

Programmieren II: Java

5. Praktikum (Abgabe 5. Juli, 24 Uhr)

Auer, Hanel, Jürgensen, Lehner, Riemenschneider



Lernziele

Aufgabe 1: Iterator für zweidimensionale Arrays

Aufgabe 2: Videospiel-Bibliothek

Inhalt

Hinweise zum Praktikum

Hinweise zum Praktikum

- ▶ Abgabetermin dieses Praktikums: **Abgabe 5. Juli, 24 Uhr**
- ▶ Sie dürfen die Aufgaben **alleine** oder zu **zweit** bearbeiten und 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

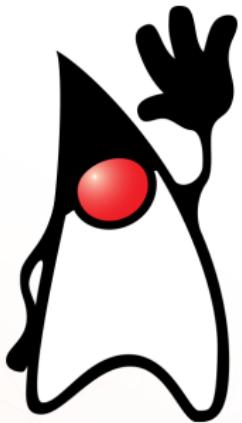
Lernziele

Lernziele

Willkommen zum 5. Praktikum!

Lernziele

- ▶ Iteratoren selbst implementieren
- ▶ Collections: erstellen, befüllen, anwenden
- ▶ Comparable: Objekte vergleichen, sortieren



Inhalt

Aufgabe 1: Iterator für zweidimensionale Arrays

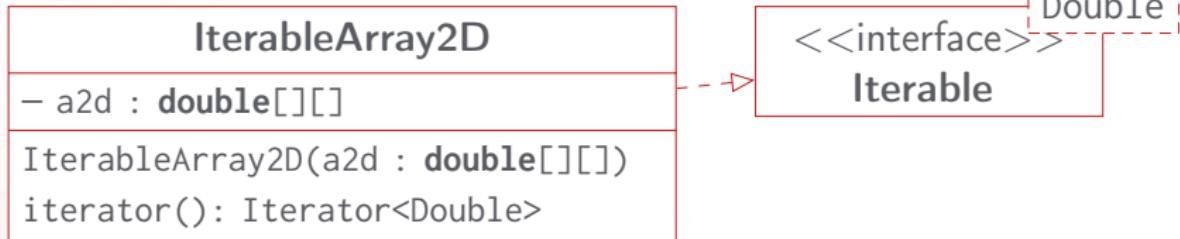
Einführung

- ▶ Um einen zweidimensionalen **double**-Array zu durchlaufen braucht es **zwei geschachtelte** Schleifen:

```
public double sum(double a[][]){  
    double sum = 0;  
    for (int i = 0; i < a.length; i++){ // 1. Schleife  
        for (int j = 0; j < a[i].length; j++){ // 2. Schleife  
            sum += a[i][j];  
        }  
    }  
    return sum;  
}
```

- ▶ Wir erstellen eine **Wrapper-Klasse** IterableArray2D für **zweidimensionale double[][]-Arrays**
 - ▶ IterableArray2D ist **iterierbar**
 - ▶ Ein Iterator von IterableArray2D **durchläuft** die Elemente wie in den obigen **geschachtelten** Schleifen

IterableArray2D



- ▶ a2d ist die **Referenz** auf den zweidimensionalen **double-Array**
- ▶ IterableArray2D **implementiert** Iterable<Double>
 - ▶ iterator() erzeugt einen Iterator<Double>...
 - ▶ der die Elemente von a2d **durchläuft**, z.B. äquivalent zu oben:

```
public double sum(double a[][]){  
    double sum = 0;  
    for (double e : new IterableArray2D(a)){  
        sum += e;  
    }  
    return sum;  
}
```

IterableArray2D

✓ Implementieren Sie IterableArray2D:

- ▶ Konstruktor IterableArray2D(**double[][] a2d**) mit
 - ▶ a2d darf **nicht null** sein
 - ▶ Kein Eintrag a2d[i] darf **null** sein oder Länge **0** haben (**a2d[i].length == 0**)
 - ▶ Generieren Sie eine **IllegalArgumentException** wenn eine der **vorherigen Eigenschaften** nicht erfüllt ist
- ▶ iterator() erzeugt einen Iterator<Double>
 - ▶ Implementieren Sie den Iterator als **private innere Klasse** (**nicht static**) von IterableArray2D mit Namen **Array2DIterator**
 - ▶ hasNext() und next() implementieren Sie wie in der Vorlesung angegeben
 - ▶ Wird next() aufgerufen wenn es **kein nächstes Elemente** mehr gibt, erzeugen Sie eine **NoSuchElementException**
- ▶ Verwenden Sie zum Testen ein eigenes Hauptprogramm und
 - array2d-iterator/IterableArray2DTest.java

