

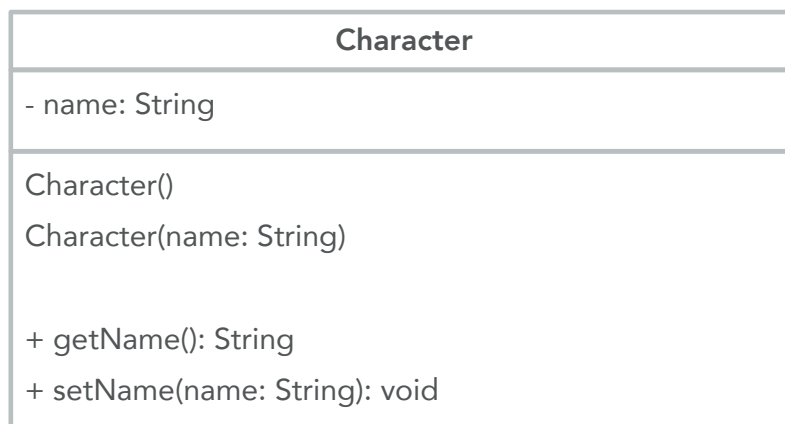
Fakultät Wirtschaft - Wirtschaftsinformatik

Aufgaben: Fortgeschrittene Programmierung (SS 2024)

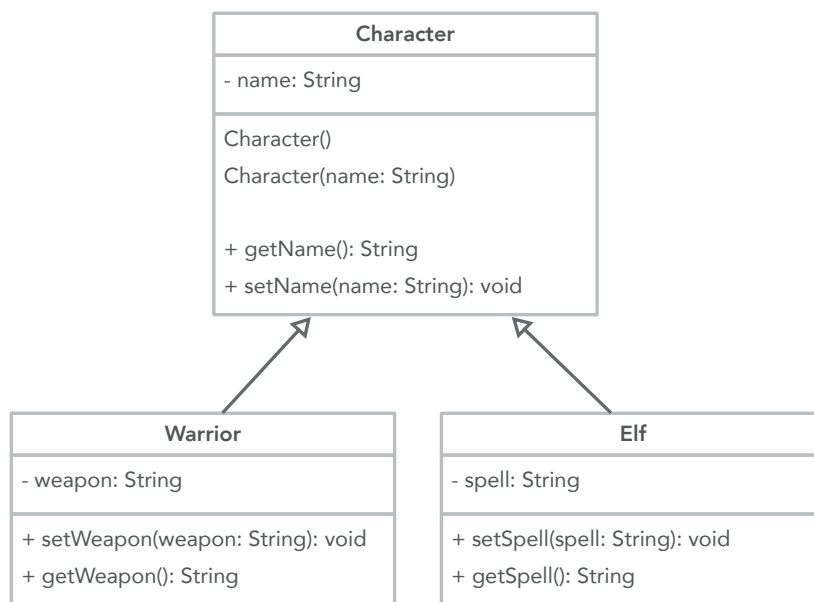
Studienrichtung: EG/EH

1. Wiederholung Klassen!

- (6) (a) Erstellen Sie die Klasse **Character** nach folgendem Klassendiagramm.

Abbildung 1: Klassendiagramm **Character**

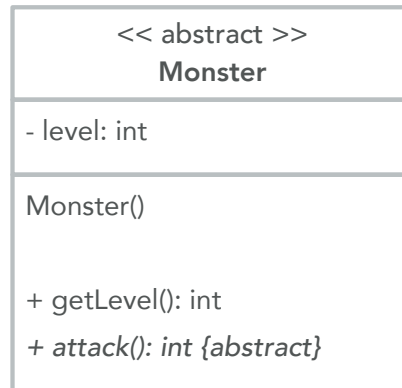
- (5) (b) Legen Sie zwei Objekte dieser Klasse an. Geben Sie einem Objekt den Namen über den Konstruktor mit. Dem zweiten Objekt über die Setter Methode. Lassen Sie sich dann von beiden Objekten den Namen in der Konsole ausgeben.

