Praktikum

Bitte erstellen Sie für dieses Blatt ein IntelliJ-Modul „**Abgabe 10**“.

Zeichenketten

Froschrennen

Es soll ein Spiel „Froschrennen“ implementiert werden.

- Package: **frograce**
- Klassen: **Frog**, **Race**, **RaceTrack** und **FrogManager**

Teil 1) Ein Frosch hüpft

Die Klasse **Frog**

- Ein Frosch hat
  - Einen Namen.
  - Eine maximale Hüpfweite in cm, das ist die Entfernung, die ein Frosch bestenfalls hüpfen kann.
  - Eine Wegstrecke in cm (int currentDistanceInCm), so weit ist der Frosch im aktuellen Rennen schon gehüpft.
- Erstellung eines Frosches
  - Der Name wird als Parameter in den Konstruktor übergeben.
  - Die maximale Hüpfweite soll **50** cm betragen.
- Ein Frosch kann
  - Hüpfen: Der Frosch hüpft dabei eine zufällige Strecke zwischen **0** und seiner maximalen Hüpfweite. Diese Strecke wird auf seine bereits zurückgelegte Wegstrecke addiert.
  - Auskunft geben über seinen Namen und die im aktuellen Rennen bereits zurückgelegte Wegstrecke (Getter-Methoden).
  - Sich selbst mit der toString()-Methode als String-Darstellung zurückgeben (siehe Bild).

```
==============================================> Hugo
```

- Hinweise zur toString()-Methode
  - Verwenden Sie zur Zusammenstellung des Strings einen StringBuilder display.
  - Die zurückgelegte Wegstrecke wird durch „=“ ausgedrückt.
    - Genau ein „=“ pro zurückgelegte 10 cm.
    - Dies kann leicht mit einer for-Schleife realisiert werden.
    - **Beispiel:** Der Rennfrosch Hugo hat 30 cm zurückgelegt: `===> Hugo`

Die Klasse **FrogManager**

- Die Klasse besitzt eine main-Methode zum Erzeugen eines **FrogManager**.
- Testen Sie die Klasse **Frog** in einer manageFrogs()-Methode der Klasse **FrogManager**.
  - Erzeugen Sie einen Frosch Frog hugo.
  - Lassen Sie eine Schleife **20**-mal laufen.
    - Lassen Sie den Frosch in der Schleife hüpfen.
    - Geben Sie den Frosch auf der Kommandozeile aus System.out.println(hugo).
    - Fügen Sie eine Wartezeit von **200 Millisekunden** ein.
- Sobald der Frosch richtig hüpft, geben Sie den Frosch nicht mehr auf der Kommandozeile, sondern in **GameView** aus: gameView.print(hugo.toString(), 3);

Teil 2) Vier Frösche hüpfen auf einer Rennstrecke

Die Klasse **RaceTrack**

- Eine Rennstrecke hat
  - Einen Namen.
  - Eine Länge in cm (int lengthInCm).
  - Ein Array von Fröschen, die das Rennen bestreiten sollen.
- Erstellung einer Rennstrecke
  - Der Name und die Länge werden als Parameter an den Konstruktor übergeben.

- Eine Rennstrecke kann
  - Ein Frosch-Array für das nächste Rennen entgegennehmen (Setter-Methode).
  - Die Länge zurückgeben (Getter-Methode).
  - Sich selbst mit der toString()-Methode als String-Darstellung zurückgeben (siehe Bild).

```
Monte Carlo: 6 Zentimeter

 Start |-----------------------------------------------------------| Ziel

        ================================================> Hugo
        ==================================> Berta
        ========================================> Fatso
        ==================================================> Emma

 Start |-----------------------------------------------------------| Ziel
```

- Hinweise zur toString()-Methode
  - Verwenden Sie zur Zusammenstellung des Strings einen StringBuilder display.
  - Lagern Sie die Erstellung der Randlinien in eine Methode aus (pro 10 cm gibt es ein „-“): void createEdgeLine(StringBuilder display)
  - Rufen Sie für die Darstellung eines Frosches die toString()-Methode des Frosches auf.
  - Passen Sie die Ausgabe so an, dass Frösche auf der Startlinie zu hüpfen beginnen.
  - Fügen Sie zusätzliche Leerzeilen ein, damit die Rennstrecke nicht so gedrängt wirkt.

