Prof. Dr. Christoph Bockisch
MSc Steffen Dick
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Tutorial exercises
Objektorientierte Programmierung: Wintersemester 2021/2022
Nr. 8

Task 8.1: Welcome to the Aperture Science computer-aided enrichment center

Look at the following code that computes the cubic root of a given number (the formula could trigger Déjà-vu):

```
1  public static double cubicRoot(double number, double delta){
2    return cubicRootHelp(number, delta, number);
3  }
4
5  private static double cubicRootHelp(double number, double
    ↪  delta, double x_n){
6    double x_nPlusOne = 1.0 / 3.0 * (2 * x_n + number /
      ↪  Math.pow(x_n, 2));
7    if(Math.abs(x_nPlusOne - x_n) < delta){
8      return x_nPlusOne;
9    }
10   return cubicRootHelp(number, delta, x_nPlusOne);
11 }
```

Write a sufficient number of meaningful JUnit-Tests for `cubicRoot` in which positive and negative numbers are tested.

Task 8.2: It's over 9000!!!!!

Start a new project in IntelliJ and add the following classes:

```
1 public class Sayan{
2   private int basePowerLevel
     ↪  = 10;
3   private int transformation
     ↪  = 0;
4   final String name;
5
6   public Sayan(String name){
7     this.name = name;
8   }
9 }
```

```
1 public class Namekian{
2   private int basePowerLevel
     ↪  = 8;
3   private int transformation
     ↪  = 0;
4   final String name;
5
6   public Namekian(String
     ↪  name){
7     this.name = name;
8   }
9 }
```

```
1 public class Human{
2   private int basePowerLevel = 5;
3   final String name;
4
5   public Human(String name){
6     this.name = name;
7   }
8 }
```

a) Create an interface `Fighter`. This interface should require the methods `void train(int hours)` and `int getPowerLevel()`. Furthermore, implement the default-method `default void gravityTrain(int hours, int gForce)` that should call `train` whereas `hours` is multiplied with `gForce`.

b) Change the classes in a way that all 3 implement the interface `Fighter`.

Attention: IntelliJ should notify you that your code won't work because your classes do not implement the required methods. Klick on the name of your class within the source code and press *alt* and *Enter* keys at the same time. Click on *Implement Methods* in the menu that pops up. IntelliJ will now generate so called stubs for each missing method.

c) Implement the added methods.

A human should get `hours` added to their `basePowerLevel` when training. The `basePowerLevel` should also be returned when `getPowerLevel` is called.

A namekian's `basePowerLevel` should increase by double the `hours` when training. The powerlevel is computed by using `basePowerLevel * (transformation * 1.8 + 1)`.

A sayan even gains triple `hours` to their `basePowerLevel` when training. Their powerlevel is computed by using `basePowerLevel * (transformation * 2 + 1)`.

d) Add `void namekianAbsorb(int i)` to `Namekian` that sets `transformation` to *i*. Furthermore, create the methods `void supaSayajin(int i)` that sets `transformation` to `i` whereas `i` can be between 1 and 4 and `void`

`supaSayajinGoto()` that sets `transform` to `666` within `Sayan`. Implement **`void`** `powerdown()` that sets `transformation` to `0` in both classes.

e) Test your methods using JUnit-tests.

f) `Sayan` should now also implement the interface `Comparable`. Implement the missing method in such a way that objects are sorted by using `getPowerLevel()`.

g) Create 4 different objects of `Sayan` in a main-method: *Son Goku*, *Vegeta*, *Nappa* and *Radditz*. Change their powerlevel by training or transforming and add them (unsorted) into an array. Iterate over that array using a for-loop and print their names to the console. Use `Arrays.sort` to sort the array and print the names to the console again.