Abbildung 2: Klassendiagramm **Character** mit Kindklassen

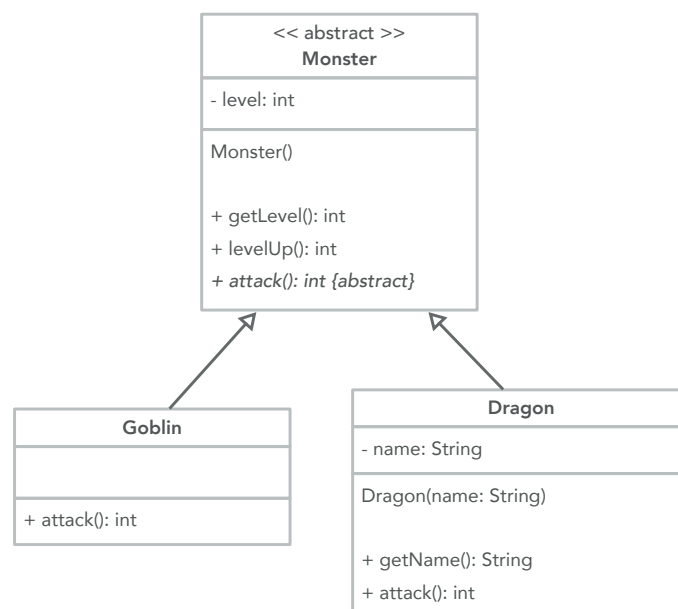
- (5) (c) Legen Sie eine zweite Klasse **Warrior** an, die von der Klasse **Character** abgeleitet ist (s. Abb. 2).
- (5) (d) Legen Sie eine dritte Klasse **Elf** an, die von der Klasse **Character** abgeleitet ist (s. Abb. 2).
- (10) (e) Legen Sie jeweils ein Objekt der Klasse **Elf** und eines der Klasse **Warrior** an. Vergeben Sie für beide Objekte Namen. Das Objekt der Klasse **Warrior** soll dann ein **Schwert** als Waffe erhalten. Das Objekt **Elf** Zauberspruch **Feuerball**. Lassen Sie sich anschließend den Namen der Objekte und ihrer jeweiligen Waffen auf der Konsole ausgeben.
- (5) (f) Erweitern Sie die Klasse **Warrior** so, dass es möglich ist, ihr über den Konstruktor einen Namen zuzuweisen. Es gibt dafür zwei Möglichkeiten. Schreiben Sie beide auf.

2. Wiederholung Abstrakte Klassen!

- (6) (a) Erstellen Sie die abstrakte Klasse **Monster** nach folgendem Klassendiagramm.

Abbildung 3: Klassendiagramm **Monster**

- (4) (b) Legen Sie nun eine Kindklasse **Goblin** an, in der Sie die abstrakte Methode `attack()` auscodieren (s. Abb. 4). Als Rückgabewert soll hierbei das Level multipliziert mit 10 genommen werden. Legen Sie dann ein Objekt dieser Klasse an. Lassen Sie sich das Level ausgeben, bevor und nachdem der Goblin einmal gelevelt hat. Lassen Sie sich dann den Attack-Wert anzeigen.

Abbildung 4: Klassendiagramm **Monsters**

- (4) (c) Legen Sie nun die Kindklasse **Dragon** an, die über die Eigenschaft **name** verfügt. Legen Sie einen eigenen Konstruktor an und codieren Sie die Methode **attack()** aus. Diese soll den 200-fachen Wert des Levels zurückgeben (s. Abb. 4). Legen Sie dann ein Objekt dieser Klasse an und lassen sich den Namen, das Level und den Attack-Wert anzeigen.

3. Interfaces

- (10) (a) Implementieren Sie die Klassen **Monster** und **Dragon** aus dem Klassendiagramm 5. Legen Sie drei Objekte der Klasse **Dragon** an und geben Sie jedem Drachen einen Namen und eine Startposition. Lassen Sie sich dann von allen Drachen den Namen und die Position in der Konsole ausgeben.

