = 0)$ befindet sich in der unteren linken Ecke.
- Mit `StdDraw.setPenRadius(0.01)` wird die Linienbreite für das gesamte Bild auf 0.01 gesetzt. Sie können diese Linienbreite anpassen, falls das für die Generierung des Bildes notwendig sein sollte.
- `StdDraw.setPenColor(StdDraw.BLUE)` setzt die aktuelle Zeichenfarbe auf blau. Andere Farben können entsprechend gesetzt werden.
- Verwenden Sie für die diagonalen Linien eine geeignete Schleife.
- Die Dokumentation unter dem Link `http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html` kann Ihnen dabei helfen, diese Aufgabe zu lösen.

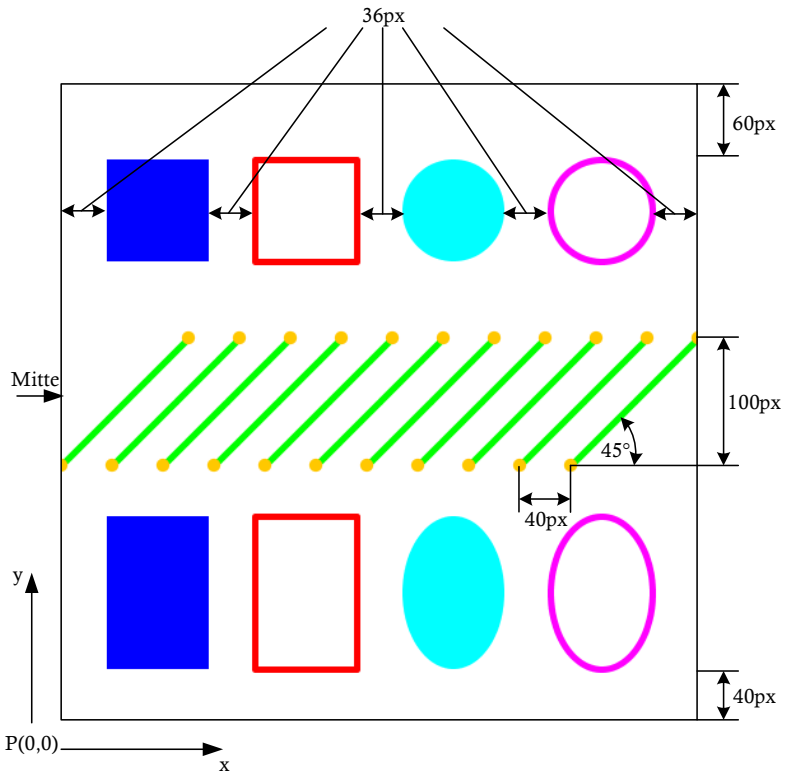


Abbildung 1: Ergebnisbild mit den entsprechenden Maßangaben

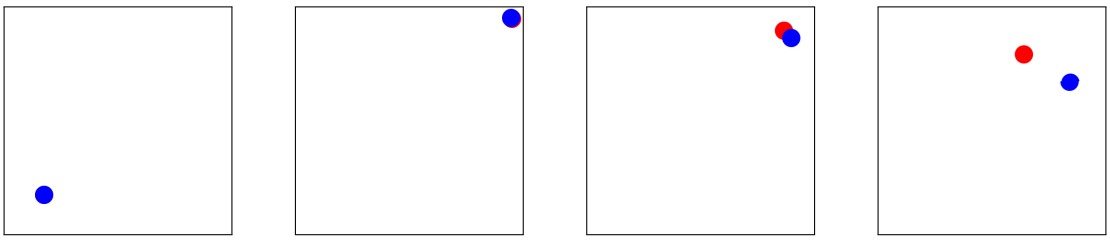
Zusatzfrage:

1. Sind verschiedene Arten von Schleifen gegeneinander austauschbar?

Aufgabe 2

Erweitern Sie die main-Methode um folgende Funktionalität:

