Übung 7: Objektorientierung

1. Aufgabe: Konstruktoren, Objektmethoden

Punkte: 10

Lernziel: In dieser Aufgabe sollen Sie den Umgang mit *einfachen Klassen*, *Konstruktoren* und *Objektmethoden* üben.

1.1. Klasse `Farm`

Erstellen Sie eine Klasse `Farm`, die einen Bauernhof darstellt.

Die Klasse soll Felder für den Besitzer des Bauernhofes (`String owner`) sowie für die Stadt in der der Bauernhof steht (`String city`) definieren.

Erstellen Sie einen Konstruktor, der ein neues `Farm`-Objekt mit einem gegebenen Besitzernamen und einer gegebenen Stadt initialisiert.

Implementieren Sie eine Methode `String toString()`, welche die Eigenschaften des Bauernhofs wie folgt als Text zurückgibt:

```
(Beispiel 1)
Farm in Linz owned by Andreas Schorfenhummer.

(Beispiel 2)
Farm in St. Pölten owned by Franziska Flink.
```

1.2. Klasse `Stable`

Erstellen Sie eine Klasse `Stable`, die einen Stall auf einem Bauernhof darstellt.

Die Klasse soll Felder für die Stallnummer (`int nr`) und den Bauernhof (`Farm farm`) definieren.

Erstellen Sie einen Konstruktor, der ein neues `Stable`-Objekt mit einer gegebenen Stallnummer und einem gegebenen Bauernhof initialisiert. *Hinweis: Sie können davon ausgehen, dass die übergebene `Farm` nie `null` ist.*

Implementieren Sie eine Methode `String toString()`, welche Informationen über den Stall wie folgt als Text zurückgibt:

```
(Beispiel 1)
Stable #1 of
Farm in Linz owned by Andreas Schorfenhummer.

(Beispiel 2)
Stable #1 of
Farm in St. Pölten owned by Franziska Flink.

(Beispiel 3)
Stable #2 of
Farm in St. Pölten owned by Franziska Flink.
```

Nutzen Sie hier für die zweite Zeile des Textes die `toString`-Methode des Bauernhofs.

1.3. Klasse `Animal`

Erstellen Sie eine Klasse `Animal`, die ein Tier darstellt.

Die Klasse soll Felder für den Namen des Tieres (`String name`), dessen Stall (`Stable stable`) und seine tägliche Futtermenge in kg (`double foodPerDay`) definieren.

Erstellen Sie in der Klasse `Animal` drei Konstruktoren:

- `Animal(String name, Stable stable, double foodPerDay)` soll alle Felder mit den gegebenen Werten initialisieren.
- `Animal(String name, Stable stable)` soll den ersten Konstruktor mit den gegebenen Parametern aufrufen und für die tägliche Futtermenge den fixen Wert 1.0 übergeben.
- `Animal(String name)` soll den zweiten Konstruktor mit dem gegebenen Namen aufrufen und als Stall `null` übergeben.

Implementieren Sie eine Methode `boolean changeFoodRation(double delta)`. Wenn die neue tägliche Futtermenge (*bisherige Futtermenge + delta*) positiv ist, soll die tägliche Futtermenge des Tieres angepasst und `true` zurückgegeben werden. Wenn die neue tägliche Futtermenge jedoch ≤ 0 ist, soll die tägliche Futtermenge des Tieres *nicht* angepasst und `false` zurückgegeben werden.

Implementieren Sie eine Methode `void print()`, welche die Daten des Tieres ausgibt (Name, Stall, Futtermenge). Benutzen Sie zur Ausgabe des Stalls die zuvor implementierte Methode `toString()` der `Stable`-Klasse. Achten Sie darauf, dass der Stall potentiell `null` sein kann.