Erweitern Sie den Code in der Klasse **FrogManager**.

- Erzeugen Sie zusätzlich eine Rennstrecke.
- Legen Sie Ihren Frosch in ein Array.
- Fügen Sie das Array der Rennstrecke hinzu.
- In der Schleife:
  - Lassen Sie den Frosch hüpfen.
  - Geben Sie die Rennstrecke aus, nicht den Frosch.
  - Warten Sie 200 Millisekunden.
- Legen Sie zusätzlich noch drei weitere Frösche in das Array.
- Erzeugen Sie eine innere Schleife über das Frosch-Array und lassen Sie darin alle Frösche hüpfen.

Teil 3) Das Wettrennen

Die Klasse **Race**

- Ein Rennen hat
  - Eine Rennstrecke und ein Array von Fröschen, die das Rennen bestreiten sollen.
  - Eine Zahl winningFrogIndex, die **-1** ist, solange noch kein Frosch gewonnen hat, ansonsten zeigt sie auf den „Gewinner-Frosch“ im Array.
- Erstellung des Rennens
  - Die Rennstrecke und die Frösche werden als Parameter in den Konstruktor übergeben.
- Ein Rennen kann sich selbst als String darstellen. Dazu wird einfach die Rennstrecke dargestellt. Falls ein Frosch gewonnen hat, wird zusätzlich der Gewinner ausgegeben, z.B. „**Hugo hat gewonnen!**“
- Das Rennen kann durchgeführt werden void startRace().
  - Das Frosch-Array wird in die Rennstrecke gesetzt.
  - Es wird solange eine Methode void round() aufgerufen, bis winningFrogIndex nicht mehr **-1** ist.
- Durführen einer Runde in round()
  - Bitte übertragen Sie Teile des Codes aus dem **FrogManager** in diese Methode. Übertragen Sie auch die Benutzung von **GameView** in die Klasse **Race**.
    - Die Klasse **FrogManager** hat jetzt nur noch zwei Aufgaben
      - Die Erzeugung der Frösche und des Arrays.
      - Erzeugen und starten eines Rennens.
  - Eine Runde wird so durchgeführt
    - Alle Frösche hüpfen.
    - Wenn einer die Ziellinie erreicht hat, wird die Variable winningFrogIndex richtig gesetzt.
    - Die innere Schleife muss **sofort** abgebrochen werden, kein Frosch hüpft mehr.
    - Das Rennen wird mit GameView ausgegeben: gameView.print(this.toString(), 3);
    - Eine Wartezeit von 200 Millisekunden wird eingefügt.
- Fügen Sie zum Testen noch eine paketsichtbare Getter-Methode für winningFrogIndex ein.

BigDecimal

Annäherung der Kreiszahl π

Es gibt unterschiedliche Formeln, um ***π*** numerisch auf eine bestimmte Anzahl von Stellen anzunähern. In dieser Aufgabe soll die **Bailey-Borwein-Plouffe-Formel** angewandt werden, die seit dem Jahr 1995 bekannt ist.

- Package: **pi**
- Klassen: **Pi**

Je größer die Iterationszahl ***N*** ist, desto genauer wird die Zahl ***π*** bestimmt.

$$\pi = \sum_{n=0}^{N} \frac{1}{16^n}\left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\right)$$

Erstellen Sie eine Methode BigDecimal approximatePi(int precision, int iterations). Diese Methode soll die oben genannte Formel umsetzen und die angenäherte Kreiszahl ***π*** ausgeben.

