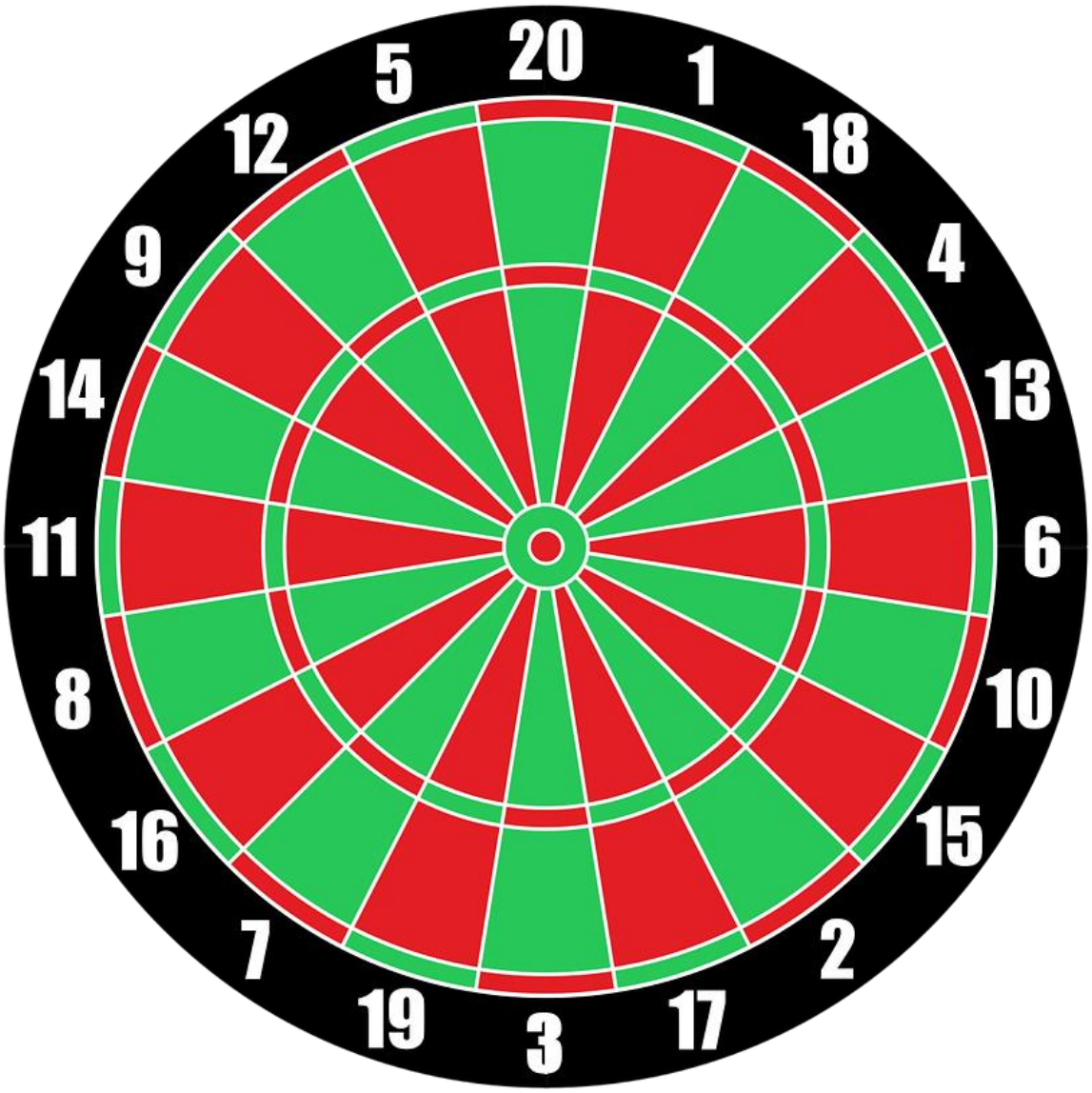

Darts Counter

Hinweis zur Bewertung:

- 1/4 der Punkte (25%) wird nach Funktionstests Ihrer Lösung vergeben.
- 3/4 der Punkte (75%) werden entsprechend des in den Teilaufgaben angegebenen Schlüssels auf Basis des Quellcodes vergeben.