Beispielformat:

```
(Beispiel 1, Stall ist nicht null)
Animal Susi
Stable #2 of
Farm in St. Pölten owned by Franziska Flink.
Daily food: 15.80kg

(Beispiel 2, Stall ist null)
Animal Bello
no stable
Daily food: 1.00kg
```

1.4. Test

Erstellen Sie eine Klasse `FarmApplication`. Implementieren Sie darin eine `main`-Methode und testen Sie Ihre Klassen darin folgendermaßen:

- Legen Sie zumindest **ein** `Farm`-Objekt an.
- Legen Sie zumindest **zwei** `Stable`-Objekte an.
- Legen Sie zumindest **drei** `Animal`-Objekte an. Beim Anlegen dieser Objekte soll jeder verfügbare Konstruktor zumindest einmal verwendet werden. Eines der Animal-Objekte (jenes, welches über den Konstruktor `Animal(String name)` angelegt wird) ist also keinem Stall zugeordnet.
- Rufen Sie die `print`-Methode eines jeden Tieres auf.
- Ändern Sie die Futtermenge (`changeFoodRation`) eines Tieres mit einem gültigen Wert und geben Sie dieses Tier erneut mit `print` aus. Prüfen Sie den Rückgabewert von `changeFoodRation` und geben Sie eine entsprechende Meldung aus.

- Ändern Sie die Futtermenge (`changeFoodRation`) eines Tieres mit einem ungültigen Wert (so dass sich eine negative Futtermenge ergibt) und geben Sie dieses Tier erneut mit `print` aus. Prüfen Sie den Rückgabewert von `changeFoodRation` und geben Sie eine entsprechende Meldung aus.

Beispielausgabe:

```
Animal Susi
Stable #1
Farm in Linz owned by Andreas Schorfenhummer
Food per day: 15,80

Animal Chicky
Stable #2
Farm in Linz owned by Andreas Schorfenhummer
Food per day: 1,00

Animal Bello
no stable
Food per day: 1,00

Food ration change by 20.0 for Susi successful      // Anmerkung: Ausgabe in main() basierend auf Rückgabewert
Animal Susi
Stable #1
Farm in Linz owned by Andreas Schorfenhummer
Food per day: 35,80

Food ration change by -10.0 for Bello unsuccessful  // Anmerkung: Ausgabe in main() basierend auf Rückgabewert
Animal Bello
no stable
Food per day: 1,00
```

2. Aufgabe: Vererbung, Dynamische Bindung

Punkte: 14

Lernziel: In dieser Aufgabe sollen Sie den Umgang mit *Vererbung* und *dynamischer Bindung* üben.

Implementieren Sie die Klassenhierarchie aus Abbildung 1. Diese Klassenhierarchie bildet eine einfache Aufgabenverwaltung ab, in der Sie drei Klassen realisieren müssen.

