

Practice examples for Programming

Max Schik

Way too early in the morning

Contents

Small disclaimer

Please don't make syntax mistakes. Don't miss a semicolon or a missing "this" or something like "for (int i = 0; i \leq 420; i++)". You can use `System.out.println()` for printing out stuff. Don't use any packages that have to be imported. Follow the KIT styleguide.

1 Idiot test

1.1 Basic java syntax

Write down a main function. It should create a variable named "funny" for a byte sized number and give it the value 69. Also add an array named "bird" (dt. Feld, warum auch immer) of single characters that holds only the values 'F', 'o', 'g', 'e', 'l'. Print out the 3rd element of the array.

```
// Idiot test
```



1.2 Mandatory interface exercise

Add the method "endMe" to the interface. The method gets a string "please" as a parameter and returns an Object of the type "Attention".

```
interface CryForHelp {  
      
}
```

1.3 Basic class syntax

Add a private string of name "name" to the class. Also add a public constructor that takes a string as an argument and creates a class with the attribute "name" set to this string. Add

getter and setter for name and write a Java-style comment(!) above it why you should use getter and setters.

```
public class Idiot {
```

```
    [redacted]
```

```
    [redacted]
```

```
// getter
```

```
    [redacted]
```

```
// setter
```

```
    [redacted]
```

```
}
```

2 Flow control

2.1 Fruity loopy

Expand the method to print out every third element starting with the 3rd element

```
public static void thirdElements(int[] someArray) {  
    for ([redacted]) {
```

```
        [redacted]
```

```
    }
```

```
}
```

2.2 Lowest number


Expand the method to return the lowest number from an array of ints.

```
public static [redacted] getLowestNumber(int[] numbers) {
```

```
}
```

2.3 Count characters

Expand the method to return the amount of a given character in an given string. You can use `string.split("")` to split the string to an string array. The character is also given as a string.

```
public static  countCharacters(String character, String string) {
```

```
}
```

2.4 Find shared numbers

Given two integer arrays, find all shared numbers (numbers that are in both arrays) and print them. Not on a piece of paper but on the command line. You fucking small brain. You can expect that every number is unique in its array. Expand the method accordingly.

```
public static  sharedNumbers(, ) {
```

```
}
```

3 Object oriented plitsch platsch

3.1 Some basic questions

What does the operator *static* mean? What's special about static functions?



Name three differences between *Abstract class* and *interface*:



Who can access a class function with the modifier *protected*?



What do you write in a Javadoc-style comment for a method?



3.2 Inheritance


Given the class *Bird*, inherit a new class from it with the name *Lovebird* and override the *fly* method so that it returns "Cute flying". Also, add a public constructor and a new public function named *beCute* that returns nothing, takes no arguments and has an empty body.

```
// Bird class
public class Bird {
    private Color color;

    public Bird(Color color) {
        this.color = color;
    }

    public String fly() {
        return "Just flying";
    }
}

// Lovebird class
```



3.3 Polymorphie

First, checkout those awesome classes and functions I wrote.

```
public class Dog {
    /** .... */

    public Dog(** ... */) {
        /** ... */
    }

    public String bark() {
        return "Woof";
    }
}

public class ScaredDog extends Dog {
    public ScaredDog(** ... */) {
        /** ... */
    }

    @Override
    public String bark() {
        return "scared wuf"
    }

    public String runAway() {
        return "*sound of dog running away*"
    }
}

public class OldDog extends Dog {
    public OldDog(** ... */) {
        /** ... */
    }
}
```

Write down the given function calls output. If it would throw an error, write down "Error" and try to explain why it would throw an error.

```
ScaredDog ike = new ScaredDog();
Dog wufwuf = new ScaredDog();
OldDog ronja = new OldDog();
```

```
// write down the given output;
```

```
ike.runAway();
```

```
wufwuf.runAway();
```

```
wufwuf.bark();
```

```
ronja.bark();
```

```
ike.runAway();
```

4 Find the errors

Mistakes were made. People were made. Some people are mistakes. But that is non of your business, so now you search for mistakes of any kind (except grammar cause I suck at it). That means logic errors, syntax errors or violations against the styleguide (the worst kind of mistake in my humble opinion).

