Programmieren II (Java)

5. Praktikum: Collections und Iteratoren

Sommersemester 2023

Christopher Auer, Tobias Lehner

Abgabetermine

Lernziele

- `Iterator` und `Iterable`
- *Java-Collections*: erstellen, befüllen, bearbeiten und abfragen
- `Comparator` und `Comparable`

Hinweise

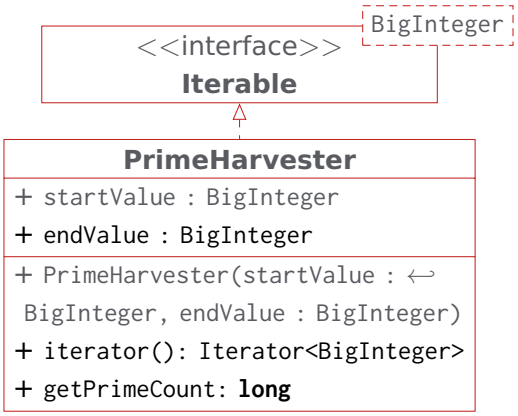
- Sie dürfen die Aufgaben *alleine* oder zu *zweit* bearbeiten und abgeben
- Sie müssen *4 der 5* Praktika bestehen
- *Kommentieren* Sie Ihren Code
  - Jede *Methode* (wenn nicht vorgegeben)
  - *Wichtige* Anweisungen/Code-Blöcke
  - *Nicht kommentierter* Code führt zu *Nichtbestehen*
- Bestehen Sie 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*!

Aufgabe 1: Prime-Harvester ⁂ bis ⁂

Die moderne asymetrische Kryptographie basiert auf sehr großen Primzahlen. Wir wollen eine Klasse schreiben, die sehr große Primzahlen in einem zuvor bestimmten Zahlenbereich zählen kann.

Um die Korrektheit Ihrer Implementierung zu testen, müssen Sie die gegebenen *JUnit-Tests* verwenden. Importieren Sie dazu das bereitgestellte *Gradle-Projekt* in `SupportMaterial/primeharvester` in Ihre Entwicklungsumgebung. Eine Main-Klasse `PrimeHarvesterMain` ist bereits gegeben. Testen Sie die Klasse `PrimeHarvester` auch damit!

- Implementieren Sie eine Klasse `PrimeHarvester` im Paket `primeharvester`.



- Bei `startValue` und `endValue` handelt es sich um `BigInteger`-Typen. Lesen Sie die Dokumentation zu `BigInteger` sorgfältig. Sie finden die Dokumentation unter `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/math/BigInteger.html`. Die Klasse `BigInteger` enthält hilfreiche Methoden im Umgang mit Primzahlen. *Aber Vorsicht: Die arithmetischen Operatoren funktionieren mit BigInteger nicht!*
- `startValue` und `endValue` sind unveränderlich. Der Zugriff ist deswegen auch von außen erlaubt.
- Der Konstruktor prüft, ob die übergebenen Werte ungleich **`null`** sind, ob `startValue` größer 1 ist und ob der `endValue` größer als `startValue` ist. *Vorsicht: Auch die Vergleichsoperatoren funktionieren mit BigInteger nicht!*
- Implementieren Sie den `Iterator` als *anonyme Klasse*. Diese greift *lesend* auf `startValue` und `endValue` in `PrimeHarvester` zu. Der Iterator soll innerhalb des Zahlenbereichs immer die nächste Primzahl liefern. Zum Beispiel soll der Iterator im Bereich zwischen 2 und 12 nacheinander die Zahlen 2,3,5,7,11 liefern.
- Die Methode `getPrimeCount()` verwendet den Iterator um die Zahlen im Zahlenbereich zu zählen und das Ergebnis zurückzugeben.

Aufgabe 2: Bundesliga ★☆ bis ★★

Die Bundesligasaison 22/23 ist vorbei und Bayern ist mit viel Getöse irgendwie mal wieder Meister geworden. Aber nach der Saison ist vor der Saison und Millionen Fußballtrainer vor dem Rechner wollen mit Zahlenmaterial versorgt werden. Sie programmieren deswegen in dieser Aufgabe ein Programm, mit dem Bundesliga-Spielergebnisse in Form von Tabellen auswertbar sind.

Anders als bei bisherigen Aufgaben erhalten Sie nicht nur *JUnit-Tests*, sondern auch die Klassen `Team`, `League`, `Game`, `Table` und `TableEntry`. In den Klassen finden Sie Anweisungen, was zu tun ist. Die Anweisungen werden immer innerhalb eines Kommentars mit `TODO:` eingeleitet.

Importieren Sie dazu das bereitgestellte *Gradle-Projekt* in `SupportMaterial/bundesliga` in Ihre Entwicklungsumgebung. Eine Main-Klasse `BundesligaMain` ist ebenfalls bereits gegeben. Sie enthält alle Spiele der Bundesliga der vergangenen Saison.

Die Klassen im Package `bundesliga` hängen folgendermaßen zusammen:

- Die Klasse `Team` repräsentiert ein Team einer Liga. Die Klasse enthält eine statische `Map`, die alle bereits erzeugten Teams enthält. Die Map wird immer vom Konstruktor um das aktuelle Objekt ergänzt. Ein Team kann grundsätzlich auch in mehreren Ligen spielen (zum Beispiel Bundeliga und Champions League). Ein Team ist also nicht auf eine Liga festgelegt.
- Die Klasse `League` repräsentiert eine Liga (-saison). Auch Sie enthält eine statische Map, die alle bereits erzeugten Ligen enthält. Auch in dieser Klasse ergänzt der Konstruktor die statische Map um das aktuelle Objekt. Eine Liga verfügt weiterhin über ein `Set` vom Typen `Game`. Das Set muss anhand der natürlchen Ordnung von `Game` sortiert sein.
- Die Klasse `Game` repräsentiert ein Fußballspiel innerhalb einer Liga. Bereits der Konstruktoraufruf von `Game` hinterlegt das Spiel der entsprechenden Liga.
- Die Klasse `Table` repräsentiert eine Tabelle. Eine Tabelle kann für einen bestimmten Spieltag ausgegeben werden. Dann werden nur die Spiele bis einschließlich dieses Spieltags berücksichtigt. Weiterhin können Tabellen auch nur für Heimspiele oder nur für Auswärtsspiele ausgegeben werden.
- In der Tabelle gibt es eine Reihe von Zeilen. In jeder Zeile steht eine Fußballmannschaft mit deren Daten (Punkte, Toredifferenz, etc.). Eine Zeile der Tabelle wird von der Klasse `TableEntry` repräsentiert. Bereits der Konstruktor errechnet alle benötigten Werte. Objekte der Klasse `TableEntry` werden im Konstruktor der Klasse `Table` erzeugt.

Für eine weitere Erklärung der Attribute und Methoden der Klassen lesen Sie bitte die Dokumentation (JavaDoc) und den Quellcode. Vervollständigen Sie die Klassen und testen Sie Ihre Ergebnisse mit den *JUnit-Tests*. Importieren Sie dazu das bereitgestellte *Gradle-Projekt* in `SupportMaterial/bundesliga` in Ihre Entwicklungsumgebung. Eine Main-Klasse `BundesligaMain` ist ebenfalls bereits gegeben. Sie enthält alle Spiele der Bundesliga der vergangenen Saison.