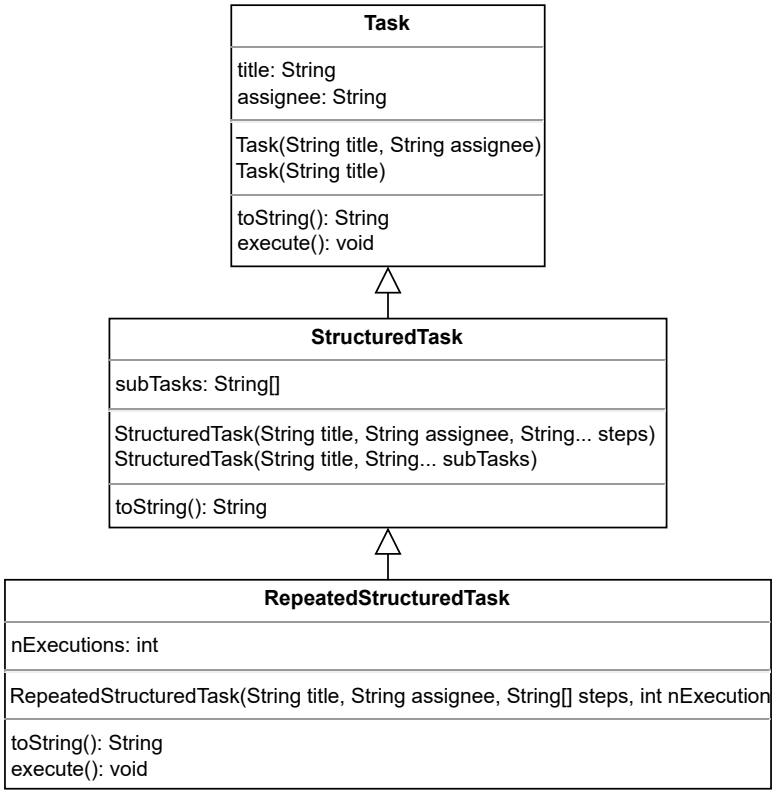

Abbildung 1: Klassendiagramm.

2.1. Klasse `Task`

Ein `Task` stellt eine Aufgabe dar und besteht aus folgenden Komponenten:

- Ein Feld `title` für den Aufgabennamen.
- Ein Feld `assignee` für die Person, die die Aufgabe ausführen soll.
- Ein Konstruktor `Task(String title, String assignee)`, der alle Felder mit den gleichnamigen Parametern initialisiert. *Hinweis: Sie können annehmen, dass sowohl `title` als auch `assignee` mit gültigen Werten übergeben werden und müssen daher keine Fehlerüberprüfung durchführen.*
- Ein Konstruktor `Task(String title)`, der den zuvor erstellten Konstruktor mit dem gegebenen Titel und mit dem String "No assignee" als `assignee` aufruft.

- Eine Methode `String toString()`, die den Aufgabennamen sowie die Person als `String` zurückgibt.

  Beispielformat:

  ```
  (Beispiel 1)
  Do laundry (Markus)

  (Beispiel 2)
  Wash the dishes (Birgit)

  (Beispiel 3)
  Water the plants (No assignee)
  ```

- Eine Methode `void execute()`, welche die Aufgabe ausführt. Die Ausführung soll durch die Ausgabe des Textes *Executing task:* gefolgt vom `toString()-String` simuliert werden. *Beispielausgabe:*

  ```
  Executing task: Wash the dishes (Birgit)
  ```

2.2. Klasse `StructuredTask`

`DetailedTask` erbt von `Task` und stellt eine Aufgabe dar, die aus mehreren Teilaufgaben besteht. Definieren Sie dazu folgende Komponenten:

- Ein Feld `String[] steps`, das die Teilaufgaben beschreibt.
- Ein Konstruktor `DetailedTask(String title, String assignee, String... steps)`, der den Superkonstruktor mit dem gegebenen Aufgabentitel sowie der gegebenen Person aufruft. Das Feld `steps` wird auf den gleichnamigen Parameter gesetzt. *Hinweis: Sie können annehmen, dass `steps` ein nicht-leeres Array ist.*
- Ein Konstruktor `DetailedTask(String title, String... steps)`, der den Superkonstruktor nur mit dem Titel aufruft. Das Feld `steps` wird auf den gleichnamigen Parameter gesetzt.
- `toString()` soll überschrieben werden. Dabei soll ein `String` zurückgegeben werden, der das Resultat des `super`-Aufrufs enthält, gefolgt von allen `steps`.

  Beispielformat:

  ```
  (Beispiel 1)
  Do homework (Gudrun) [Work on first assignment, Work on second assignment, Test implementations]

  (Beispiel 2)
  Change car tires (No assignee) [Buy new tires, Remove old tires, Put on new tires, Store or sell old tires]
  ```

Klasse `RepeatedStructuredTask`

`RepeatedStructuredTask` erbt von `StructuredTask` und stellt eine Aufgabe dar, die aus Teilaufgaben besteht und mehrfach ausgeführt werden soll. Definieren Sie dazu folgende Komponenten:

- Ein Feld `int nExecutions`, das angibt, wie oft die Aufgabe wiederholt werden soll.
- Ein Konstruktor `RepeatedStructuredTask(String title, String assignee, String[] steps, int nExecutions)`, der den Superkonstruktor mit `title`, `assignee` und `steps` aufruft und anschließend das Feld `nExecutions` mit dem gleichnamigen Parameter initialisiert.
- `toString()` soll überschrieben werden. Der resultierende `String` soll das Resultat des `super`-Aufrufs, gefolgt von der Anzahl der Wiederholungen (`nExecutions`), enthalten.

