Programmieren und Software-Engineering II
Übung 11

Name: _____ Klasse: _____ Datum: _____

Aufgabe 1: Schneckenrennen
Erstelle ein Programm zur Simulation eines Schneckenrennens. Dazu sind folgende Klassen notwendig:

Klasse Snail
Erstelle eine Klasse `Snail`, um mit einer Schnecke an einem Schneckenrennen teilzunehmen.

Eine Schnecke hat folgende Eigenschaften:
- Name
- Maximalgeschwindigkeit (für die Berechnung der zurückgelegten Wegstrecke)
- Zurückgelegte Wegstrecke

Erstelle einen Konstruktor dem man Name und Maximalgeschwindigkeit übergeben kann.

Implementiere das nachfolgende Verhalten für die Schnecke:
- `creep()` – bewegt die Schnecke um eine zufällige Zahl größer 0 und kleiner ihrer Maximalgeschwindigkeit weiter.
- `toString()` – gibt alle Daten der Schnecke als String mit folgendem Format zurück:
  `{ <Name>, <Geschwindigkeit>, <Wegstrecke> }`
  Ausgabebeispiel: `{ Sissi, 7, 32 }`

Klasse SnailRun

Erstelle die Klasse `SnailRun` für das Schneckenrennen.

Ein Schneckenrennen hat folgende Eigenschaften:
- Name
- Anzahl an Teilnehmern
- Array mit den teilnehmenden Schnecken (Größe des Arrays wird fix angegeben - zB 10)
- Streckenlänge

Erstelle einen Konstruktor dem man zumindest den Namen und die Streckenlänge übergeben kann.

Implementiere das nachfolgende Verhalten für ein Schneckenrennen:
- `addSnail(Snail newSnail)` – Hinzufügen eines neuen Teilnehmers
- `toString()` – gibt alle Daten des aktuellen Rennens als String zurück
  TIPP: Verwende die `toString()`-Methode der `Snail`-Klasse für die beteiligten Schnecken
- `getWinner()` – liefert null, wenn noch keine Schnecke im Ziel ist und andernfalls die Gewinnerschnecke
- `letThemCreep()` – lässt alle Teilnehmer einmal kriechen. Gibt für jede Schnecke `toString()` auf der Konsole aus, um das Rennen zu verfolgen.
- `startRun()` – ruft so lange die Methode `letThemCreep()` auf, bis eine Schnecke das Ziel erreicht hat.

Klassen Bet/BettingOffice

Erstelle eine Klasse `Bet` und eine Klasse `BettingOffice` um Wetten für ein Schneckenrennen entgegenzunehmen.

Eine Wette hat folgende Eigenschaften:
- Spielername
- Name der Schnecke, auf die gewettet wird
- Wetteinsatz

Erstelle einen Konstruktor mit dem alle Eigenschaften initialisiert werden können.

Programmieren und Software-Engineering II
Übung 11

Implementiere das nachfolgende Verhalten für eine Wette:

- `toString()` – gibt alle Daten der Wette als String in folgendem Format zurück:
  `{ <Spielername>, <Schneckenname>, <Wetteinsatz> }`
  Ausgabebeispiel: `{ Spieler1, Sissi, 50 }`

Ein Wettbüro (`BettingOffice`) hat folgende Eigenschaften:

- Schneckenrennen (`SnailRun`)
- Anzahl an angenommenen Wetten
- Array mit den angenommenen Wetten (Größe des Arrays wird fix angegeben - zB 10)
- Faktor (Der Wetteinsatz wird mit diesem Faktor multipliziert und ergibt somit die Höhe des Gewinnes.)

Erstelle einen Konstruktor dem man das Schneckenrennen sowie den Faktor übergeben kann.