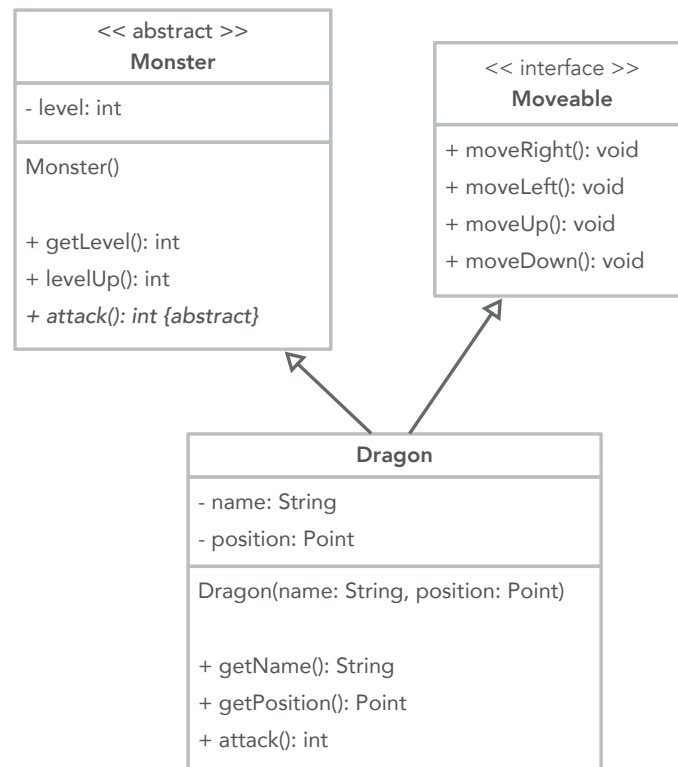


Abbildung 5: Klassendiagramm Interface

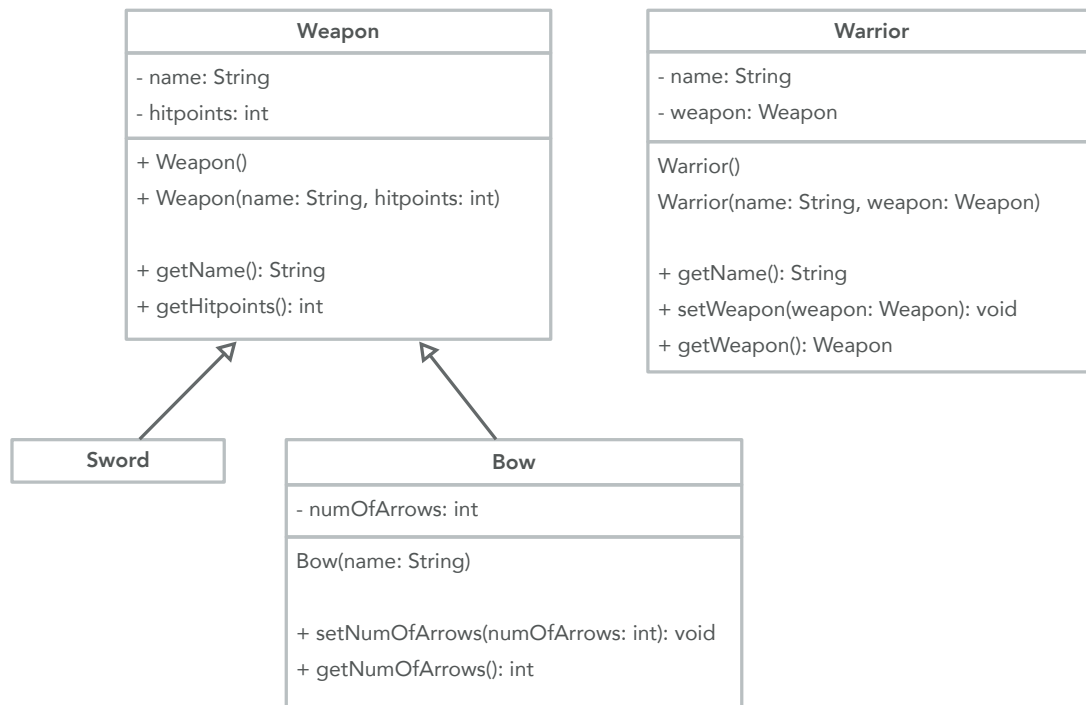
- (5) (b) Erstellen Sie das Interface **Moveable** (s. Abb. 5).
- (5) (c) Erweitern Sie die Klasse **Dragon** um das Interface **Moveable**. Implementieren Sie die vier übergebenen Methoden. Bewegen Sie dann die drei Drachen jeweils einen Schritt in eine beliebige Richtung. Lassen Sie sich die jeweiligen Schritte auf der Konsole ausgeben.
- (10) (d) Erweitern Sie die Klasse **Dragon** um die private Eigenschaft **direction**. Diese hat den Datentyp **String** und enthält die Bewegungsrichtung (**w**, **a**, **s**, **d**). Erstellen Sie die zugehörige Getter- und Setter-Methode. Erweitern Sie das Interface **Moveable** um die Methode **move**.

Implementieren Sie diese in der Klasse **Dragon**. Die Methode soll abhängig vom Wert der Eigenschaft **direction** die entsprechende Methode (**moveUp()**, **moveLeft()**, **moveDown()**, **moveRight()**) aufrufen.