  Beispielformat:

  ```
  (Beispiel 1)
  Prepare for SW1 exam (No assignee) [Study slides, Study assignments, Study self-assessments, Study last year's exams] x3

  (Beispiel 2)
  Prepare dinner table (Hans) [Put plate on table, Put glass on table, Put cutlery on table] x6
  ```

- `execute()` soll überschrieben werden. Rufen Sie darin in einer Schleife `nExecutions` mal `super.execute()` auf.

2.3. Klasse `TaskManager`

Implementieren Sie eine Klasse `TaskManager`, mit der verschiedene Aufgaben verwaltet werden können. Definieren Sie darin folgende Komponenten:

- Eine globale Konstante `static final int MAX_TASK_CAPACITY`, welche die maximale Anzahl an speicherbaren Aufgaben darstellt. Initialisieren Sie diese Konstante mit dem Wert 15.
- Ein Feld `Task[] tasks`, das alle gespeicherten Aufgaben enthält.
- Ein Feld `int nTasks`, das die letzte freie Position im `tasks`-Array angibt und beim Einfügen um 1 erhöht wird.
- Ein Konstruktor `TaskManager()`, der ein neues `Task`-Array mit der Länge `MAX_CAPACITY` anlegt und in `tasks` speichert. `nTasks` soll mit 0 initialisiert werden.
- Eine Methode `boolean addTask(Task task)`, die eine neue Aufgabe `task` hinzufügt. Wenn im `tasks`-Array noch Platz frei ist (`nTasks < tasks.length`), fügen Sie `task` an Position `nTasks` im Array ein. Erhöhen Sie dann `nTasks` um 1 und geben Sie `true` zurück. Wenn kein Platz mehr frei ist, soll das Array unverändert bleiben und `false` zurückgegeben werden.
- Eine Methode `boolean executeTask(int taskNr)`, die die Aufgabe an der Stelle `taskNr` im Array ausführt (`execute()`). Geben Sie `false` zurück, wenn `taskNr` ungültig ist. Andernfalls geben Sie `true` zurück.
- Eine Methode `void print()`, die die Anzahl der im `tasks`-Array gespeicherten Aufgaben sowie die Aufgaben selbst ausgibt. (benutzen Sie dafür die `toString()`-Methode aus `Task`)

  Beispielausgabe (Aufruf von `print()`):

  ```
  2 tasks registered:
  Wash the dishes (Flora)
  Clean the living room (Simon)
  ```

Die vorgegebene Klasse `TaskManagerApplication` ermöglicht die Bedienung eines `TaskManager`s über ein Konsolenmenü. Rufen Sie die darin enthaltene `main`-Methode auf und testen Sie damit Ihre Implementierung. Erstellen Sie mithilfe des Konsolenmenüs mindestens einen Task je Typ (`Task`, `StructuredTask`, `RepeatedStructuredTask`) und führen Sie alle im Menü gegebenen Kommandos aus. Geben Sie die dabei entstandenen Ausgaben ab.