Inhalt

Aufgabe 2: Videospiel-Bibliothek

Einführung

Platform

Game

GamesLibrary

Inhalt

Aufgabe 2: Videospiel-Bibliothek

Einführung

Einführung

- ▶ Ihre **Videospiel-Sammlung** ist über die Jahre sehr gewachsen
- ▶ Wir schreiben ein kleine **Java-Bibliothek** zur Verwaltung Ihrer Videospiel-Sammlung
- ▶ Dazu verwenden wir **Collections** und das **Comparable**-Interface
- ▶ Definieren Sie alle Klassen im **Package gameslib**
- ▶ Verwenden Sie zum Testen
 - ▶ ein **eigenes Hauptprogramm** mit Beispieldaten von
<https://www.metacritic.com/game>
 - ▶ die **JUnit-Tests**
 - `gameslib/PlatformTest.java` ,
 - `gameslib/GameTest.java` und
 - `gameslib/GamesLibraryTest.java` ,

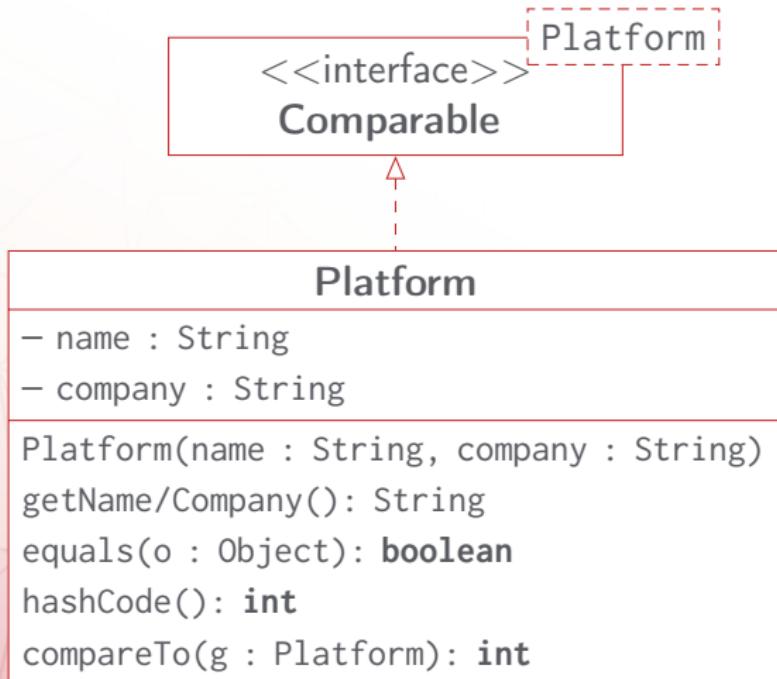


Inhalt

Aufgabe 2: Videospiel-Bibliothek

Platform

Platform



- ▶ Platform modelliert eine **Videospiel-Plattform** (z.B. „Playstation 4“ oder „PC“)

Platform



Implementieren Sie Platform wie folgt

► Platform ist **unveränderlich**

► **Attribute**

► name — Name der Plattform (z.B. „Switch“, „Playstation 4“, etc.), **unveränderlich**, darf **nicht null** oder **leer** sein

► company — Hersteller der Plattform (z.B. „Nintendo“, „Sony“, etc.), **unveränderlich**, darf **nicht null** oder **leer** sein

► **Methoden**

► Konstruktor **prüft** und **initialisiert** die Attribute

► Getter für name und company

► equals vergleicht **alle Attribute** (muss von Ihnen implementiert werden)

► hashCode können Sie generieren lassen

Platform — Comparable

- ▶ Platform implementiert Comparable<Platform>
- ▶ `int Comparable.compareTo(Platform p)`
soll die Attribute in folgender Reihenfolge vergleichen
 - 1. name
 - 2. company
- ▶ Implementieren Sie compareTo wie in der Vorlesung vorgestellt