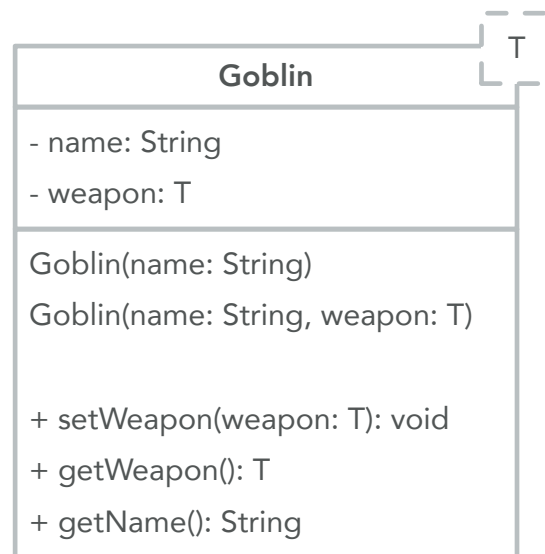
- (5) (e) Erstellen Sie eine Klasse **Mover**. Diese enthält eine statische Methode **moveDragons**, an die als Parameter die **Dragon**-Objekte übergeben werden. Innerhalb der Methode werden dann die Drachen bewegt (Methode **move**).
- (10) (f) Erstellen Sie eine Klasse **Character** (vgl. Abb. 1). Diese enthält analog zur Klasse **Dragon** die Methoden **moveUp()**, **moveLeft()**, **moveDown()**, **moveRight()** sowie die Methode **move**. Der **Character** darf pro Bewegung 2 Felder vorrücken. Erstellen Sie außerdem die Eigenschaft **direction** mit der zugehörigen Getter- und Setter-Methode.
- (2) (g) Erweitern Sie die Klasse **Mover** um eine weitere statische Methode **moveAll**. Diese nimmt analog zur Methode **moveDragons** bewegliche Objekte entgegen und bewegt diese über die Methode **move**.
- (4) (h) Legen Sie ein Objekt der Klasse **Character** an und initialisieren Sie dieses mit sinnvollen Werten. Rufen Sie dann die Methode **moveAll** der **Mover**-Klasse auf und übergeben an diese alle Drachen und das **Character**-Objekt. Lassen Sie sich alle Schritte zur Kontrolle auf der Konsole ausgeben.

4. Generics

- (12) (a) Erstellen Sie die Klasse **Weapon** mit ihren beiden Kindern aus Klassendiagramm 6. Erstellen Sie außerdem die Klasse **Warrior**.

Abbildung 6: Klassendiagramm **Generics**

- (6) (b) Legen Sie dann ein Objekt für einen Bogen, eines für ein Schwert und zwei für einen Krieger an. Ein Krieger erhält als Waffe den Bogen, der andere das Schwert. Lassen Sie sich den Namen des Kriegers und den Namen der jeweiligen Waffe in der Konsole ausgeben. Versuchen Sie sich die Anzahl der Pfeile des Bogens ausgeben zu lassen.
- (5) (c) Legen Sie die Klasse **Goblin** an (s. Abb. 7). Diese wird mittels generischer Typen erstellt, d.h. die Eigenschaft **weapon** ist generisch und hat in der Klasse noch keinen Datentyp. Dieser wird erst zur Laufzeit festgelegt.
- (5) (d) Legen Sie zwei Objekte der Klasse **Goblin** an und weisen einem den Bogen und dem anderen das Schwert aus Teilaufgabe 4.b zu. Lassen Sie sich dann den Namen der Goblints sowie ihrer Waffen auf der Konsole ausgeben. Lassen Sie sich dann auch die Anzahl der Pfeile des Bogens

Abbildung 7: Klassendiagramm **Goblin**

ausgeben.

5. Dragons vs. Wizards - Graphical User Interface (GUI)

Wir wollen nun ein Spiel erstellen, in dem - wie der Name schon sagt - Drachen gegen Zauberer kämpfen (s. Abb. 8). Nach Auswahl eines Drachens und eines Zauberers sollen die jeweiligen Werte angezeigt werden, ehe der Kampf beginnt. Das Ergebnis des Kampfes soll ebenfalls angezeigt werden. Der Verlierer des Kampfes wird aus der jeweiligen Auswahlliste entfernt. Das Spiel ist gewonnen, wenn ein Team keinen Kämpfer mehr hat.



Abbildung 8: Dragons vs. Wizards

- (10) (a) Erstellen Sie das Programm „Dragons vs. Wizards“. Fügen Sie eine GUI-Form hinzu (File → New → Swing UI Design → GUI Form). Geben Sie der Form einen Namen (bspw. Arena). Platzieren Sie vier Textfelder (JTextField) in zwei Spalten darauf.

Fügen Sie jeweils ein Label hinzu. Wundern Sie sich über die seltsame Anordnung. Benennen Sie die Eingabefelder in der linken Spalte `dragonName` und `dragonHitpoints`. Die Eingabefelder in der rechten Spalte `wizardName` und `wizardHitpoints`. Fügen Sie dann einen Button (JButton) unterhalb hinzu. Und ein weiteres Label unterhalb dieses Buttons.

Den Button beschriften Sie mit „Fight!“ und das Label erhält den Namen `resultLog`. Geben Sie dem `JPanel` einen Namen (`field name`) (bspw. `arena`).

Fügen Sie der eingebundenen Klasse `Arena` folgende Methode hinzu:

```
1 public static void main(String[] args) {  
2     JFrame frame = new JFrame("Arena");  
3     JPanel w = new Arena().arena;  
4     frame.setContentPane(w);  
5     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
6     frame.pack();  
7     frame.setVisible(true);  
8 }
```

Lassen Sie das Programm laufen und erfreuen Sie sich an dem Ergebnis.

- (10) (b) Nun, da das Formular läuft, können die Spiele beginnen.