Beispielablauf:

```
Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: s
0 tasks registered:

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: e
Task to execute: 0
Could not execute task! Invalid task number 0

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: c
Creating new task:
Type ([1] default, [2] structured, [3] repeated): 1
Title: Wash the dishes
Assignee: Hubert
Successfully registered task!

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: c
Creating new task:
```

```
Type ([1] default, [2] structured, [3] repeated): 1
Title: Clean the living room
Assignee: Gerald
Successfully registered task!

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: c
Creating new task:
Type ([1] default, [2] structured, [3] repeated): 2
Title: Do homework
Assignee: Simone
# of Steps: 3
Steps 1: Work on assignment 1
Steps 2: Work on assignment 2
Steps 3: Test implementations
Successfully registered task!

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: c
Creating new task:
Type ([1] default, [2] structured, [3] repeated): 3
Title: Prepare for SW1 exam
Assignee: Simone
# of Steps: 4
Step 1: Study slides
Step 2: Study assignments
Step 3: Study self-assessments
Step 4: Study last year's exams
# of executions: 3
Successfully registered task!

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: s
4 tasks registered:
Wash the dishes (Hubert)
Clean the living room (Gerald)
Do homework (Simone) [Work on assignment 1, Work on assignment 2, Test implementations]
Prepare for SW1 exam (Simone) [Study slides, Study assignments, Study self-assessments, Study

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: e
Task to execute: 0
Executing task: Wash the dishes (Hubert)
Task 0 successfully executed

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: e
Task to execute: 2
Executing task: Do homework (Simone) [Work on assignment 1, Work on assignment 2, Test impleme
Task 2 successfully executed

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: e
Task to execute: 3
Executing task: Prepare for SW1 exam (Simone) [Study slides, Study assignments, Study self-ass
Executing task: Prepare for SW1 exam (Simone) [Study slides, Study assignments, Study self-ass
Executing task: Prepare for SW1 exam (Simone) [Study slides, Study assignments, Study self-ass
```

```
Task 3 successfully executed

Command ([c] create task, [e] execute task, [s] print summary, [q] exit]: q

Task manager shutdown
```

Zusätzliche Anweisungen

- Die `toString`-Methoden müssen den Zugriffsspezifizierer `public` verwenden. Die Hintergründe dieser Einschränkung werden wir in der nächsten Stunde genauer besprechen.
- Verwenden Sie zur Implementierung der Ein- und Ausgabe für alle Programme nur die Funktionen der beiden bereitgestellten Klassen `In` und `Out`, wie in der Übung gezeigt. Die entsprechenden Java-Dateien samt HTML-Dokumentation sind in Moodle unter *InOut.zip* zu finden.
- Implementieren Sie formatierte Ausgaben, indem Sie `String.format` in Kombination mit `Out.print` und `Out.println` verwenden, wie in der Übung gezeigt. Verwenden Sie als Referenz den Foliensatz `StringFormat.pdf` im Moodle.
- Sie dürfen, falls nötig, `Math`-Funktionen der Java-Bibliothek nutzen.
- Verwenden Sie für Variablen passende Datentypen Ihrer Wahl.
- Alle Namen (Klassennamen, Variablennamen, Methodennamen, etc.) sind in Englisch zu wählen.
- Formatierung, Namenswahl, etc. fließen in die Bewertung mit ein.
- Vermeiden Sie Codeduplikation und unnötige Operationen (beispielsweise zu viele oder unnötigen Verzweigungen).

Abzugeben

Geben Sie *eine .zip Datei* mit dem Namen kxxxxxxxx_UExx.zip ab (Beispiel: k01234567_UE07.zip). Darin muss enthalten sein:

- Der *Source Code* aller Aufgaben der aktuellen Übung (alle `.java` Dateien, keine `.class` Dateien).
- Zu jeder Aufgabe ist ein Testprotokoll in Form von mehreren Testausgaben anzufertigen. Testen Sie nicht nur einen Fall, sondern mehrere allgemeine Fälle sowie typische Grenzfälle (negative Zahlen, null, etc.). Die Testprotokolle dürfen entweder gesamt in einer Datei zusammengefasst werden (`Testprotokoll.txt` oder `Testprotokoll.pdf`), oder in eine Datei pro Aufgabe aufgeteilt werden (`Testprotokoll_BspXX.txt` oder `Testprotokoll_BspXX.pdf`).