Programmieren II: Java

2. Praktikum (Abgabe 17. Mai, 24 Uhr)

Auer, Hanel, Jürgensen, Lehner, Riemenschneider

Aufgabe 1: Game of Thrones Charaktere

Aufgabe 2: Einheiten

Inhalt

Hinweise zum Praktikum

Hinweise zum Praktikum

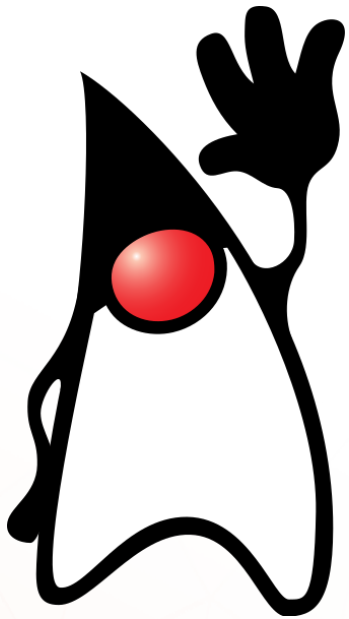
- Abgabetermin dieses Praktikums: **Abgabe 17. Mai, 24 Uhr**
- Sie dürfen die Aufgaben alleine oder zu zweit abgeben
- Kennzeichnen Sie Ihre Abgabe entsprechend mit Ihren Namen
- Sie müssen 4 der 5 Praktika bestehen
- Kommentieren Sie Ihren Code
  - Jede Methode (wenn nicht vorgegeben)
  - Wichtige Anweisungen/Code-Blöcke
  - Nicht kommentierter Code führt zu Nichtbestehen
- Bestehen Sie eine Abgabe nicht haben Sie einen zweiten Versuch, in dem Sie Ihre Abgabe verbessern müssen
- Wichtig
  - Sie sind einer Praktikumsgruppe zugewiesen, nur in dieser werden Ihre Abgaben akzeptiert
  - Beachten Sie dazu die Hinweise auf der Moodle-Kursseite

Inhalt

Willkommen zum 2. Praktikum!

Lernziele

- Implementieren nach einer Spezifikation
- Aufbau von Klassen: Attribute und Methoden
- Konstruktoren
- Getter und Setter
- Java-Standard-Methoden: `equals`, `hashCode`, `toString`
- Dokumentation: `javadoc`
- Testen mit JUnit
- **`enums`**: definieren, erweitern und verwenden



Inhalt

Inhalt

Aufgabe 1: Game of Thrones Charaktere

Aufgabendefinition

Aufgabendefinition

- Schwierigkeitsgrad: mittel
- Wir modellieren Charaktere aus „Game of Thrones“ in Java
- `https://usefulcharts.com/blogs/charts/game-of-thrones-family-tree` (**Spoileralert!**)
- Die Aufgabe ist inspiriert durch das Programmierpraktikum SS19 von Prof. Schiedermeier

Die Klasse Character

- Die Character-Klasse modelliert einen Character aus der Serie

Character
– name : String – house : House – mother : Character – father : Character – alive : **boolean** – fightSkills : **int**
+ Character(Character other) + Character(String name, fightSkills : **int**, house : House) + Character(n : String, m : Character, f : Character) + fight(other : Character) + toString() + equals(other : Object) + hashcode()

- Setter und Getter sind aus Platzgründen nicht aufgeführt (s. unten)

Die Klasse Character

- Attribute
  - `name`: Name des Characters (nur getter, unveränderbar, darf nicht **`null`** und leer sein)
  - `house`: Haus des Characters (s. unten, nur getter, unveränderbar, darf **`null`** sein)
  - `mother`: Mutter (nur getter, unveränderbar, darf **`null`** sein)
  - `father`: Vater (nur getter, unveränderbar, darf **`null`** sein)
  - `alive` wenn **`false`** dann tot, sonst am Leben (getter, setter)
  - `fightSkills`: Kampffähigkeiten, **`int`**-Wert von 0 bis 100 mit 0 für „völlig wehrlos“ und 100 für „perfekter Kämpfer“
- Methoden
  - Konstruktor `Character(other : Character)`: kopiert den anderen Character (flach)
  - Konstruktor `Character(name : String, house : House, alive : boolean, fightSkills : int)`: erstellt Character mit Namen, Haus, Lebend-status und Kampffähigkeiten