DHBW Karlsruhe, Vorlesung Programmieren I, Probe-Programmentwurf

Bearbeitungszeit: 90 Minuten

Aufgabe

Schreiben Sie eine Java-Anwendung „*DartsCounter*“, die einen Zählautomaten für eine Spielrunde („Leg“) Darts implementiert.

Gespielt wird im klassischen Modus: 501 Punkte müssen entsprechend der Punktewertigkeit der getroffenen Felder des Dartboards genau auf 0 gebracht werden. Es sollen immer genau 2 Spieler abwechselnd ihre Würfe eingeben können.

.

Teilaufgabe a) **[5%]**

Um ein Dartboard in Java abzubilden, implementieren Sie zunächst die Klasse `Field`. Jedes Feld besteht aus einem Label (`String label`) (Bezeichnung des Feldes, s. unten), einer Punktewertigkeit (`int value`) sowie einem Wahrheitswert, welcher besagt ob dieses Feld ein „Doppelfeld“ ist (`boolean doubleField`).

Ermöglichen Sie durch einen geeigneten Konstruktor das Setzen sämtlicher Attribute bei der Objekterzeugung sowie das Auslesen der Werte mit „Getter“-Methoden. Alle Attribute sind mit dem Modifikator `private` zu deklarieren!

Teilaufgabe b) **[10%]**

Erstellen Sie anschließend eine Klasse zur Repräsentierung des gesamten Dartboards (`Board`). Ein Dartboard besteht aus 62 unterschiedlichen Feldern:

- Für die Zahlen 1 – 20 je ein Feld für einfache, zweifache und dreifache Wertigkeit der betreffenden Zahl.
  - Label von Feldern mit einfacher Wertigkeit: *Zahlenwert* (z.B. „5“ für einfache 5)
  - Label von Feldern mit zweifacher Wertigkeit: D*Zahlenwert* (z.B. „D16“ für doppelte 16)
  - Label von Feldern mit dreifacher („Triple“) Wertigkeit: T*Zahlenwert* (z.B. „T20“ für dreifache 20)
- Ein Feld für den äußeren Ring in der Mitte („Single Bull“, 25 Punkte, Label: „25“)
- Ein Feld für die Mitte („Bullseye“, 50 Punkte, Label: „BULL“)

Die einfachen Felder der Zahlen von 1-20 gibt es genau genommen zweimal auf dem Board, dies ist für die Aufgabenstellung jedoch zu vernachlässigen.

Erzeugen Sie die `Field`-Instanzen automatisch bei der Erzeugung eines `Board`-Objekts und speichern Sie diese in einem Array. Beachten Sie, dass sämtliche Felder mit *zweifacher Wertigkeit* sowie das *Bullseye* als Felder mit der Eigenschaft „Doppelfeld“ gelten. Darüber hinaus soll ein „Hilfsfeld“ mit dem Label „x“ und der Wertigkeit 0 für Treffer neben dem Board definiert werden.

Um im späteren Verlauf eine Eingabe verarbeiten zu können, erweitern Sie die Klasse `Board` um die Instanzmethode

```
public Field parseField( String label )
```

Diese soll die `Field`-Instanz zurückliefern, deren Label mit dem Parameter übereinstimmt (Groß- und Kleinschreibung spielen hierbei keine Rolle). Wird keine Übereinstimmung gefunden, soll ein `null`-Wert zurückgegeben werden.

Teilaufgabe c) **[10%]**

Entwickeln Sie darüber hinaus eine Klasse `Visit`, welche einen Wurfdurchgang eines Spielers („Besuch am Dartboard“) mit maximal 3 Würfen repräsentiert.

Speichern Sie die getroffenen Felder als private Instanzvariable (`Field[] fields`) und ermöglichen Sie die Übergabe im Konstruktor. Prüfen Sie hier auch, ob die Anzahl der geworfenen Darts (Dartpfeile) drei Darts nicht übersteigt. Falls doch, soll eine `IllegalArgumentException` ausgelöst werden!