- Der Parameter **precision** definiert, mit welcher Genauigkeit gerechnet werden soll, also die Anzahl der Vorkommastellen und Nachkommastellen.
- Mit dem Parameter **interations** wird die Anzahl an Iterationen ***N*** eingestellt.
- Nutzen Sie ein Objekt der Klasse **MathContext**, um die Genauigkeit zu setzen.
  - Setzen Sie die Rundungsmethode so, dass bei **0.5** aufgerundet wird.
  - Sie müssen den MathContext bei (fast) jeder Berechnung angeben.
- Nutzen Sie die Konstanten **ZERO**, **ONE**, … überall wo es möglich ist.
- Machen Sie die Methode paketsichtbar, damit Sie getestet werden kann.

Geben Sie in einer main-Methode das Ergebnis für **precision = 25** und **iterations = 100** aus.

```
3.1415926535897932384626433
```

Verständnisaufgaben

Zufallszahlen

Froschrennen mit Wiederholung in Zeitlupe

Das Froschrennen soll um eine Zeitlupenwiderholung erweitert werden.

Die Klasse **Frog**

- Fügen Sie eine neue Methode ein, welche die gehüpfte Distanz auf **0** zurücksetzt.
- Ändern Sie die Hüpfen-Methode so ab, dass ein Übergabeparameter vom Typ **Random** übergeben wird. Nutzen Sie dieses Objekt um die Hüpfweite zu würfeln.

Die Klasse **Race**

- Fügen Sie eine neue Instanzvariable vom Typ **Random** ein.
  - Sie wird im Konstruktor initialisiert.
  - In der Methode round() wird die Variable beim Hüpfen der Frösche übergeben.
  - In der Methode startRace() wird ein Seed für Random gesetzt: die aktuelle Systemzeit.

Wiederholung in Zeitlupe

Sie möchten gerne das jeweils letzte Rennen in Zeitlupe wiederholen können. Führen Sie dazu in der Klasse **Race** eine neue Methode void repeatLastRaceInSlowMotion() ein.

- Bei der Wiederholung verwenden Sie dann den selben Seed, wie beim letzten Rennen.
- Verlangsamen Sie die Darstellung: **500** Millisekunden, anstatt **200** Millisekunden.
- Vergessen Sie nicht, die Distanz der Frösche auf **0** und winningFrogIndex auf **-1** zurückzusetzen, bevor Sie die Zeitlupe starten.
- Vermeiden Sie unbedingt redundanten Code! Lagern Sie doppelten Code in eine Hilfsmethode aus.

Die Klasse **FrogManager** hatte in der Praktikumsaufgabe bereits ein Rennen gestartet. Warten Sie danach eine Sekunde und widerholen Sie dann das Rennen noch einmal in Zeitlupe.

Exceptions

Froschrennen mit Ausnahme

Jede Rennstrecke muss mindestens **50** cm lang sein, ansonsten wird im Konstruktor von **RaceTrack** eine **RaceLengthException** mit der Nachricht "A race track needs to have at least a length of 50 cm" ausgelöst.

- Die **RaceLenghtException** soll von der Klasse **Exception** abgeleitet werden.
- Die Ausnahme wird vollständig bis zur main-Methode durchgereicht.

Fügen Sie in der Klasse **FrogManager** am Anfang der Methode manageFrogs() folgenden Code ein. Damit können Sie den Benutzer nach der gewünschten Länge der Rennstrecke fragen.

```
String input = JOptionPane.showInputDialog("Bitte Länge der Rennstrecke in cm eingeben: ");
int lengthByUser = Integer.parseInt(input);
```

Im **FrogManager** wird bereits eine Rennstrecke „Monte Carlo“ erzeugt.

- Ändern Sie die Erzeugung der Rennstrecke so ab, dass lengthByUser benutzt wird.
- Umschließen Sie die Erzeugung der Rennstrecke mit einer try-catch-Behandlung.
- Erzeugen Sie im Catch-Block eine Strecke mit **600** cm, falls der User eine zu kurze Strecke wählt.