SoSe 2019

Prof. Dr. Margarita Esponda

Objektorientierte Programmierung

6. Übungsblatt

Abgabe am 11. Juni, um 12:10 Uhr in der Vorlesung und Online

Ziel: Erste Auseinandersetzung mit der Java-Syntax, Objekten und Klassen.

1. Aufgabe (3 Punkte)

Die Multiplikation von zwei natürlichen Zahlen kann als Reihensumme dargestellt werden und mit einer naiven Implementierung berechnet werden, die rekursiv oder iterativ programmiert werden kann.

Es gibt aber bessere Methoden, um zwei Zahlen zu multiplizieren, die auch bei Hardware-Implementierungen verwendet werden. Eine davon ist die berühmte russische Multiplikation. Der Name kommt von der Tatsache, dass diese Methode unter russischen Bauern üblich war.

Bauern-Multiplikation:

Wenn beispielsweise die Zahlen **a** und **b** multipliziert werden müssen, wird:

1) die Zahl **a** in eine Spalte so lange halbiert bis diese den Wert **1** oder **0** erreicht hat. Gleichzeitig wird die Zahl **b** in einer zweiten Spalte so lange verdoppelt.

2) Die Zeilen, bei denen sich in der ersten Spalte gerade Zahlen befinden, werden gestrichen.

3) Das Ergebnis der Multiplikation ergibt sich aus der Summe der übrig gebliebenen Zahlen der zweiten Spalte.

Beispiel:

45 x 33

```
      45        33
     ~~22~~     ~~66~~
      11       132
       5       264
      ~~2~~     ~~528~~
       1       1056
  ----------------------
            1485
```

Schreiben Sie eine Java statische Methode, die den Bauern-Algorithmus verwendet, um zwei Zahlen zu multiplizieren. Sie dürfen selbstverständlich keine Multiplikation oder Division dabei benutzen, in Ihrer Funktion sind nur Bit-Operationen, Vergleiche und Summen erlaubt.

Analysieren Sie die Komplexität Ihres Algorithmus mit Hilfe der O-Notation und begründen Sie Ihre Antwort.

2. Aufgabe (6 Punkte)

Schreiben Sie ein Java-Programm, dass das Drei-Türen-Spiel oder Monty-Hall-Spiel simuliert.

Spielregeln:

Ein Auto und zwei Ziegen werden zufällig hinter drei Tore platziert, die geschlossen sind.

Der Spieler, wählt ein Tor aus, das aber vorerst verschlossen bleibt.

Der Moderator öffnet von den zwei nicht gewählten Toren, das Tor hinter dem sich kein Auto befindet und fragt den Kandidat, ob er bei seiner Wahl bleibt.

Der Spieler entscheidet zufällig, ob er seine Entscheidung ändere oder nicht.

Lassen Sie mit Hilfe von Zufallszahlen das Spiel simulieren und berechnen Sie wie oft der Spieler gewinnt.

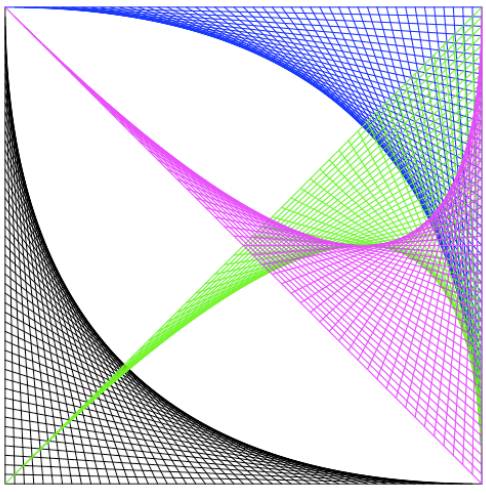
a. Wenn er seine Entscheidung nie ändert.