Implementiere das nachfolgende Verhalten für das Wettbüro:

- `addBet(Bet newBet)` – schließt eine neue Wette ab
- `toString()` – gibt die Daten des Wettbüros, die Daten des betreuten Schneckenrennens sowie sämtliche angenommenen Wetten als String zurück
- `execute()` – startet das Schneckenrennen, gibt die Gewinnerschnecke auf der Konsole aus und gibt die Gewinne aus

Teste das Programm mit folgender `main`-Methode:

```
public static void main(String[] args) {
    SnailRun snailRun1 = new SnailRun("Perger Schneckenmarathon", 42);
    BettingOffice bettingOffice = new BettingOffice(snailRun1, 2.0);
    bettingOffice.execute();
    System.out.println();

    SnailRun snailRun2 = new SnailRun("Iron Snail 2016", 100);
    Snail sissi = new Snail("Sissi", 17);
    snailRun2.addSnail(sissi);
    Snail franz = new Snail("Franz", 16);
    snailRun2.addSnail(franz);
    bettingOffice = new BettingOffice(snailRun2, 1.25);
    bettingOffice.execute();
    System.out.println();

    SnailRun snailRun3 = new SnailRun("3. International Schnecken Berglauf", 22);
    Snail helga = new Snail("Helga", 5);
    snailRun3.addSnail(helga);
    Snail knut = new Snail("Knut", 5);
    snailRun3.addSnail(knut);
    bettingOffice = new BettingOffice(snailRun3, 2.25);
    bettingOffice.addBet(new Bet("Player1", "Helga", 50));
    bettingOffice.addBet(new Bet("Player2", "Helga", 60));
    bettingOffice.addBet(new Bet("Player3", "Knut", 65));
    bettingOffice.execute();
    System.out.println();

    System.out.println(bettingOffice.toString());
}
```

Mögliche Erweiterungen

- Implementiere die Methode `removeSnail(String name)` in der Klasse `SnailRun` um eine Schnecke aus dem Rennen zu nehmen.
- Erweitere das Programm um eine variable Anzahl an Wetten bzw. Schnecken/Rennen. Das Array muss bei Bedarf automatisch vergrößert werden.
- Wie kannst du verhindern, dass dieselbe Schnecke für ein Rennen mehrmals eingetragen wird?
- Kriechen zwei Schnecken gleichzeitig ins Ziel, gewinnt die Schnecke, die von der Methode `getWinner()` zuerst gefunden wird. Das ist doch ganz schön ungerecht, oder?

Programmieren und Software-Engineering II
Übung 11

Aufgabe 2: Kassenbon
Erstelle ein Programm zur Simulation eines Zahlvorganges und zur anschließenden Erstellung eines Kassenbons.

```
****************************************
*                                      *
*  HTL Schulbuffet                     *
*  Machlandstraße 48                   *
*  4320 Perg                           *
*  07262/555 0815                      *
*  schulbuffet@htl-perg.ac.at          *
*                                      *
****************************************

                            BETRÄGE IN EUR
----------------------------------------
Cola 0,5l           2x1,20          2,40
Pizzaeck            1x1,90          1,90
M&Ms 200g           3x1,70          5,10
----------------------------------------
GESAMT                              9,40
----------------------------------------
----------------------------------------

GEGEBEN                            10,00
ZURÜCK                              0,60
----------------------------------------
MWST: 10%                           0,85
----------------------------------------
DATUM:                        04.01.2017
UHRZEIT:                           11:21
----------------------------------------
Vielen Dank für Ihren Einkauf!
----------------------------------------
```

Dazu sind folgende Klassen notwendig:

Shop: Erstelle eine Klasse `Shop`, die die Firmeninformation beinhaltet.

Ein Geschäft hat folgende Daten: Name, Straße + Hausnummer, PLZ + Wohnort, Optional: Telefonnummer, Optional: Email-Adresse

Erstelle sinnvolle Konstruktoren um auch die Angabe optionaler Angaben zu unterstützen.
HINWEIS: Ein Konstruktor kann einen anderen Konstruktor der gleichen Klasse aufrufen!

Implementiere eine Methode `toString()` für die Ausgabe der Shop-Daten.

ReceiptItem: Erstelle die Klasse `ReceiptItem` für einen einzelnen Eintrag auf dem Kassenbon.

