TU Berlin, Fakultät IV
Blankertz/Röhr/Stahl
MAR 4-3 und MAR 5-1

Aufgabenblatt 2

Java Einführung II

Abgabe (bis 07.05.2019 19:59 Uhr)

Die folgenden Dateien müssen für eine erfolgreiche Abgabe im git Ordner eingecheckt sein:

Geforderte Dateien:	
`Blatt02/src/Item.java`	Aufgabe 1.1
`Blatt02/src/KitchenItem.java`	Aufgabe 1.2
`Blatt02/src/ManagetheChaos.java`	Aufgabe 1.3 bis 1.6

Als Abgabe wird jeweils nur die letzte Version im git gewertet.

Aufgabe 1: Collections (Hausaufgabe)

In dieser Aufgabe werden Sie mit Datenstrukturen, copy constructors und Java Collections arbeiten. Sie beschäftigen sich mit dem Aufräumen einer kleinen WG-Küche.
Verdeutlichen Sie sich die Aufgabe an folgendem einfachen Beispiel: Sie haben 5 Tassen, Sie brauchen 3. Sie haben eine Abstellkammer in die Sie Dinge, die Sie nicht brauchen, stellen können. In diesem Fall 2 Tassen. Wenn Ihnen dann eine Tasse kaputt geht, haben Sie noch eine in der Abstellkammer. Sie können diese also ersetzen. Wenn Sie keine Tassen haben, brauchen Sie diese auch nicht aufräumen. Ihre Abstellkammer ist für diese Aufgabe ein Stack. Sie ist aber nicht unbedingt sinvoll gepackt, d.h. es könnte Freiräume geben, in denen nichts steht, obwohl danach noch weitere Sachen reingestellt wurden. Diese Freiräume sind mit `null` belegt.

1.1 Implementation der Klasse `Item` (10 Punkte)

Implementieren Sie die Klasse `Item` gemäß der folgenden API:

public class Item implements Comparable <Item>		
	`Item(String name, String ownersName)`	Erzeugt `Item` Objekt, ein Gegenstand mit einem Namen und Besitzer.
`String`	`getOwner()`	Gibt den Besitzer zurück.
`void`	`setOwner(String name)`	Setzt den Besitzer.
`int`	`compareTo(Item o)`	Gibt zurück welches der beiden Objekte größer ist. Der Vergleich soll anhand der Namen der Besitzer durchgeführt werden.
`String`	`toString()`	Gibt das Item als String dieser Form zurück `owner: item`.
`boolean`	`equals(Object obj)`	Gibt zurück, ob zwei Objekte gleich sind. Hier sind sie für `null` nicht gleich und sonst gleich, wenn die Objektnamen übereinstimmen.
`int`	`hashCode()`	Ist bereits implementiert und gibt den Hashcode zurück.

1.2 Implementation der Klasse `KitchenItem` (20 Punkte)

Implementieren Sie die Konstruktoren einschließlich des copy constructors der Klasse `KitchenItem`.

public class KitchenItem extends Item	
`KitchenItem(String name, int n)`	Erzeugt `KitchenItem` Objekt, ein Gegenstand mit einem Namen und einer gegebenen Gesamtanzahl. Der Besitzer ist in diesem Fall `shared` und das Item nicht in der Abstellkammer.
`KitchenItem(String name, int n, String ownersName)`	Erzeugt `KitchenItem` Objekt, ein Gegenstand mit einem Namen, einem Besitzer und einer gegebenen Gesamtanzahl. Das Item ist nicht in der Abstellkammer.
`public KitchenItem(KitchenItem thing)`	Copy Constructor für KitchenItem.

1.3 Potenziellen Ersatz finden (15 Punkte)

Implementieren Sie die Methode

```
public List <Item> findSpares()
```

der Klasse `ManagetheChaos`. Die Methode überprüft wie viele von jedem Item da sind und wieviele gebraucht werden. Sie gibt eine Liste mit allen KitchenItems, die gerade nicht gebraucht werden zurück. Sie müssen für den Umgang mit der Liste `allItems` nicht beachten, wie viele Dinge tatsächlich in der Liste sind, sondern nur, wie viele als Gesamtanzahl bzw. Anzahl der gebrauchten Items für das Objekt angegeben sind. Die nicht gebrauchten Items werden dann in der entsprechenden Anzahl der Liste, die Sie zurückgeben sollen, hinzugefügt.

1.4 Wegräumen I (10 Punkte)

Implementieren Sie die Methode

```
public void putAway()
```

der Klasse `ManagetheChaos`. Nutzen Sie hierfür die vorherige Teilaufgabe. Diese Methode räumt die nicht gebrauchten KitchenItems in die Abstellkammer. Dies passiert auf die faule Weise: es wird einfach vorne rangestellt, ohne die Kammer vorher auszuräumen. Dabei muss bei dem entsprechenden Item die Variable `intheCubby` neu gesetzt werden und der Stack `cubby` sortiert gefüllt werden, sodass die Items von einer Person zusammenstehen. Die Liste `allItems` sollte sich dabei nicht verändern. Werfen Sie außerdem eine `RuntimeException`, wenn der Cubby zu voll werden sollte.