Die Klasse soll folgende Instanzmethoden realisieren:

- `public Field[] getFields()`: liefert alle getroffenen Felder des Visits
- `public int getValue()`: liefert den aufsummierten Punktewert aller getroffenen Felder des Visits
- `public Field getLastField()`: liefert die Instanz des zuletzt getroffenen Feldes

Teilaufgabe d) **[10%]**

Ein Dartspiel kann jedoch nicht ohne Spieler funktionieren. Realisieren Sie eine Klasse `Player`, welche zunächst den Namen des Spielers (`String name`) als Attribut besitzt und diesen mit Hilfe des Konstruktors setzen kann.

Darüber hinaus sollen eine Zählvariable für die geworfenen Darts (`int countDartsThrown`) und ein Feld für maximal 10 Wurfdurchgänge (`Visit[] visits`) realisiert werden.

Sämtliche Attribute sind auch hier mit dem **`private`** -Modifikator zu versehen und – sofern nötig – über „Getter"-Methoden von außen zugänglich zu machen.

Ein Spieler-Objekt soll folgende Instanz-Methoden bereitstellen:

- **`public int`** `getRemainingPoints()`: soll die Punktzahl zurückliefern, die der Spieler aktuell noch benötigt, um die anfangs 501 Punkte auf 0 zu bringen.
- **`public boolean`** `addVisit( Visit visit )`: Fügt einen Wurfdurchgang zum aktuellen Spieler hinzu. Hierbei gilt zu beachten:
  - Ein Wurfdurchgang wird nur dann hinzugefügt, wenn die aktuelle Restpunktzahl abzüglich des Durchgangs *nicht kleiner als 0* ist („Überworfen")
  - Die Anzahl der geworfenen Darts wird in jedem Fall um die Anzahl der Felder im Wurfdurchgang erhöht
  - Der Rückgabewert signalisiert, ob ein Wurfdurchgang hinzugefügt wurde (**`true`**) oder nicht (**`false`**)

Hinweis: In Teilaufgabe g) wird diese Funktionalität erweitert!

Teilaufgabe e) **[20%]**

Implementieren Sie nun eine Klasse `Game`, welche eine Spielrunde („Leg") realisiert. Der Konstruktor der Klasse `Game` soll zunächst eine Instanz eines Dartboards sowie ein Array von Spielern entgegennehmen und in Instanzvariablen speichern (*vgl. `main`-Methode in Hinweisen*). Durch Aufruf ihrer zu definierenden Methode

```
public void start()
```

wird das Spiel gestartet. Solange das Spiel nicht beendet ist (also keiner der Spieler bei genau 0 Punkten gelandet ist und die maximal 10 Wurfdurchgänge nicht erreicht sind) soll abwechselnd für jeden Spieler von der Kommandozeile (bzw. dem Konsolenfenster) der aktuelle Wurfdurchgang eingelesen werden. Jede Feldangabe wird hierbei durch ein Leerzeichen von der nächsten getrennt (*Hinweise beachten!*).

Vor jeder Wurfeingabe soll der aktuelle Spieler sowie dessen Restpunktzahl ausgegeben werden. Nach der Wurfeingabe soll die geworfene Punktzahl ausgegeben werden.

Ist das Spiel beendet, soll entweder
- der siegreiche Spieler ausgegeben werden oder
- eine Nachricht angezeigt werden, dass das Spiel wegen Überschreitung der Maximalzahl der Wurfdurchgänge vorbei ist.