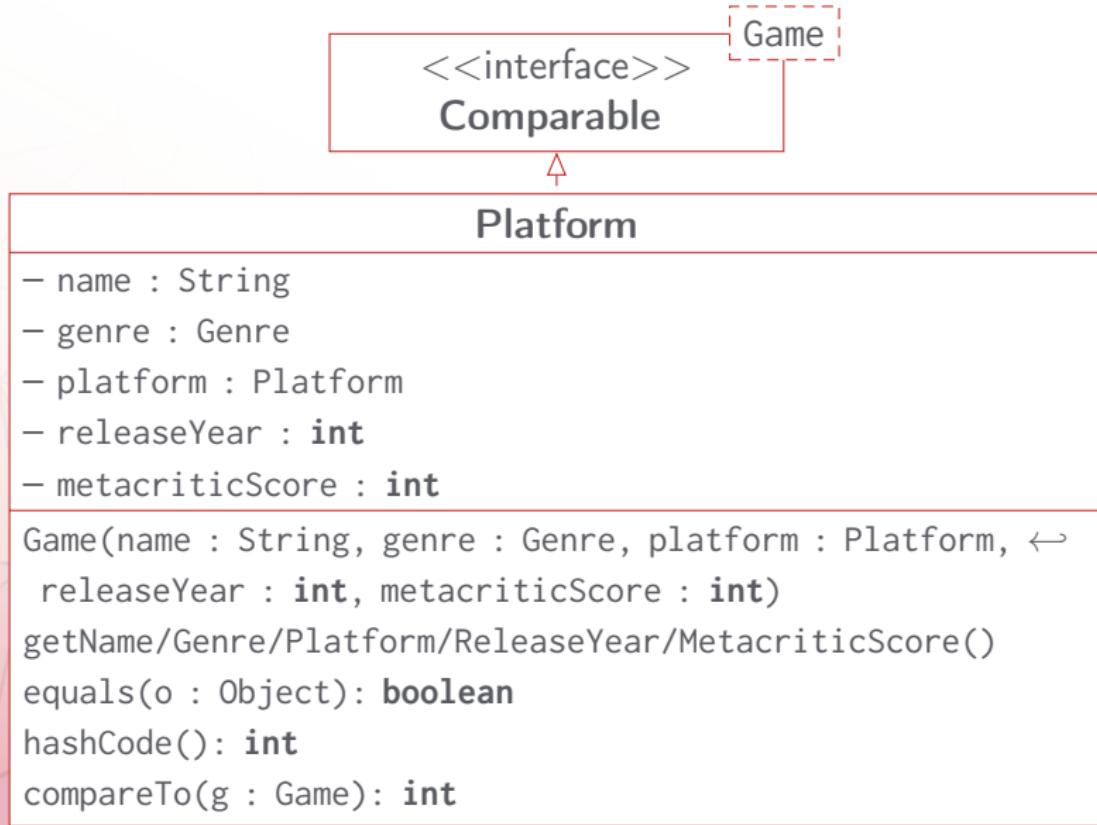


Inhalt

Aufgabe 2: Videospiel-Bibliothek

Game

Game



► Game modelliert ein **Videospiel**

Game

Implementieren Sie Game wie folgt



- ▶ Game ist **unveränderlich**
- ▶ **Attribute (alle unveränderlich)**
 - ▶ name — Name des Spiels (z.B. „Half-Life“, „World of Warcraft“, etc.), **nicht null** oder **leer**
 - ▶ genre — Genre des Spiels (**enum**), **nicht null**;
Genre ist bereits gegeben in
 - `gameslib/Genre.java`
 - ▶ platform — Plattform, für das Sie das Spiel besitzen, **nicht null**
 - ▶ releaseYear — Jahr der Erscheinung, ≥ 1970
 - ▶ metacriticScore — **Metacritic Score** von der Seite <http://www.metacritic.com/>, ≥ 0 und ≤ 100
- ▶ Methoden
 - ▶ Konstruktor prüft und initialisiert die Attribute
 - ▶ Getter für alle Attribute

Methoden (Fortsetzung)



- ▶ `equals` vergleicht alle Attribute (muss von Ihnen implementiert werden)
- ▶ `hashCode` können Sie generieren lassen
- ▶ Game implementiert Comparable<Game>
- ▶ int Comparable.compareTo(Game g) soll die Attribute in folgender Reihenfolge vergleichen
 1. name
 2. genre (verwenden Sie Genre.compareTo)
 3. platform (Beachten Sie: Platform implementiert Comparable<Platform>)
 4. releaseYear
 5. metacriticScore (gameA < gameB wenn Metacritic Score von gameA kleiner als die von gameB)
- ▶ Implementieren Sie compareTo wie in der Vorlesung vorgestellt