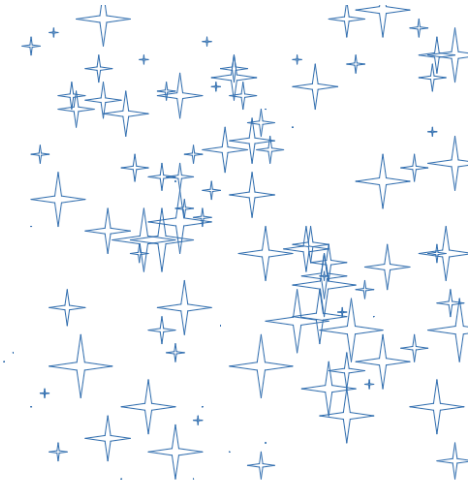
b. Wenn er seine Entscheidung immer ändert.

c. Wenn er jedes Mal zufällig seine Entscheidung ändert.

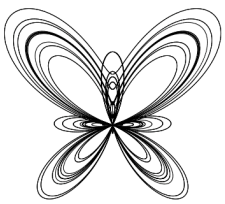
3. Aufgabe (12 Punkte)

a) Schreiben Sie unter Verwendung der **StdDraw**-Klasse (siehe unsere Veranstaltung-Seite) zwei eigene Klassen, die folgende Bilder produzieren.



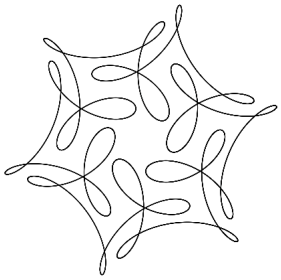


b) Definieren Sie eine Butterfly-Klasse, die die bekannte *Butterfly* parametrische Funktion zeichnen kann.



$$x = \sin(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right)$$
$$y = \cos(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right)$$

c) Definieren Sie unter Verwendung der StdDraw-Klasse eine **Spirograph**-Klasse, in der folgende parametrische Funktion gezeichnet wird.



$$x = a \cdot \cos(t) + b \cdot \cos(m \cdot t) + c \cdot \sin(n \cdot t)$$
$$y = a \cdot \sin(t) + b \cdot \sin(m \cdot t) + c \cdot \cos(n \cdot t)$$
mit
$$a = 1, \quad b = \frac{1}{2}, \quad c = \frac{1}{3}$$
$$m = 7, \quad n = 17, \quad -5 < t < 5$$

Zwischen b) und c) muss nur eins von beiden gelöst und abgegeben werden.

4. Aufgabe (15 Punkte)

a) (2 P.) Programmieren Sie die Klasse **Circle**, die einfache Kreis-Objekte darstellt.

b) (1 Punkte) Folgende zwei Konstruktoren sollen in der **Circle** Klasse beinhaltet sein:

```
public  Circle( double  x, double  y, double  radius) {
        this.x = x;
        this.y = y;
        this.radius = radius;
}
public  Circle() {
        this ( 0, 0, 10);
}
```

c) (12 P.) Erweitern Sie die **Circle**-Klasse um folgende Instanz-Methoden.

```
/* Ein zweites identisches Circle-Objekt wird erstellt */
public Circle clone()

/*  Ein Circle-Objekt vergleicht seine Fläche mit der Fläche eines zweiten
   Circle-Objekts c, das als Parameter übergeben wird und ergibt -1, 0, oder 1, je nachdem,
   ob die Fläche jeweils kleiner, gleich oder größer ist. */
public int compareTo( Circle c )

/* Testet, ob ein Circle c komplett in dem Circle-Objekt, das gerade
   die Methode ausführt, beinhaltet ist */
public boolean contains( Circle c)

/* Ein Circle-Objekt überprüft, ob eine Überlappung mit dem Circle c
   existiert. */
public boolean overlaps( Circle c )
```

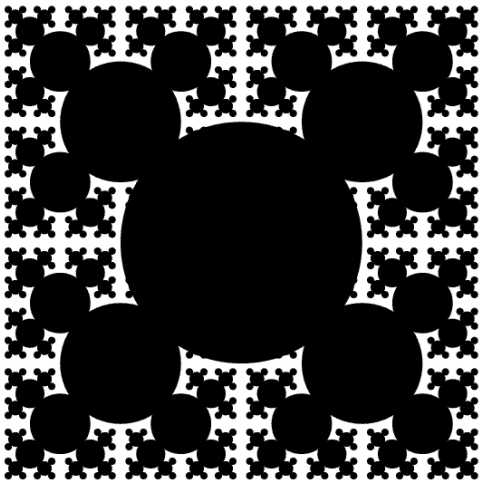
```
/* Berechnet das kleinste Rectangle-Objekt, das ein Array von Circle-Objekten umrahmen
   kann. */
public static Rectangle smallestBoundingRectangle ( Circle[] circles )
```

d) Laden Sie die Klasse **TestCircle** aus den Veranstaltung-Ressourcen runter, und starten Sie die Programmausführung mit der **TestCircle**-Klasse.