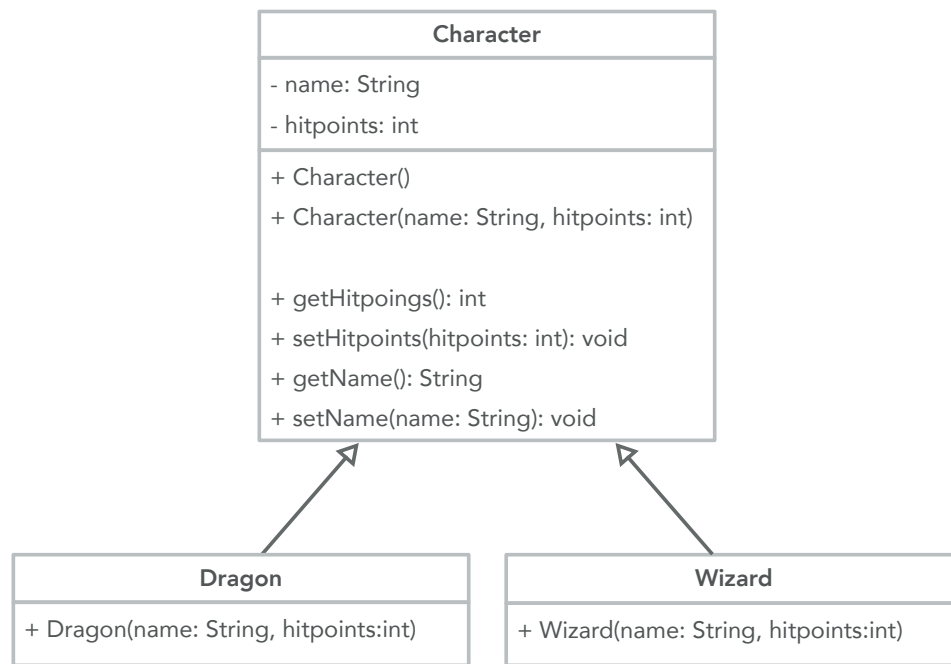
Wir möchten nun, dass der Spieler für einen Drachen und einen Zauberer einen Namen, sowie dessen Hitpoints eingibt, bei einem Klick auf den Fight-Button, beginnt der Kampf und der Spieler mit den höheren Hitpoints gewinnt. Das Ergebnis des Kampfes lassen Sie sich in dem Label unterhalb des Buttons ausgeben. Wenn also Smaug (100 Hitpoints) gegen Bilbo (10 Hitpoints) kämpft, sähe die Ausgabe so aus: **Smaug gewinnt gegen Bilbo (100:10)**

Erstellen Sie hierfür einen Action-Listener (RMB → **Create Listener** → **ActionListener**).

Erstellen Sie darin den entsprechenden Code. Testen Sie alle drei möglichen Fälle durch.

- (10) (c) Erstellen Sie die Klassen, wie sie in Abb. 9 angegeben sind. Legen Sie dann im Konstruktor der `Arena` Klasse jeweils ein Objekt der Klasse `Dragon` und der Klasse `Wizard` an. Diese beiden Objekte sollen nun gegeneinander kämpfen. Dazu sollen ihre Namen und Hitpoints in die jeweiligen Felder geschrieben werden.

- (10) (d) Erweitern Sie das Spiel so, dass in den beiden Spalten jeweils ein DropDown (`JComboBox`) Feld eingerichtet ist, das alle möglichen Drachen bzw. Zauberer anzeigt, die gegeneinander kämpfen sollen. Bei Auswahl eines Drachen werden dessen Werte in die Eingabefelder über-

Abbildung 9: Klassendiagramm **Character**

nommen. Analog beim Zauberer. Legen Sie hierfür jeweils drei Objekte an und fügen Sie diese dem Auswahlfeld hinzu.

- (10) (e) Erweitern Sie das Spiel so, dass die Zauberer und Drachen einen Avatar erhalten, der angezeigt wird, wenn diese ausgewählt werden. Verwenden Sie zur Anzeige die `Icon`-Eigenschaft eines Labels.
- (10) (f) Erweitern Sie die Figuren so, dass sie jeweils einen Special Move haben, der die Hitpoints einmalig erhöht. Dieser Special Move hat eine gewisse Ausführungswahrscheinlichkeit und wird daher nicht immer ausgeführt.
- (4) (g) Erweitern Sie das Spiel so, dass der Verlierer aus dem Auswahlfeld gelöscht wird.
- (5) (h) Erweitern Sie das Spiel so, dass ein Game Over Screen angezeigt wird, wenn ein Team keine Spieler mehr hat.

