Beziehungen zwischen Klassen und Objekten

1. Schneckenrennen

1 Rennschnecke

1. Erstellen Sie eine Klasse *Rennschnecke*.
2. Die Rennschnecke soll folgende Eigenschaften besitzen:
   - einen Namen
   - eine Rasse
   - eine Maximalgeschwindigkeit (Veränderung der Strecke pro Schritt)
   - die Schnecke soll wissen welchen Weg sie bereits zurück gelegt hat

1. Erstellen Sie für die Klasse *Rennschnecke* einen Konstruktor, der den Eigenschaften beim Erstellen einer neuen Instanz (Objekt zu dieser Klasse) die Werte über dessen Parameter zuweist.
2. Legen Sie in der Klasse *Rennschnecke* eine Methode *void krieche()* an, welche die Schnecke abhängig von ihrer Maximalgeschwindigkeit eine zufällige Strecke weiter bewegt. Soll heissen: Sie kriecht eine zufällige Strecke grösser null und kleiner ihrer Maximalgeschwindigkeit. Nehmen Sie als Zeiteinheit 1 an.

   Tipp: Mit der folgenden Methode können Sie einen Zufallswert generieren:

   ```
   public float getRandom(float min, float max){
         return min + (float)(Math.random() * ((max - min) + 1));
   }
   ```

3. Legen Sie in der Klasse *Rennschnecke* eine Methode *String getData()* an, welche die Daten der Schnecke als String zurückgibt.
4. Testen Sie Ihre Klasse, indem Sie probehalber ein Rennschneckenobjekt in einer Hauptklasse (Klasse mit Main-Methode) erzeugen und seine Daten auf der Konsole ausgeben.

   Tipp: Verwenden Sie zum Ausgeben der Daten die *getData()* Methode der Rennschnecke.

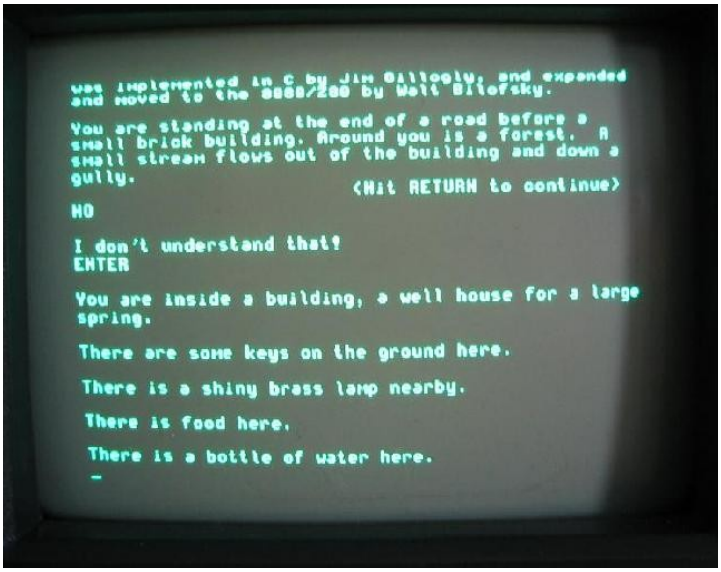
2 Rennen

1. Erstellen Sie eine Klasse *Rennen* welche von der **vorgegeben** Klasse *Rennstruktur* erbt.
2. Ein Rennen soll folgende Eigenschaften haben:
   - einen Namen
   - die Anzahl der Teilnehmenden Schnecken
   - die teilnehmenden Schnecken selbst als Array.
   - Die Länge der zu kriechenden Strecke
3. Überlegen Sie sich, welche dieser Werte im Konstruktor schon gesetzt werden können.
4. Legen Sie in der Klasse *Rennen* eine Methode *void addRennschnecke(Rennschnecke neue_rennschnecke)* an, welche eine Schnecken dem Rennen hinzufügt.
5. Legen Sie in der Klasse *Rennen* eine Methode *void removeRennschnecke(String name)* an, welche eine Schnecke aus dem Rennen entfernt. (Falls dieser Schritt zu schwer ist, kann dieser vorläufig auch übersprungen werden).
6. Legen Sie in der Klasse *Rennen* eine Metohde *String getData()* an, welche die Daten des Rennens als String zurück gibt. Dabei können Sie alle Daten der teilnehmenden Rennschnecken ausgeben (deren *getData()* Methode).
7. Testen Sie Ihre Klasse vom Hauptprogramm aus.
8. Legen Sie in der Klasse *Rennen* eine Methode *Rennschnecke ermittleGewinner()* an, welche *null* zurück liefert, wenn noch keine der teilnehmenden Schnecken das Ziel erreicht hat und anderenfalls die Gewinnerschnecke zurück gibt (*null* ist ein Schlüsselwort in Java, und steht in diesem Beispiel für ein nicht-instanziiertes Objekt. Sollte eine Objektvariable ein *null* beinhalten, können Sie auch entsprechend dies überprüfen: *mein_object == null*).
9. Legen Sie in der Klasse *Rennen* eine Methode *void lasseSchneckenKriechen()* an, welche alle teilnehmenden Schnecken einmal kriechen lässt.
10. Überschreiben Sie in der Klasse *Rennen* die Methode *boolean durchfuehren()*. Diese soll bei jedem Aufruf *lasseSchneckenKriechen()* ausführen und zugleich überprüfen ob eine Schnecke das Ziel erreicht hat. Wenn keine Schnecke nach diesem Schritt das Ziel erreicht hat, soll diese Methode ein *false* zurück geben, ansonsten ein *true*.