Die Klasse Character

- Methoden
  - „Geburts“-Konstruktor
    `Character(n : String, m : Character, f : Character)`:
    erstellt Kind nach folgenden Regeln
    - `f` ist Vater, `m` ist Mutter (keiner darf **`null`** sein und beide müssen leben)
    - Vater und Mutter dürfen nicht gleich sein
    - Das Kind ist lebendig
    - Die Kampfähigkeiten sind der Mittelwert der Kampfähigkeiten der Eltern (gerundet)
    - Mutter vererbt Haus wenn dies nicht **`null`** ist, sonst der Vater
  - `fight`: Kämpft gegen anderen Character mit:
    - Es dürfen nur lebendige Character kämpfen
    - Ein Character darf nicht gegen sich selbst kämpfen
    - Es überlebt nur der mit dem größeren Kampfähigkeiten
    - Bei Gleichstand überleben beide
  - `toString`: gibt String-Repräsentation zurück, bspw.

```
{ name = "Joffrey", alive = true, father = "Jamie", ↩
    mother = "Cersei", house = "Lannister (Casterly ↩
    Rock)", fightSkills = 32 }
```

Die Klasse House

- Die House-Klasse modelliert ein Haus (Familie) in Game of Thrones

House
– name : String – seat : String
+ House(House other) + House(name : String, seat : String) + toString(): String + equals(other : Object): **boolean** + hashCode(): **int**

- Attribute: Name und Sitz der Familie, z.B. Lannister mit Sitz Casterly Rock (nur getter, unveränderlich, dürfen nicht **null** und leer sein)
- Kopier-Konstruktor und Version mit Name und Sitz
- `toString` gibt String-Repräsentation zurück im Format

```
{ name = "Lannister", seat = "Casterly Rock" }
```

Inhalt

Aufgaben

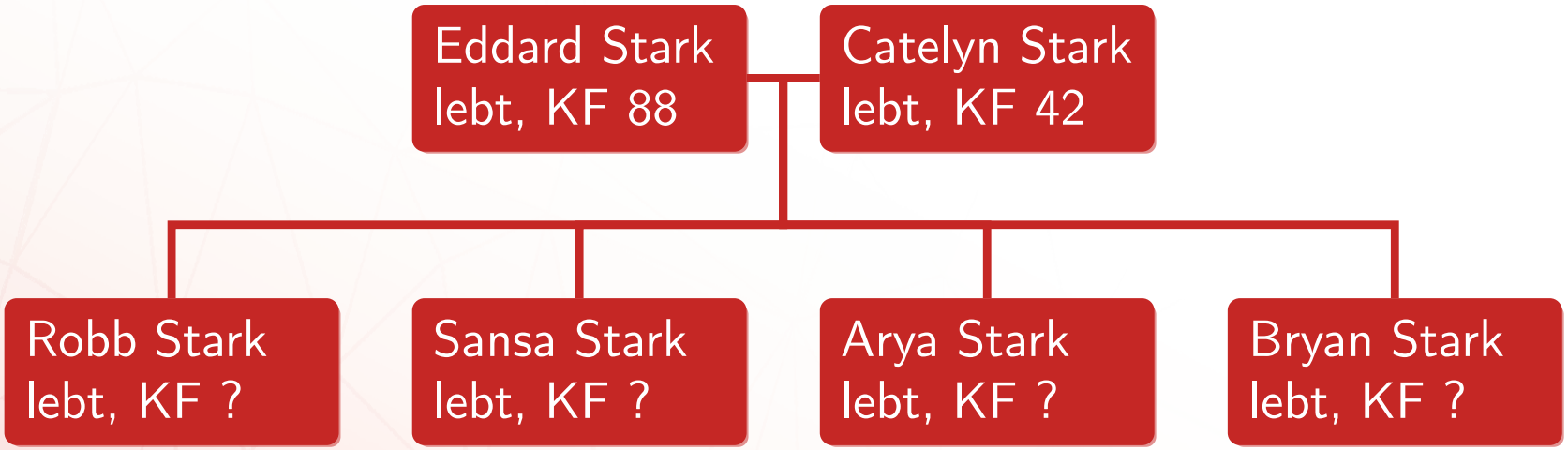
- Implementieren Sie die Klassen `House` und `Character`
- Achten Sie dabei auf Folgendes:
  - Benutzen Sie **`final`** wo überall möglich
  - Prüfen Sie Eingabeparameter auf Gültigkeit
  - Sollte ein Eingabeparameter ungültig sein, generieren Sie eine Ausnahme mit

```
throw new IllegalArgumentException("Parameter xyz ↩
    darf nicht null sein.");
```

  - Implementieren Sie selbstständig die Methode `equals` für beide Klassen
  - `hashCode` können Sie durch die IDE generieren lassen
  - Dokumentieren Sie Ihre Implementierungen mit `javadoc`-Kommentaren
- Nutzen Sie zum Testen Ihrer Implementierung die gegebenen JUnit-Tests
  - got/HouseTest.java got/CharacterTest.java
  - Sie können die JUnit-Tests von Ihrer IDE aus ausführen