Innerhalb der **TestCircle**-Klasse werden Ihre Methoden getestet.

5. Aufgabe (5 Punkte)

Mit Hilfe der StdDraw.java Klasse schreiben Sie eine rekursive Funktion, die folgendes "Mickey mouse"-Fraktal malt.



6. Aufgabe (12 Bonuspunkte)

Positive und negative ganze Zahlen können mit Hilfe von Zahlenpaaren aus

natürlichen Zahlen dargestellt werden (Unterlagen), wobei das erste Element des Paares die negativen Zahlen und das zweite Element die positiven Zahlen darstellt.

Eine Zahl ist gleich dem Paar (a, b) = b - a

Z.B. (3, 0) = -3

(0, 3) = 3

Definieren Sie eine Java-Klasse **ZInteger**, in der Sie diese Art der Zahlendarstellung mit folgenden Eigenschaften modellieren.

a) Eine **ZInteger**-Zahl ist nicht veränderbar (unmutable).

b) **ZInteger**-Zahlen können addiert, subtrahiert und multipliziert werden und dabei werden immer neue Zahlen als Ergebnisse erzeugt.

c) Die Darstellung einer **ZInteger**-Zahl ist nicht eindeutig, weil jede Zahl unendlich viele Darstellungsmöglichkeiten hat. Die Zahl -2 kann z.B. mit (2,4) oder (4,6) dargestellt werden. Schreiben Sie dann eine **simplify** Hilfsmethode, die nach jeder Operation die Darstellung berechnet, bei der das Zahlenpaar am kleinsten ist. Z.B. simplify (3, 7) => (0, 4)

d) Definieren Sie die Vergleichsoperationen (==, <, <=, >, >=)

e) Definieren Sie einen geeigneten Konstruktor, in dem automatisch verhindert wird, dass ungültige Zahlenpaare eingegeben werden.

f) Definieren Sie eine **toString**-Methode, so dass eine **ZInteger**-Zahl als Tupel ausgegeben wird (mit runden Klammern und Komma dazwischen).

g) Schreiben Sie eine **TestZInteger**-Klasse, in der Sie Ihre **ZInteger**-Klasse ausführlich testen.

7. Aufgabe (3 Punkte)

a) Verwandeln Sie die **for**-Schleife der mersenne-Funktion in eine **while**-Schleife

b) Überlegen Sie sich geeignete Vorbedingungen, Nachbedingungen und Invarianten für die while-Schleife und testen Sie diese mit entsprechenden assert-Anweisungen.

```
def mersenne(k):
    """ Berechnet die Mersenne-Zahlen 2**k-1 """
    base = 2
    for i in range(k):
        mersenne = base-1
        base <<= 1
    return mersenne
```

Wichtige Hinweise für die Java-Programmierung:

1) Verwenden Sie selbsterklärende Namen von Variablen und Methoden.
2) Für die Namen aller Bezeichner müssen Sie die Java-Konventionen verwenden.
3) Verwenden Sie vorgegebene Klassen- und Methodennamen.
4) Methoden sollten klein gehalten werden, sodass auf den ersten Blick ersichtlich ist, was diese Methode leistet.
5) Methoden sollten möglichst wenige Argumente haben.
6) Methoden sollten entweder den Zustand der Eingabeargumente ändern oder einen Rückgabewert liefern.
7) Verwenden Sie geeignete Hilfsvariablen und definieren Sie sinnvolle Hilfsmethoden in Ihren Klassendefinitionen.
8) Zahlen sollten durch Konstanten ersetzt werden.
9) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.