Ein einzelner Eintrag hat folgende Eigenschaften: Name des Produkts, Anzahl, Einzelpreis in Cent, Mehrwertsteuersatz (zB 10, 20)

HINWEIS: Aufgrund der Gleitkommaproblematik empfiehlt es sich die Preise in Cent anzugeben und erst für die Ausgabe in EUR umzurechnen. (siehe https://de.wikipedia.org/wiki/Gleitkommazahl#Eigenschaften_einer_Gleitkommaarithmetik)

Erstelle sinnvolle Konstruktoren.

Implementiere das nachfolgende Verhalten für einen Kassenbon-Eintrag:
- `getSum()` – Multipliziert Anzahl x Preis
- `toString()` – Ausgabe eines Eintrags

Receipt: Erstelle eine Klasse `Receipt` für den Kassenbon selbst.

Ein Kassenbon hat folgende Eigenschaften: Geschäft (`Shop`), Array mit den einzelnen Einträgen (`ReceiptItem`) - Größe des Arrays wird fix angegeben, Währung als Zeichenkette (zB EUR), Nachricht (zB „Vielen Dank für Ihren Einkauf“), Anzahl der bereits hinzugefügten Einträge

Programmieren und Software-Engineering II
Übung 11

Verwende folgende Konstanten für die Größe des Arrays und die Breite des Kassenbons:

```
public static final int MAX_LINELENGTH = 40;
public static final int MAX_NUMBER_OF_RECEIPTITEMS = 10;
```

Erstelle sinnvolle Konstruktoren.

Implementiere das nachfolgende Verhalten für einen Kassenbon:

- `addItem(ReceipeItem item)` – ein neuer Eintrag wird hinzugefügt. Befindet sich das Produkt bereits auf der Rechnung, so wird kein neuer Eintrag gemacht, sondern der vorhandene Eintrag aktualisiert.
  HINWEIS: Objekte der Klasse `String` immer mit der `equals()`-Methode vergleichen!
- `removeItem(ReceipeItem item)` – ein vorhandener Eintrag wird entfernt
  HINWEIS: Entstandene Lücken beim Entfernen eines Eintrags müssen wieder geschlossen werden!
- `calcSumVatTotal(int percent)` – berechnet die Summe der MWSt. für den gegebenen MWSt.-Satz
- `calcSumTotal()` – berechnet die Summe für den gesamten Einkauf
- `bill(int money)` – ein Geldbetrag wird übergeben und der Kassenbon wird erstellt.
  HINWEIS: An dieser Stelle ist es sinnvoll eine Methode `match(int money)` zu implementieren, die feststellt ob das gegebene Geld für den Einkauf ausreicht!

Teste das Programm mit folgender `main`-Methode:

```
Shop shop = new Shop("HTL Schulbuffet", "Machlandstraße", 48, 4320, "Perg", "07262/555    0815",
                        "schulbuffet@htl-perg.ac.at");

Receipt receipt = new Receipt(shop, "EUR", "Vielen Dank für Ihren Einkauf!");

receipt.addItem(new ReceiptItem("Cola 0,5l", 1, 120, 10));
receipt.addItem(new ReceiptItem("Cola 0,5l", 1, 120, 10));
receipt.addItem(new ReceiptItem("Pizzaeck", 1, 190, 10));
receipt.addItem(new ReceiptItem("M&Ms 200g", 3, 170, 10));

receipt.bill(1000);
```

Schreibe weitere Testfälle um den gesamten Funktionsumfang (auch Grenzfälle bzw. ungültige Angaben) abzudecken!

Da das Hauptaugenmerk der Übung im Umgang mit Klassen und Objekten liegt, kann die formatierte Ausgabe aus dem Dokument *„HÜ_PR2_11_CodeSnippets.txt“* entnommen und entsprechend an den eigenen Code angepasst werden.

Aufgabe 3: Tutorials
https://www.youtube.com/watch?v=fJkw0p2GgdM

Aufgabe 4: WH der bisherigen HUE-Beispiele