6. Battle of the 8 Armies - Multi-Threading

Jedes Programm, das abläuft, läuft in einem eigenen Thread. Es können aber nicht nur vollständige Programme in einem Thread ablaufen, sondern ein Programm kann in Teilaufgaben zerlegt werden, die in jeweils einem eigenen Thread laufen. Verfügt der Rechner über mehrere Prozessoren (oder Kerne), können diese Threads parallel bearbeitet werden. In diesem Fall spricht man von Nebenläufigkeit oder Multi-Threading.

Zunächst wollen wir uns mit dem Lebenszyklus von Thread beschäftigen. Java Threads haben fünf Zustände (States).

- New
- Active (Runnable and Running)
- Blocked/Waiting
- Timed Waiting
- Terminated (Dead)

Wenn ein Thread angelegt wird, hat er den Zustand NEW. Nach dem Starten (Methode `start()`) geht er in den Zustand RUNNABLE über. Während er läuft, hat er den Zustand RUNNING. Wartet der Thread auf ein Ergebnis von außen, bspw. die Ergebnisse einer Datenbankanfrage, ist er im Zustand (TIMED) WAITING. Wird der Thread beendet geht er in den Zustand TERMINATED über (s. Abb. 10).

Wir wollen dies nun mit einem kleinen Programm ermitteln.

- (10) (a) Erstellen Sie ein neues Projekt. Legen Sie darin die in Abbildung 11 angegebenen Klassen an.
- (10) (b) Nachdem die Grundgerüste stehen, müssen wir Objekte der jeweiligen Klassen anlegen. Legen Sie dazu in der `main`-Methode ein Objekt der Klasse `CounterCommand` an. Verwenden Sie die hierfür zuvor definierte statische Variable `c`.

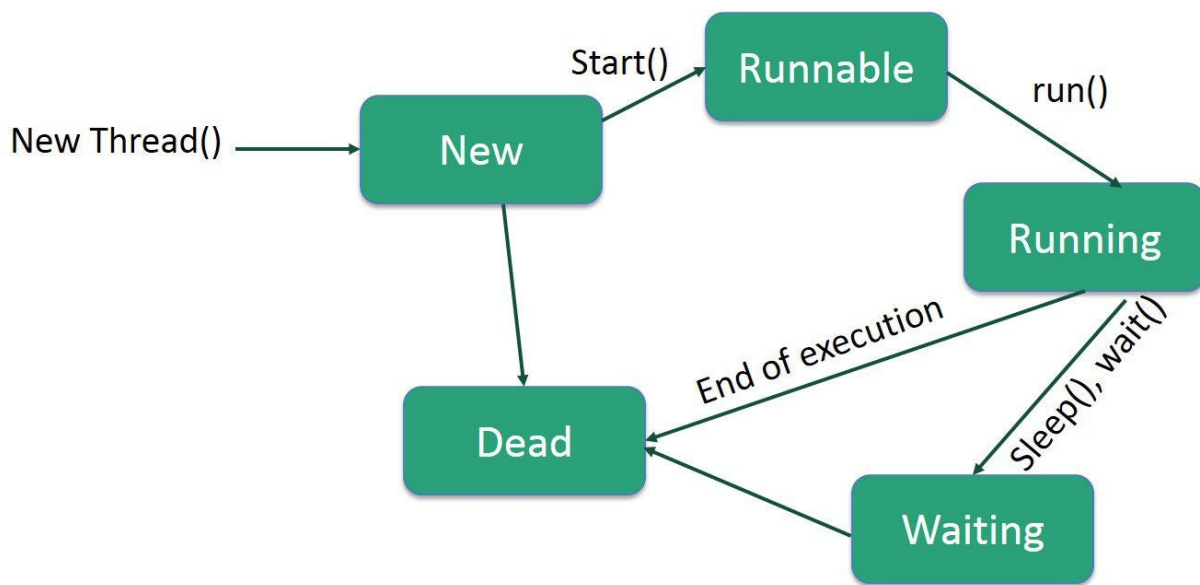


Abbildung 10: Java Thread Lebenszyklus

Legen Sie dann ein Thread Objekt an und weisen dieses der statischen Variablen `ct` zu. Der Konstruktor des Threads verlangt ein `Runnable` Objekt. Dieses haben Sie mit `c` angelegt. Übergeben Sie dieses und übergeben Sie als zweiten Parameter einen selbstgewählten Namen für den Thread.

Verfahren Sie analog für das `DateCommand` Objekt und den entsprechenden Thread.

Lassen Sie sich den Namen und Zustand beider Threads in der Konsole ausgeben. Verwenden Sie hierfür die Getter Methoden des Thread Objekts für den Namen und den Zustand (State).

Achtung! Damit das Programm läuft, müssen Sie in den Klassen `CounterCommand` und `DateCommand` die Methode `run()` zumindest anlegen, auch wenn sie im Moment noch keinen Inhalt hat.

Lassen Sie das Programm laufen. Beide Threads sollten im Zustand `NEW` sein.