1.5 Wegräumen II (20 Punkte)

Implementieren Sie die Methode

```
public void putAwaySmart()
```

der Klasse `ManagetheChaos`. Nutzen Sie hierfür die Teilaufgabe 1.3. Diese Methode räumt die nicht gebrauchten KitchenItems in die Abstellkammer. Dabei muss bei dem entsprechenden Item die Variable `intheCubby` neu gesetzt werden und der Stack `cubby` sortiert gefüllt werden, sodass die Items von einer Person zusammenstehen. Diesmal wird die Kammer allerdings vorher ausgeräumt und sollte irgendwo Platz (`null`) sein, wird dort schon einmal das erste Item der sortierten Liste eingefügt. Die Liste allItems sollte sich dabei nicht verändern. Werfen Sie außerdem eine `RuntimeException`, wenn der Cubby zu voll werden sollte.

1.6 Ersetzen (15 Punkte)
Implementieren Sie die Methoden

```
public boolean replaceable(Item item)
public Item replace(Item item)
```

der Klasse `ManagetheChaos`. `replaceable` überprüft, ob ein Item ersetzt werden kann. Die Methode `replace` gibt ein Ersatz aus der Abstellkammer zurück. Wenn `replace` `null` ersetzen soll, sollte eine `RuntimeException` geworfen werden. Außerdem wirft die Methode eine `RuntimeException`, wenn der Gegenstand nicht ersetzt werden kann.
Es ist Ihnen freigestellt, ob sie an der Stelle, wo vorher das Ersatzteil stand einen Freiraum lassen (`null`) und dann alles so einräumen wie es vorher war, oder ob den freien Platz nutzen, das nächste Item reinzustellen.

Hinweis: Schauen sie sich noch einmal die `equals()` Methode an und nutzen sie ggf. `==`.

Aufgabe 2: Design von Datentypen (Tut)

2.1 Betrachten Sie die folgende Klasse:

```
public class StringGenome {
    private String s = "";


    public void addNucleotide(char c) {
        if (c == 'A'  c == 'C'  c == 'G'  c == 'T')
            s = s + c;
        else throw new RuntimeException("Illegal nucleotide");
    }

    public char nucleotideAt(int i) {
        if (i < s.length()) return s.charAt(i);
        else throw new RuntimeException("Genome out of bounds");
    }

    public int length()      { return s.length(); }
    }
```

Was implementiert diese Klasse? Was machen die einzelnen Methoden?

2.2 Wie könnte man eine `main` Methode implementieren, um die Klasse zu testen?

2.3 Was wird Ihnen ausgegeben, wenn Sie ein `StringGenome` mit `System.out.println()` ausgeben? Was könnte diese Ausgabe verbessern?

2.4 Wenn Sie zwei Objekte `genome1` und `genome2` von Typ `StringGenome` erzeugen und beiden das Nucleotid `A` hinzufügen, was gibt Ihnen die equals-Methode zurück? Welche equals-Methode wird benutzt?

2.5 Wie implementieren Sie eine eigene `equals`-Methode?

2.6 Wenn Sie nun z.B. eine `LinkedList samples` aus verschiedenen Genomen erstellen. Was muss die `equals`-Methode erfüllen, damit contains korrekt ausgeführt wird?

2.7 Kann `samples` derzeit mittels `Collections` sortiert werden? Muss die Klasse ggf. verändert werden, damit das funktioniert? und wenn ja, wie?

2.8 Was ist ein copy constructor und was sind die Vorteile davon einen zu haben?

2.9 Nennen Sie einige Faktoren, die für das Design einer neuen Klasse relevant sind.

Aufgabe 3: Laufzeit kleiner Programmabschnitte (Tut)

Bestimmen Sie die Wachstumsordnung der Laufzeit von den in Listing 1 angegebenen Funktionen `f1` bis `f6`.

Listing 1: Java Snippets zur Laufzeitbestimmung

```java
public static int f1(int N) {
  int x = 0;
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      x++;
  return x;
}
```

(a) Funktion `f1`

```java
public static int f2(int N) {
  int x = 0;
  for (int i = 0; i < N/2; i++)
    for (int j = 0; j < i; j++)
      x++;
  return x;
}
```

(b) Funktion `f2`

```java
public static int f3(int N) {
    if (N == 0)
        return 1;
    else
        return 2 * f3(N -1) + 1;
}
```

(c) Funktion `f3`

```java
public static int f4(int N) {
  int x = 0;
  for (int i = 1; i < N; i *= 2)
    x += f3(i);    // f3!
  return x;
}
```

(e) Funktion `f4`

```java
public static int f5(int N) {
    if (N <= 1)
        return N+1;
    else
        return f5(N-2) / f5(N-1);
}
```

(d) Funktion `f5`

```java
public static int f6(int N) {
  if (N == 0)
    return 1;
  return 2 * f6(N/2);
}
```

(f) Funktion `f6`