Zur Verdeutlichung (Orange = Benutzereingabe):

Beispielausgabe für erfolgreiches Spiel

```
Player: Michael van Gerwen, 501 points remaining.
Enter visit: T20 T20 T20
Scored: 180
======================
Player: Rob Cross, 501 points remaining.
Enter visit: T20 T20 T20
Scored: 180
======================
Player: Michael van Gerwen, 321 points remaining.
Enter visit: T20 T20 T20
Scored: 180
======================
Player: Rob Cross, 321 points remaining.
Enter visit: T20 T20 T20
Scored: 180
======================
Player: Michael van Gerwen, 141 points remaining.
Enter visit: T20 T19 D12
Scored: 141
======================

Game shot and the leg, Michael van Gerwen!
```

Beispielausgabe für nicht erfolgreiches Spiel

```
Player: Michael van Gerwen, 501 points remaining.
Enter visit: 1 1 1
Scored: 3
======================
Player: Rob Cross, 501 points remaining.
Enter visit: 1 1 1
Scored: 3
======================

( ... and so on ... )

======================
Player: Michael van Gerwen, 474 points remaining.
Enter visit: 1 1 1
Scored: 3
======================
Player: Rob Cross, 474 points remaining.
Enter visit: 1 1 1
Scored: 3
======================

You're too bad for this game!
```

Teilaufgabe f) **[5%]**

Erweitern Sie die Klasse `Game` so, dass nach erfolgreichem Spiel eine Art High-Score in einer Textdatei (`highscore.txt`) gespeichert wird. Schreiben Sie hierzu eine Zeile mit dem *Namen des Spielers* sowie die *Anzahl der benötigen Würfe* in die Datei, bspw.:

```
Michael van Gerwen won with 9 darts.
```

Sollte die Datei bereits bestehen, soll die Zeile am Ende der Datei angehängt werden.

Teilaufgabe g) **[5%]**

Aktuell kann mit jedem getroffenen Feld ein Spieldurchgang beendet werden, dessen Punktzahl der Restpunktzahl entspricht. Im klassischen Spiel ist dies jedoch nur mit dem Wurf auf ein Doppelfeld möglich.

Erweitern Sie die Methode `addVisit` der Klasse `Player` so, dass Wurfdurchgänge nur noch dann angenommen werden, wenn:

- im Falle, dass die Punktzahl nach dem Wurfdurchgang genau auf 0 gespielt wurde, das letzte getroffene Feld ein Feld mit der Eigenschaft „Doppelfeld“ war,
- die Restpunktzahl nach dem Durchgang nicht 1 beträgt (Rest 1 ist nicht möglich bei „Double Out“)

Teilaufgabe h) **[10%]**

Erweitern Sie ihr Programm so, dass bei einer Restpunktzahl von 170 Punkten oder weniger ein möglicher „Checkout“-Weg angezeigt wird, d.h. eine mögliche Kombination von Treffern, welche die Punktzahl mit abschließendem Wurf auf ein Doppelfeld auf 0 bringen.

Lesen Sie hierfür die bereitgestellte Datei „`checkouts.txt`“ ein. In dieser Datei steht pro Zeile

- entweder ein Minus, wenn kein Checkout mit abschließendem Doppelfeld möglich ist (bspw. 1)
- oder ein möglicher Checkout-Weg für die jeweilige Restpunktzahl (passend zur Zeile).

Dabei steht der Checkout-Weg in Zeile 1 für die Restpunktzahl 1, in Zeile 2 für die Restpunktzahl 2, in Zeile 3 für die Restpunktzahl 3, etc.

Hinweise:

Starten:

Starten Sie die Anwendung mit einer Klasse `DartsCounter` (siehe Download), die etwa so aussieht:

```
public class DartsCounter {
	public static void main( String[] args ) {
		final Board b = new Board();

		final Player[] players = new Player[] {
				new Player( "Michael van Gerwen" ),
				new Player( "Rob Cross" )
		};
		final Game g = new Game( b, players );
		g.start();
	}
}
```

Auftrennen an Leerzeichen:

Um eine Zeichenkette an Leerzeichen in ein Array aufzusplitten, können Sie die Instanzmethode *`stringVar`*`.split(...)` verwenden:

```
String foo = "T20 T19 T18";
String[] bar = foo.split( " " ); // liefert Array mit 3 Einträgen: {"T20","T19","T18"}
```

Fehler bei der Eingabe:

Zur Vereinfachung dürfen Sie davon ausgehen, dass nur valide Eingaben erfolgen.