HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN
WÜRZBURG-SCHWEINFURT

Programmieren II
Sommersemester 2016

Übungsblatt 11

Iterator

Der Begriff „Iterator“ wurde als Entwurfsmuster (wie Decorator, Singleton, etc.) von Erich Gamma et. al. konkretisiert: Er ermöglicht „den sequenziellen Zugriff auf die Elemente eines zusammengesetzten Objekts, ohne seine zugrundeliegende Repräsentation offenzulegen.“[1] Die wiederholte (iterierte) Nutzung eines Iterators ergibt also eine Folge von Elementen, die entweder in einer Datenstruktur gespeichert sind (z.B. in einem Array), oder sukzessive berechnet werden. Im letzteren Fall kann es sein, dass nie die Gesamtheit aller Elemente im Speicher gehalten, sondern immer nur das jeweils aktuelle nächste Element vom Iterator ermittelt wird.

In Java wird ein Iterator durch das Interface `java.util.Iterator<E>` beschrieben. Es umfasst insgesamt drei Methoden, wobei die letzte Methode in der API-Dokumentation als „optional“ bezeichnet wird:

- `boolean hasNext()`
  Diese Methode ergibt `true`, falls der Iterator in der Lage ist, noch weitere Elemente auszuliefern (mindestens eines).
- `E next()`
  Diese Methode liefert das nächste Element zurück.
- `void remove()`
  Diese Methode entfernt das zuletzt zurückgelieferte Element aus der Folge. Diese Methode ist nicht immer realisierbar – insbesondere dann nicht, wenn die Elemente jeweils neu berechnet werden müssen. Daher bezeichnet die API diese Methode als optional.
  Die Java-Syntax erzwingt aber in jedem Fall, dass diese Methode implementiert wird. Durch das Werfen einer `UnsupportedOperationException` wird signalisiert, dass diese Methode beim vorliegenden Iterator nicht sinnvoll ist.

Das Durchlaufen einer `Collection` ist also nur eine Einsatzmöglichkeit des Interfaces `Iterator<E>`. In der folgenden Übung soll daher ein Iterator implementiert werden, der nicht auf einer `Collection` basiert.

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Entwurfsmuster, Addison-Wesley, 1996

Aufgabe „exercise11“

Beim wiederholten Werfen eines normalen sechsseitigen Würfels entsteht eine Folge von ganzzahligen Zufallszahlen aus dem Intervall [1; 6]. Diese (potenziell unendliche) Folge soll mit Hilfe eines Iterators (teilweise) durchlaufen werden.

Ihre Aufgabe:

Im Verzeichnis `exercise11` finden Sie die abstrakte Klasse `AbstractDie`, die als Oberklasse für eine von Ihnen zu implementierende konkrete Klasse `Die` (Würfel) dienen soll. Zu implementieren ist in `Die` in jedem Fall die abstrakte Methode

```
public abstract Iterator<Integer> iterator()
```

die ein Iterator-Objekt für die Folge von Zufallszahlen zurückliefern muss. Sie müssen hierzu eine weitere Klasse mit einem Namen Ihrer Wahl implementieren, die das Interface `Iterator<Integer>` erfüllt.

Die in der abstrakten Klasse `AbstractDie` implementierte Klassenmethode

```
public static void test(AbstractDie aDie)
```

kann z.B. in der `main`-Funktion der (Unter-)Klasse `Die` benutzt werden, um die korrekte Funktion des Iterators zu überprüfen:

```
public static void main(String[] args)
{
    test(new Die());
}
```

Hinweise:

- `Math.random()` liefert eine `double` Zufallszahl aus dem Intervall [0;1[
- Die Folge der Zufallszahlen ist potenziell unendlich und kann daher nicht im Speicher gehalten werden. Berücksichtigen Sie dies bei der Implementierung von `hasNext()` und `next()`.
- Aus diesem Grund ist die Implementierung der „optionalen“ Methode `remove()` nicht möglich. Lösen Sie das Problem wie in der API vorgesehen.