    Tipp: Ob eine Schnecke im Ziel angekommen ist, können Sie mit Ihrer Methode *ermittleGewinner()* herausfinden.

11. Instantiieren Sie die **vorgegebene** Klasse *RennenHandler* in Ihrer Hauptklasse. Mit der Methode *setRennen(Rennen)* im RennHandler können Sie ein Rennen setzten. Mit der Methode *start()* im *RennHandler*, wird jede Sekunde Ihre überschriebene Methode *durchfuehren()* solange ausgeführt, bis diese *true* zurück gibt (Schnecke hat das Ziel erreicht).

2. Text Adventure (Verkettung) (Knacknuss)

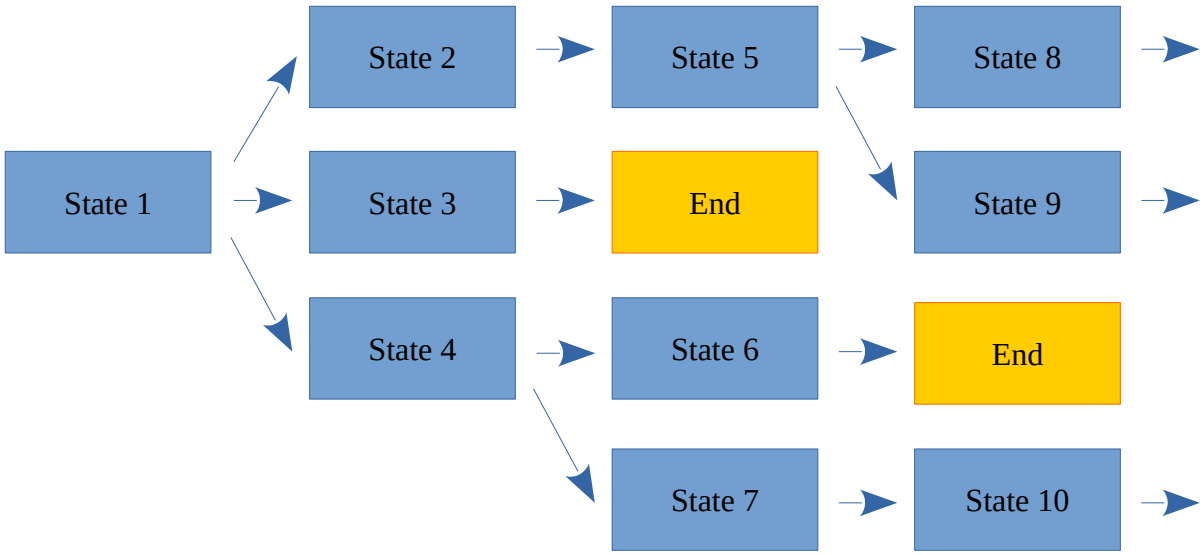


Text-Adventure-Spiele sind Spiele die grundlegend in der Textform präsentiert werden. Dabei wird dem Spieler eine Situation beschrieben und ihm durch die Auswahl von Optionen eine Entscheidung überlassen. Z.B. „Sie befinden sich vor zwei Türen“ Auswahl: „1) Tür 1 betreten“ „2) Tür 2 betreten“. Nach einer Wahl der Optionen wird dem Spieler eine neue Situation vorgestellt. Durch den geringen technischen Aufwand sind solche Spiele schon seit 1978 bekannt, und bis heute in Kombination mit Grafiken weiterhin vertrieben. Unter folgendem Link finden Sie eine Browser-Variante eines Text-Adventure Spiel: https://writer.inklestudios.com/stories/mbv8

Mit dem Wissen der Programmstrukturen würden Sie diese Spielvariante wahrscheinlich durch verschachteln von if-Abfragen programmieren. Diese Variante wird jedoch sehr Aufwendig, wenn das Spiel eine gewissen tiefe Aufweist:

```
...
System.out.println(„Sie befinden sich vor zwei Türen“);
...
if(player_selection == 1){
      System.out.println(„Sie befinden sich vor einer Truhe“);
      ...
      if(player_selection == 1){
            System.out.println(„In der Truhe befinden sich ein Pergament und
            einen Schlüssel“);
            if(player_selection == 2){
                  ...
            } else (player_selection == 2) ...
      } else if(player_selection == 2){
            ...
      }
} else if(player_selection == 2){
      ...
}
...
```

In diesem Beispiel (mit einer Entscheidungstiefe von 3) wird die Struktur relativ schnell unübersichtlich und Komplex. Dank der Objektorientierten Programmierung haben Sie jedoch die Möglichkeit eine Struktur aufzubauen, die eine solche Implementierung vereinfacht. Dabei soll folgende Idee verfolgt werden:



Diese „States“ könnten jeweils ein Objekt sein welches den Vater-State und die Kinder-States beinhaltet (Mit Vater und Kind ist hier **kein** Veerbungsverhältnis gemeint sondern die Referenzen zu den Objekten). Das bedeutet, dass ein State-Objekt z.B. die Referenzen zu dem Vater-State und 3 Kinderstates abspeichern kann. So ein State kann zudem den Adventure-Text beinhalten. Am Schluss kann beim Ausführen der erste State ausgeführt werden und je nach Entscheidung das entsprechende Kind-State. Dieser Prozess kann dann so weiter geführt werden.

Versuchen Sie diese Struktur zu implementieren und erstellen Sie damit ein kleines Text-Adventure-Spiel.