- Implementieren Sie fliegende Kreise, wie in Abbildung 2a–2d gezeigt.
- Das gesamte Bild hat eine Größe von 500×500 Pixel und der Punkt $(0, 0)$ befindet sich in der unteren linken Ecke.
- Der rote und blaue Kreis starten beide zu Beginn mit ihrem Mittelpunkt beim Punkt $(0, 0)$ und haben einen Radius von 20 Pixel.
- Die Kreismittelpunkte werden innerhalb einer Schleife bei jedem Schleifendurchlauf um ihre Änderungswerte in x- und y-Richtung verschoben. Die Änderungswerte in x- und y-Richtung sind zu Beginn für beide Kreise auf den Wert 2 gesetzt.
- Implementieren Sie nach jedem Schleifendurchlauf eine Wartezeit zwischen zwei Bildern von $10ms$ (siehe `StdDraw.pause(int t)`).
- Wenn die Kreise nun auf eine Begrenzung des Bildfensters stoßen (Abbildung 2b), dann wird das Vorzeichen der Richtungsänderung geändert. Für die linke/rechte Begrenzung für den x-Wert und für die obere/untere Begrenzung für den y-Wert.
- Zusätzlich werden nach dem Aufprall die Änderungswerte beider Richtungen um einen zufälligen Wert geändert. Die Änderung soll maximal 2 Pixel ausmachen und soll zufällig (siehe `Math.random()`) berechnet werden. Zusätzlich muss nach einem Aufprall verhindert werden, dass sich die Kreise über die Begrenzung hinaus bewegen.
- Wie in Abbildung 2c gezeigt, werden beide Kreise individuell verarbeitet, d.h. es können nach dem ersten Auftreffen am Bildrand unterschiedliche Flugrichtungen entstehen.
- Wenn der Änderungswert einer Richtung größer gleich 5, oder kleiner gleich 0 wird, dann wird der Änderungswert jener Richtung auf 2 zurückgesetzt. Das soll verhindern, dass z.B. die Bewegungen der Kreise zu schnell werden.
- Hinweis: Für ein ruckelfreies Fliegen können Sie sogenanntes *DoubleBuffering* anwenden. Mehr Informationen dazu finden Sie unter `http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html`.



(a) (b) (c) (d)

Abbildung 2: Bildausschnitte des Ergebnisses. Bilder 2a–2d zeigen die fliegenden Kreise.

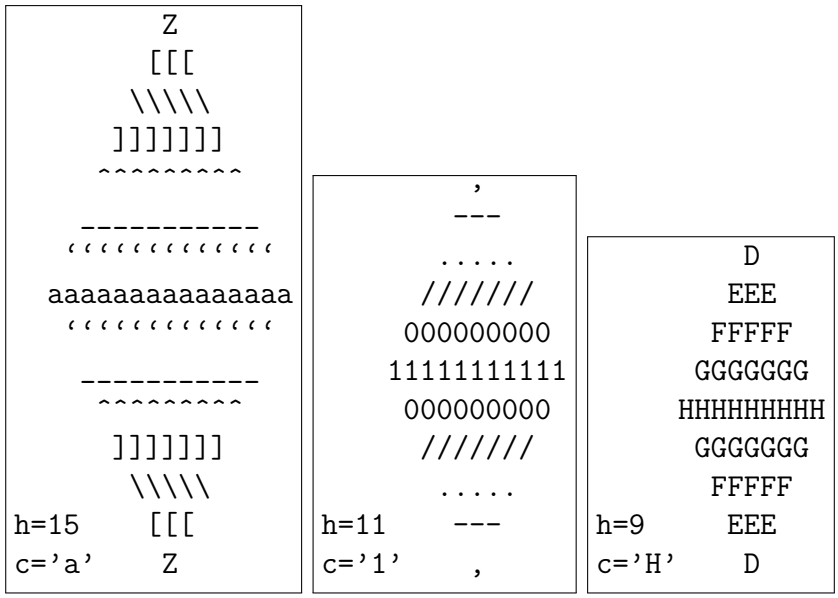
Zusatzfrage:

1. Wann ist die Verwendung einer while-Schleife von Vorteil, und wann die einer for-Schleife?

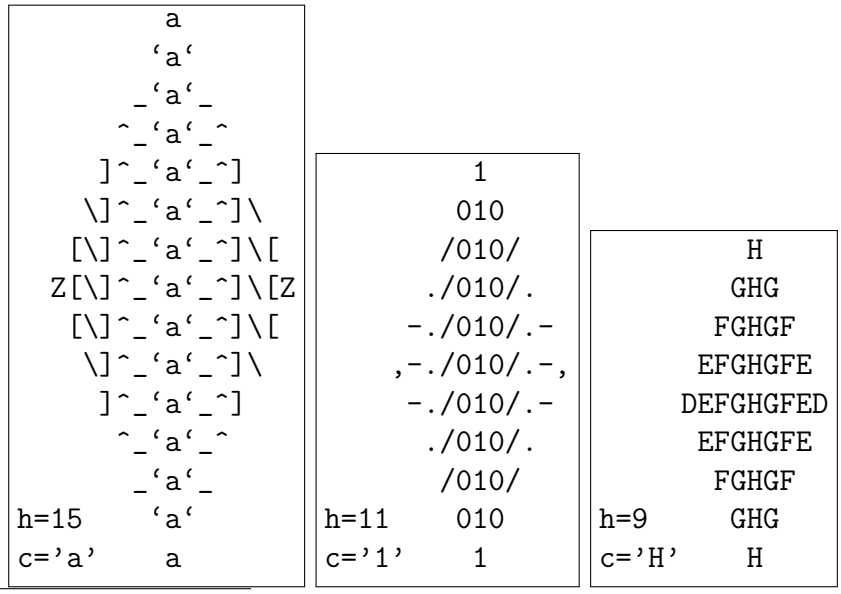
Aufgabe 3

Erweitern Sie die main-Methode um folgende Funktionalität:

- Implementieren Sie die Ausgabe einer Raute bestehend aus ASCII-Zeichen [1].
- Die Ausgabe wird mit drei Parametern gesteuert. Die Höhe ist eine positive ganzzahlige Variable `int h` und steuert die Anzahl der Zeilen, die die Raute umfassen soll. Als Höhe sollen nur ungerade Zahlen akzeptiert werden, ansonsten wird das Programm mit einer entsprechenden Meldung abgebrochen. Eine zweite Variable `char c` bestimmt das ASCII-Zeichen entlang der Horizontalen in der Mitte der Raute, falls eine dritte Variable `boolean printVertical` den Wert `false` aufweist. Beispiele für horizontale Ausgaben:



- Die zweite Variable `char c` bestimmt das ASCII-Zeichen entlang der Vertikalen in der Mitte der Raute, falls die dritte Variable `boolean printVertical` den Wert `true` aufweist. Beispiele für vertikale Ausgaben:



[1] ASCII-Code: `https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange`

- Bei horizontaler Ausrichtung wird ausgehend vom Zeichen `char c` in der Mitte der Raute bei jeder Zeile nach oben bzw. unten das jeweils nachfolgend kleinere Zeichen in der ASCII-Tabelle verwendet. Verwendet man eine vertikale Ausrichtung, wird ausgehend vom Zeichen `char c` in der Mitte der Raute bei jeder Spalte nach links bzw. rechts das jeweils nachfolgend kleinere Zeichen in der ASCII-Tabelle verwendet. Zum Beispiel entspricht `B` dem Wert 66 in der ASCII-Tabelle und das nächstkleinere mit dem Wert 65 steht für das Zeichen `A`, 64 steht für `@`, 63 steht für `?` usw.

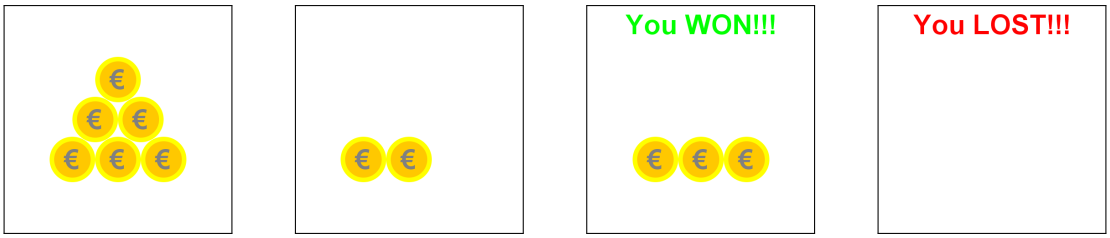
Zusatzfrage:

1. Wie viele Schleifen braucht man mindestens, um dieses Beispiel zu lösen?

Aufgabe 4

Erweitern Sie die main-Methode um folgende Funktionalität:

- Implementieren Sie das *Guessing Game* aus der Vorlesung und erweitern Sie es um grafische Komponenten. Verwenden Sie, wie in der Vorlesung vorgestellt, den Scanner, um die Daten von der Konsole einzulesen.
- Die generierte Zufallszahl (siehe `Math.random()`) soll zwischen 1 und 100 liegen und man hat maximal 6 Versuche um die Zahl zu erraten.
- Für die grafische Ausgabe legen Sie ein Fenster der Größe 500×500 Pixel an.
- Wie in Abbildung 3a gezeigt, werden die verbleibenden Versuche mittels Münzen dargestellt. Für jeden falschen Versuch wird eine Münze entfernt (siehe Abbildung 3b). Das Spiel ist zu Ende, wenn man die Zahl erraten bzw. alle Versuche aufgebraucht hat. Die grafische Ausgestaltung und Formatierung (inkl. Position) der Münzen bleibt Ihnen überlassen. Hinweis: Folgende Formatierungsdaten wurden für die Darstellung der hier gezeigten Münzen verwendet:
  - Schriftgröße 60 Pixel, Schriftart *Arial* und Farbe `StdDraw.GRAY`
  - Äußerer Münzenradius 50 Pixel und Farbe `StdDraw.YELLOW`
  - Innerer Münzenradius 40 Pixel und Farbe `StdDraw.ORANGE`
- Zusätzlich soll die Information, ob man gewonnen hat, mit `You WON!!!`, wie in Abbildung 3c gezeigt, realisiert werden. Äquivalent dazu soll der Schriftzug `You LOST!!!` (Abbildung 3d) angezeigt werden, wenn man verloren hat.
- Wie in der Vorlesung demonstriert, soll nach jedem Rateversuch ein Hinweis gegeben werden, ob die gesuchte Zahl größer oder kleiner ist, als die zuletzt eingegebene Zahl.
- Falsche Eingaben wie Fließkommazahlen oder Zeichen werden beim Einlesen ignoriert und es geht kein Rateversuch (Münze) verloren. Auch die Eingabe einer Zahl kleiner 1 oder größer 100 wird beim Einlesen ignoriert.



(a) (b) (c) (d)

Abbildung 3: Bildausschnitte der Spielzustände.

Zusatzfragen:

1. Warum muss eine ungültige Eingabe aus dem Input-Stream des Scanners entfernt werden?
2. Wie kann man die eingegebenen Daten (bzw. deren Datentypen) unterscheiden?

Aufgabe 5

Erweitern Sie die main-Methode um folgende Funktionalität:

- Implementieren Sie die in Abbildung 4 gezeigte optische Täuschung, basierend auf der *Ouchi-Illusion*[2].
- Setzen Sie die Fenstergröße auf 800×800 Pixel. Die einzelnen (weißen und schwarzen) Rechtecke haben eine Abmessung von 32×8 Pixel.
- Das Quadrat in der Mitte des Fensters hat eine Größe von 200×200 Pixel. Die Rechtecke im inneren Quadrat sind um 90° gedreht und haben eine Abmessung von 8×32 Pixel.

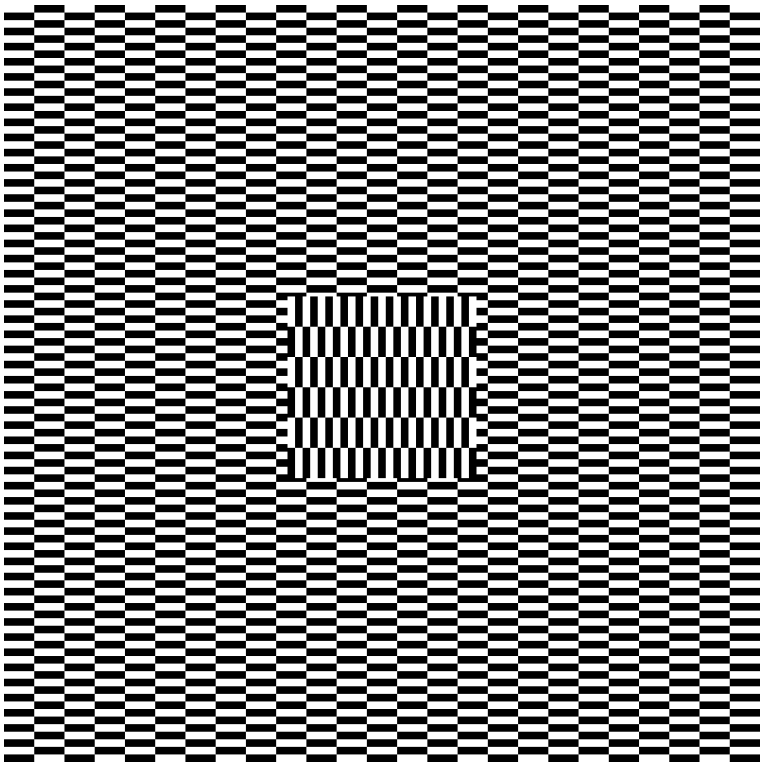


Abbildung 4: Ergebnis der Ouchi-Illusion laut den entsprechenden Vorgaben.

Zusatzfrage:

1. Wenn Sie for-Schleifen verwendet haben, überlegen Sie: Ist es sinnvoll und möglich, diese durch while-Schleifen zu ersetzen?

[2]Die *Ouchi-Illusion* wurde nach seinem Entdecker, dem japanischen Künstler Hajime Ouchi, benannt.