4.1 Treshhold function

```
/**
 * this method prints all numbers from an array that are bigger
 * than the treshhold to the command line
 *
 * @param inputArray the given array
 * @param number the treshhold
 */
public static String Treshhold(inputArray, void number) {
    for (int i; i <= inputArray.length; i++) {
        if (inputArray[i] < number) {
            Sytem.out.println(inputArray[i])
        }
    }
}
```

4.2 Concatenate a string array

```
/**
 * adds all strings from an array together
 * @param Strings the given array of strings
 * @return the concatenated string
 */
public static concatenateString(String[] Strings) {
    output = ""
    for (String strings : Strings) {
        output += strings;
    }
    return output;
}
```

4.3 Some class-stuff

```
public class VideoGame {
    public int hours;

    public VideGame() {
        hours = hours;
    }

    private getName() {
        return this.name;
    }
}

VideoGame.setHours(250)
VideoGame.getName();
```

5 Solutions

1.1

```
// Idiot test
public static void main(String[] args) {
    byte funny = 69;
    char[] bird = new char[]{'F', 'o', 'g', 'e', 'l'};

    System.out.println(bird[2]);
}
```

1.2

```
interface CryForHelp {
    public Attention endMe(String please);
}
```

1.3

```
public class Idiot {
    private String name;

    public Idiot(String name) {
        this.name = name;
    }

    // we need getters and setters for security reasons and data encapsulation

    // getter
    public String getName() {
        return this.name;
    }

    // setter
    public void setName(String name) {
        this.name = name;
    }
}
```

2.1

```
public static void thirdElements(int[] someArray) {
    for (int i = 2; i < someArray.length; i+=3) {
        System.out.println(someArray[i]);
    }
}
```

2.2

```
public static int getLowestNumber(int[] numbers) {
    int lowestNumber = numbers[0];

    for (int i = 1; i < numbers.length; i++) {
        if (numbers[i] < lowestNumber) {
            lowestNumber = numbers[i];
        }
    }

    return lowestNumber;
}
```

2.3

```
public static int countCharacters(String character, String string) {
    int count = 0;
    String[] stringArray = string.split("");

    for (String stringChar : stringArray) {
        if (stringChar.equals(character)) {
            count++;
        }
    }

    return count;
}
```

2.4

```
public static void sharedNumbers(int[] firstArray, int[] secondArray) {
    for (int i = 0; i < firstArray.length; i++) {
        for (int j = 0; j < secondArray.length; j++) {
            if (firstArray[i] == secondArray[j]) {
                System.out.println(firstArray[i]);
            }
        }
    }
}
```

3.1

What does the operator *static* mean? What's special about static functions?

The static operator on a technical level means that the variable/attribute gets initialized on compile time, not on runtime. That means its the same for every instance of an class (so for every object its the same) and normally gets referenced by Class.name. A static functions doesn't belong to an object, but to a class.

Name three differences between *Abstract class* and *interface*:

Just three examples:

Abstract class gets inherited, not implemented.

Abstract class allow for implementations of methods, not just declaration.

Interfaces can only have public methods, Abstract class allows for every modifier.

Who can access a class function with the modifier *protected*?

Only the class itself and classes inherited from that class.

What do you write in a Javadoc-style comment for a method?

Description of the method, paramter description, possible excpetions and expcetion cases, what the method returns.

3.2

```
// Bird class
public class Bird {
    private Color color;

    public Bird(Color color) {
        this.color = color;
    }

    public String fly() {
        return "Just flying";
    }
}

// Lovebird class

public class Lovebird extends Bird {
    public Lovebird(Color color) {
        super(color);
    }

    @Override
    public String fly() {
        return "Cute flying";
    }

    public void beCute() {
    }
}
```

3.3

```
ScaredDog ike = new ScaredDog();
Dog wufwuf = new ScaredDog();
OldDog ronja = new OldDog();

// write down the given output;
ike.bark(); // "scared wuf"

wufwuf.runAway(); // Error, wufwuf is of type dog and doesnt have runAway as a
function

wufwuf.bark(); // "scared wuf"

ronja.bark(); // "Woof"

ike.runAway(); // "*sound of dog running away*"
```

4.1

```
/**
 * this method prints all numbers from an array that are bigger
 * than the treshhold to the command line
 *
 * @param inputArray the given array
 * @param number the treshhold
 */
public static void treshhold(int[] inputArray, int number) { // return type
    should be void, treshhold should be lowerCamelCase, inputArray was missing
    its type, number is definitely not void
    for (int i = 0; i < inputArray.length; i++) { // i had no value, <= is wrong
        cause it would get out of bound of the array
        if (inputArray[i] > number) { // < should be >, logic error
            Sytem.out.println(inputArray[i]); // missing semicolon
        }
    }
}
```

4.2

```
/**
 * adds all strings from an array together
 * @param Strings the given array of strings
 * @return the concatenated string
 */
public static String concatenateString(String[] strings) { // missing return
    type, Strings should be lowerCamelCase
    String output = "" // missing type
    for (String singleString : strings) { // can use strings for only one
        variable, so renamed
        output += singleString;
    }
    return output;
}
```

4.3

```
public class VideoGame {
    private int hours; // attributes always private
    private String name; // is needed later

    public VideGame(int hours, String name) { //missing parameters
        this.hours = hours; // missing this
        this.name = name;
    }

    public String getName() { // wrong modifier, missing return type
        return this.name;
    }
}

VideoGame stardewValley = new VideoGame(17, "Stardew Valley"); // we need an
    object
stardewValley.setHours(250) // cant call non-static functions on classes, so we
    need the object
stardewValley.getName(); // same as above
```