- (4) (c) Starten Sie nun beide Threads mit der Methode `start()`. Lassen Sie sich dann erneut die Zustände auf der Konsole ausgeben. Diese sollten nun im Zustand `RUNNABLE` sein.
- (8) (d) Implementieren Sie nun in den Klassen `DateCommand` und `CounterCommand` die Methode

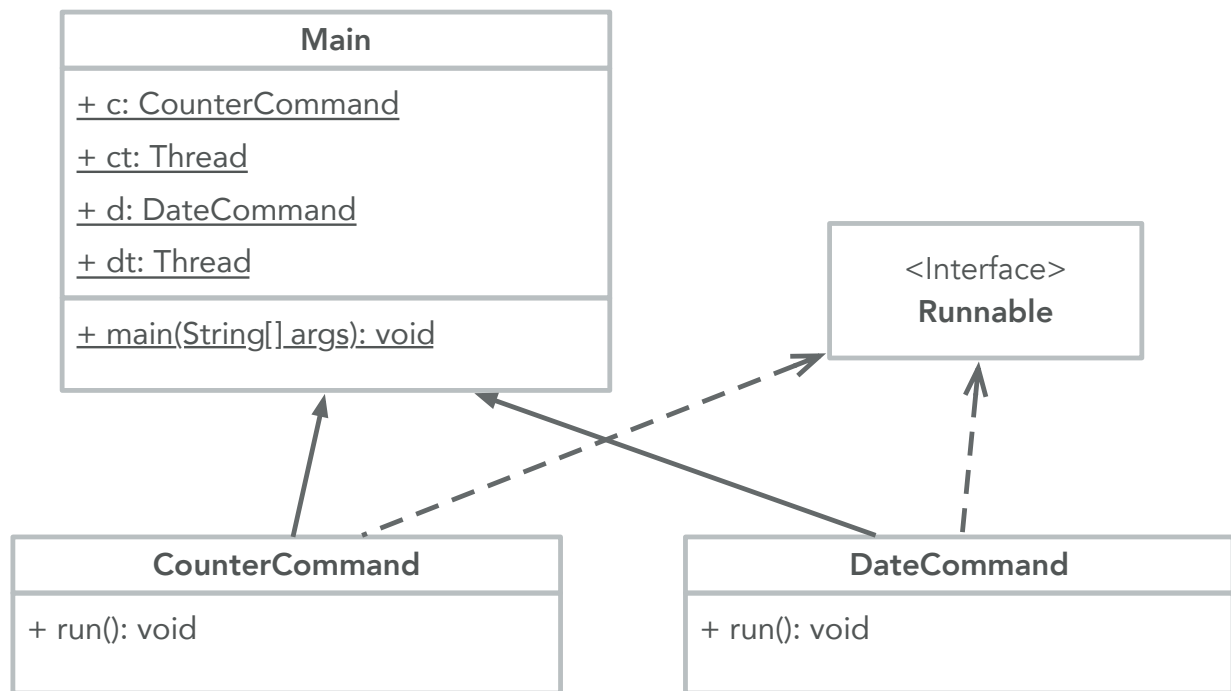
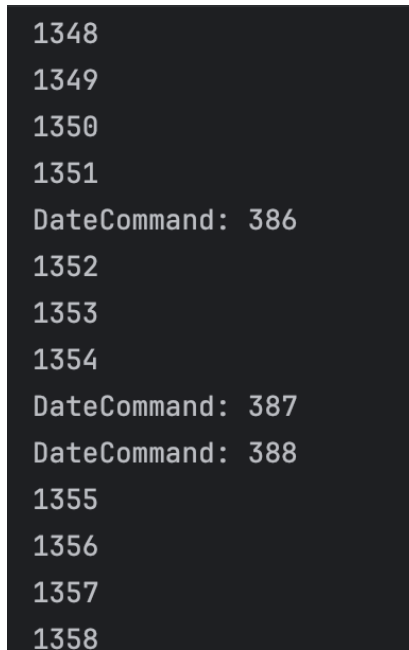


Abbildung 11: Klassendiagramm Threads

`run`. Lassen Sie sich zunächst den Status in der Konsole ausgeben. Zählen Sie dann in einer Schleife von 1 bis 200 und lassen sich jede Zahl auf der Konsole ausgeben. Lassen Sie sich bei jeder Zahl den Namen der ausführenden Klasse mit anzeigen. Führen Sie dies in beiden Klassen durch. Lassen Sie das Programm dann laufen und schauen sich das Ergebnis auf der Konsole an.