Aufgaben

- Schreiben Sie eine Java-Hauptprogramm, das folgenden (unvollständigen) Stammbaum des Hauses Stark (Sitz „Winterfell“) erzeugt



- Bilden Sie folgende Sachverhalte in Ihrem Programm ab (**Spoileralert!**)
  - Eddard Stark stirbt in der ersten Staffel der Serie
  - Raymund Frey (lebendig, KF 68 Haus „Frey“, Sitz „The Twins“) kämpft gegen Catelyn Stark
- Geben Sie am Ende Ihres Programms alle Häuser und Charaktere auf der Konsole aus

Inhalt

Inhalt

Aufgabe 2: Einheiten

Längeneinheiten als enum

Längeneinheiten

- Schwierigkeitsgrad: mittel
- Längen können in verschiedenen Einheiten angegeben werden

Symbol	Name	in Metern	System
mm	Millimeter	$1 \cdot 10^{-3}$m	SI
cm	Centimeter	$1 \cdot 10^{-2}$m	SI
m	Meter	1m	SI
km	Kilometer	$1 \cdot 10^{3}$m	SI
au	Astronomical Unit	$0.149 \cdot 10^{12}$m	SI
in	Inch	$25.4 \cdot 10^{-3}$m	imperial
ft	Foot	0.305m	imperial
yd	Yard	0.914m	imperial

(SI = „système international“, International System of Units)

Aufgabenstellung

- Definieren ein **enum** LengthUnit für die Längeneinheiten wie oben angegeben
  - Werte der Enumeration:

```
MILLIMETER, CENTIMETER, METER, KILOMETER,
ASTRONOMICAL_UNIT, INCH, FOOT, YARD
```

  - Methoden
    - **double** asMeters() gibt die Länge in Metern zurück
    - String getSymbol() liefert das Symbol zurück
    - **boolean** isSI() gibt **true** wenn es sich um eine SI-Einheit handelt, sonst **false**
- Verwenden Sie zum Testen die JUnit-Tests

length/LengthUnitTest.java

Inhalt

Aufgabe 2: Einheiten

Die Klasse Length

- Betrachten Sie folgende Klasse zum Rechnen mit Längen

Length
− value : **double** − unit : LengthUnit
+ Length(value : **double**, unit : LengthUnit) + getValue(): **double** + getUnit(): LengthUnit + as(unit : LengthUnit): Length + add(other : Length): Length + toString(): String

- Attribute
  - value: Wert der Länge
  - unit: Einheit der Länge

Aufgabenstellung

- Implementieren Sie die Klasse `Length`
  - Die Klasse ist unveränderlich
  - Konstruktor
    `Distance(value : double, unit : DistanceUnit)`
    initialisert das Objekt mit den Parametern
  - Getter: `getValue` und `getUnit` geben die entsprechenden Attribute zurück
  - as(unit : LengthUnit) gibt die ein neues Length-Objekt mit der gleichen Länge, aber in der angegebenen Einheit zurück
  - add(other : Length) addiert zur Länge die Länge `other` hinzu. Das Ergebnis ist ein neues Length-Object und hat die gleiche Einheit wie das Objekt auf dem die Methode aufgerufen wird.
  - toString() gibt die Länge im folgenden Format zurück: `1.0m`, `3.121ft`, `0.1au`
- Beachten Sie die Hinweise auf der nächsten Seite

Hinweise

- Hinweis
  - Generieren Sie bei ungültigen Parametern eine `IllegalArgumentException`
  - Verwenden Sie wenn nötig **`private`**-Methoden um Code-Duplikation zu vermeiden
  - Achten Sie darauf, dass die Klasse wirklich unveränderlich ist
  - Sie können zum Testen ein eigenes kleines Testprogramm schreiben und/oder die JUnit-Tests in length/LengthTest.java verwenden

Bonus-Aufgabe

- Schreiben Sie ein Java-Programm `LengthConverter`, das als Argumente eine Länge, eine Quell- und eine Zieleinheit annimmt und die Länge von der Quelleinheit in die Zieleinheit konvertiert. Beispiel:

```
java LengthConverter 2.32 m ft
7,606557ft
```

- Hinweis: Erweitern Sie die **enum**-Klasse `LengthUnit` um eine statische Methode `fromSymbol`

```
public static LengthUnit fromSymbol(String symbol){
  // returns LengthUnit belonging to symbol
}
```

  - Die Methode liefert die Einheit zum übergebenen Symbol
  - Beispiel

```
LengthUnit.fromSymbol("in") // -> LengthUnit.INCH
```