Inhalt

Aufgabe 2: Videospiel-Bibliothek

GamesLibrary

GamesLibrary

- ▶ GamesLibrary hat folgende Aufgaben
 - ▶ Verwalten von **Plattformen** (hinzufügen, entfernen)
 - ▶ Verwalten von **Spielen** (hinzufügen, entfernen)
 - ▶ Erstellung verschiedener **Abfragen** und **Statistiken**
- ▶ Für GamesLibrary ist ein **Grundgerüst** vorgegeben
 - ▶ `gameslib/GamesLibrary.java` mit allen Methodensignaturen und JavaDoc
 - ▶ `gameslib/GamesLibraryException.java` eigene Ausnahmeklasse
- ▶ **Wichtig:** Die **vollständige Spezifikation** der Funktionsweise der zu implementierenden Methoden ist als JavaDoc in `gameslib/GamesLibrary.java` gegeben!



GamesLibrary

- ✓ Implementieren Sie das Hinzufügen und Entfernen von Plattformen und Spielen
 - ▶ Verwalten Sie die Plattformen und Spiele in **je einer geeigneten Java-Collection** mit den Eigenschaften
 - ▶ **keine Duplikate** erlaubt
 - ▶ **Sortierung** nach Platform/Game.compareTo
 - ▶ Methoden
 - ▶ addPlatform fügt neue Platform hinzu
 - ▶ addGame fügt neues Spiel hinzu (**dazugehörige Platform muss bereits vorhanden sein**)
 - ▶ removeGame entfernt vorhandenes Spiel
 - ▶ removePlatform entfernt vorhandene Platform **und alle dazugehörigen Spielen**
- ✓ Implementieren Sie folgende Methoden in GamesLibrary
 - ▶ Set<Game> getPlatformsReadOnly() gibt die sortierte Menge der Plattformen als **read-only** Set zurück
 - ▶ Set<Game> getGamesReadOnly() gibt die sortierte Menge der Spiele als **read-only** Set zurück

GamesLibrary

✓ Implementieren Sie folgende Methoden in GamesLibrary

- ▶ Game getBestGame() liefert das Spiel mit der höchsten metacriticScore zurück (oder **null** wenn Sammlung leer)
- ▶ Map<Genre, Integer> getGenreCount() erstellt eine Map von Genre auf **Anzahl der Spiele** mit dem jeweiligen Genre (z.B. genreCount.get(Genre.ACTION)== 12 Spiele);
Map<Genre, Integer> muss **nicht sortiert** sein
- ▶ Map<Platform, Set<Game>> getGamesForPlatform() erstellt eine Map die eine vorhandene Plattform auf die Spiele der Plattform abbildet; im Rückgabetyp müssen Map und Set **nicht sortiert** sein
- ▶ Map<Platform, Double> getAverageScoreForPlatform() liefert eine Map die eine vorhandene Plattform auf die **durchschnittliche Metacritic Score** abbildet (**nicht sortiert**); verwenden Sie getGamesForPlatform für die Implementierung
- ▶ Platform getBestPlatform() gibt die Plattform mit der **höchsten durchschnittlichen Metacritic Score** zurück (oder **null** wenn Sammlung leer); verwenden Sie getAverageScoreForPlatform für die Implementierung

Implementieren Sie folgende Methoden in GamesLibrary

- ▶ `List<Game> sortGamesByMetacriticScore()` erstellt eine **nach Metacritic Score absteigend sortierte Liste aller Spiele.** Sortieren Sie die Liste über `List.sort(Comparator)` und implementieren Sie einen geeigneten Comparator als **innere Klasse MetacriticScoreComparator** von GamesLibrary
- ▶ `List<Game> sortGamesByReleaseYear()` erstellt eine **sortierte Liste aller Spiele, aufsteigend nach releaseYear** und innerhalb eines Jahres nach **name**. Sortieren Sie die Liste wieder über `List.sort(Comparator)` und implementieren Sie einen geeigneten Comparator als **anonyme Klasse innerhalb der Methode**