- (5) (e) Mit der Methode `join` des Thread-Objektes können Sie auf einen anderen Thread warten. Wenn Sie also in der `main`-Methode `ct.join()` aufrufen, wartet das Hauptprogramm solange, bis der Thread `ct` zu Ende ist. Rufen Sie dies nun auf und lassen Sie sich anschließend die Zustände beider Threads anzeigen.
- (4) (f) Erweitern Sie **DateCommand** nun so, dass sie in der Methode `run` einen weiteren Befehl am Ende der Methode eingeben. Lassen Sie sich hier den Zustand des **CounterCommand**(!) Objektes ausgeben. (Tipp: Da es sich bei dem **CounterCommand** Objekt um eine statische Eigenschaft der Klasse **Main** handelt, können Sie diese über die Klasse **Main** innerhalb der Klasse **DateCommand** aufrufen.)

- (1) (g) Führen Sie das Programm mehrfach aus und vergleichen Sie die Konsolenausgaben. Sie sollten die Verschränkung der Threads sehen können. Falls Sie die Verschränkung nicht sehen, lassen Sie beide Objekt bis 1000 zählen (s. Abb. 12).



```
1348
1349
1350
1351
DateCommand: 386
1352
1353
1354
DateCommand: 387
DateCommand: 388
1355
1356
1357
1358
```

Abbildung 12: Konsolenausgabe Threads: Die unmarkierten Zahlen sind Ausgaben des CounterObjektes

7. Battle of the 8 Armies - MultiThreading II

Wir wollen nun das Programm *Battle of the 8 Armies* erstellen, in dem acht Armeen wie in einem Turnier gegeneinander kämpfen. Immer zwei Armeen kämpfen gegeneinander. Die jeweils verbliebenen Armeen kämpfen in der nächsten Runde wieder gegeneinander usw. bis am Ende nur noch eine Armee übrig bleibt. Da die Kämpfe alle gleichzeitig stattfinden, bietet sich hier die nebenläufige Programmierung an.

- (10) (a) Erstellen Sie die Klassen aus Abb. 13.

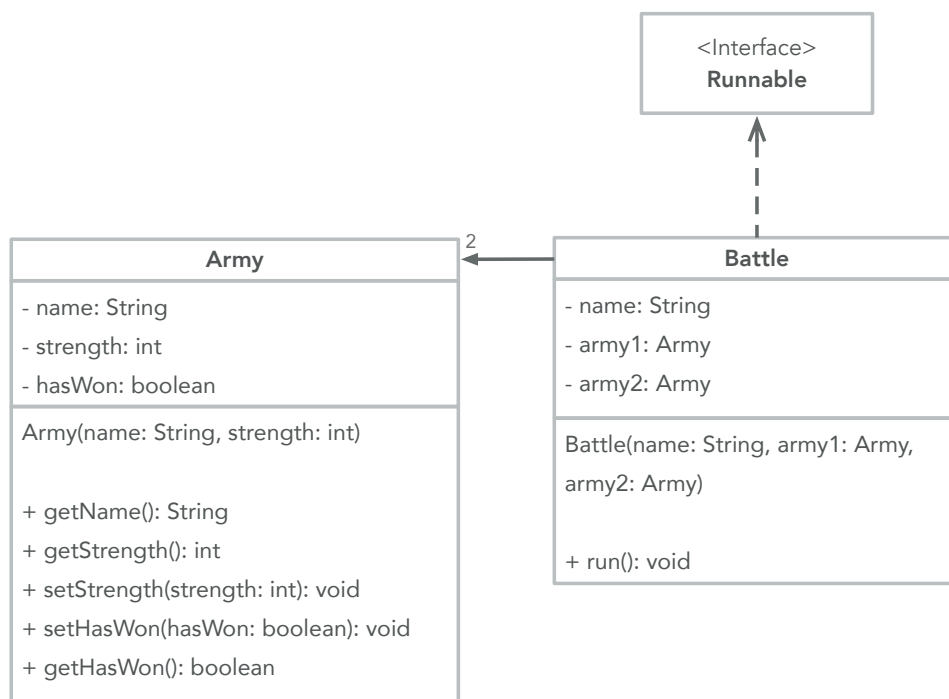


Abbildung 13: Klassendiagramm des *Battle of the 8 Armies*

- (10) (b) Erstellen Sie nun die Methode **run** in der Klasse **Battle**. Lassen Sie sich zunächst die Namen und Stärke der beiden zugewiesenen Armeen ausgeben. Lassen Sie schließlich beide Armeen gegeneinander kämpfen. Die Armee mit dem höheren Stärke-Wert gewinnt. Lassen Sie sich den Sieger auf der Konsole ausgeben. Die siegreiche Armee verliert so viel Stärke, wie ihrer gegnerische Armee hatte.
- (10) (c) Legen Sie nun in der `main`-Methode der `Main` Klasse acht Objekte an, für jede Armee eins.

Lassen Sie nun die Armeen gegeneinander antreten. Die Sieger des Viertelfinales treffen dann im Halbfinale aufeinander. Die beiden Sieger des Halbfinals stehen sich im Finale gegenüber. Ermitteln Sie so die stärkste Armee.