Lesson 1B: Instance variables, instance methods, the built-in class String, and static arrays

Mark Eric Dieckmann

January 16, 2023

Change (refactor) the name of your class **Manipulations** in your package *lesson1a* from **Manipulations** to **ManipulationsA**. Create the package ”lesson1b” in your directory **Lessons**. Place the class **Lesson1B** with the method *main(..)* and a new class **ManipulationsB** in this new package.

1 Instance variables vs class variables

Classes are useful as containers for class methods and variables, which we declared with the keyword *static*. We can access static members (variables and methods) with their class name followed by a **.**

Declare a class variable of type **String** called *species* in the class **ManipulationsB** that is not visible outside the class. Implement the class method *setSpecies(arg)* with no return value. It should be visible outside the class - like all methods unless stated otherwise. Its argument *arg* should be of type **String**. It should set the value of *species* to that of *arg*. Implement a class method *getSpecies()*, which returns the value of *species*.

Variables and methods declared static get loaded into memory during compile time. We can access them before we create any instance of **ManipulationsB**. Call the *setSpecies(arg)* method in *main(..)* in **Lesson1B** with the argument ”Human”. Call in *main(..)* the *getSpecies()* method and write the returned value to the console.

We want to have instances of **ManipulationsB** with different attributes. Add the instance variable *name* of type String to **ManipulationsB**. This is an attribute of an individual human. It should be visible from the outside. Create the instances *human1* and *human2* and set their values of *name* to ”Paul” and ”Astrid”.

Write the values of *name* of *human1* and *human2* to the console (in *main()*).

2 Instance methods

Accessing instance variables from *main()* can create messy code. For example, if the values of variables are connected and changing one requires modifying the second one.

Better: make instance variables of **ManipulationsB** invisible to *main()* and use instance methods to change them. You can specify all the steps needed to change the value of an instance variable in the method.

Add the instance methods *setName(arg)* with no return value and the argument *arg* of type **String**. It should set the value of *name* to *arg*. Add another instance method *getName()* with no argument. It should return the value of *name*.

Can you call the methods *setName(arg)* and *getName()* in *main()* through "ManipulationsB." before you create the instances *human1* and *human2*. Why does this not work? If you don't know the answer, revise the lecture notes on memory management and reference variables (Important).

Write the return values of the *getSpecies()* and *getName()* methods of *human1* and *human2* to the console. What difference between class variables and instance variables do you observe?

Use in *main()* the *setName(arg)* method to swap the values of *name* of *human1* and *human2*. You will need a temporary variable to store one of the names. Verify the correctness by writing the return values of **getName()** to the console before and after the swap.

If we swap names frequently, we should implement methods for that. Let us start with the class method "staticSwap(..)". It is a class method and is called in *main(..)* with "ManipulationsB.staticSwap(..);" How many arguments do you need in the argument list "(..)" to do the swap? Think about how you can do that with an instance method "instanceSwap(..)". How many arguments do you need in this case? Implement that, use the methods and check if they worked.

3 The class String

The class **String** provides us with an elegant way to write strings to the console. Write in *main(..)* the command line *String.* and you get a pop-up menu with the class methods of **String**. Click on the first "String.format" list item and read its description. The formatting string is a combination of your own text and text from other sources. The method returns a formatted string.

```
int days = 365;
String text = String.format("One year has %1d days", days);
System.out.println(text);
```

Run the code and check the output. What happens as you increase the value before the "d"? Take increments of 1 until you start to see a difference on the console.

We can also combine float or double precision values with text. The symbol *d* for integers is replaced by *f* for float/double precision variables. Once *text* is declared, we do not redeclare it. Run the code and check the output.

```
double number = 654.321;
text = String.format("The formatted number %7.3f is %1.1f", number, number);
```

```
System.out.println(text);
```

Increase the number 1 before the full stop in %1.1f in integer steps until you see a change. Now increase the number behind the full stop. Play around with both numbers until you figure out what they do.

We can also combine strings with the control string. Try

```
days = 365;
String myText = "year";
text = String.format("One %4s has %3d days", myText, days);
System.out.println(text);
```

Run the code. Change the number in %4s to 6 and run the code. Change the number to -6 and run the code. What happens to the result?

The class **String** has another useful method. Implement a line with several names in the same long string

```
String severalNames = "Paul Astrid,Thomas  Sybill";
```

and type *severalNames.sp* You will find a split method, which allows you to subdivide the long string at characters specified in *regex*: regular expression. The return value is a static array of strings.

Static array: it has a fixed number of elements that you cannot change after you initialized it. A static array neither has to be declared *static* nor does its content go into the stack.

A reference variable to a static String array is declared as *String[] strarray;*

When you declare the array you do not need to specify its length. You can attach to it any one-dimensional array of Strings. Try

```
String[] strarray = severalNames.split(" ");
System.out.println(strArray.length);
```

and see how many String elements you get in the array. Go through them. Now check

```
strarray = severalNames.split(" +");
System.out.println(strArray.length);
```

and see how many Strings you get. Go through them. Now check

```
strarray = severalNames.split(",");
System.out.println(strArray.length);
```

4 Developing the class

We add constructors to **ManipulationsB**, which initialize variables when the instance is created. When you add a constructor with one or more arguments, you must also specify the default constructor explicitly.

Implement a default constructor, which initializes *name* with the empty String ”” and *species* with ”Human”. It should write the content of both variables to the console.

Implement a second constructor with one argument *arg*. It should initialize *name* with the value of *arg* and *species* with ”Human”. It should write the content of both variables to the console.

Implement a third constructor with two arguments *arg1* and *arg2*. It should initialize *name* with the value of *arg1* and *species* with the value of *arg2*. It should write the content of both variables to the console.

We also want a user-defined formatted response to a *System.out.println(human)* call (*human*: reference variable). The formatted string should write ”Species: *arg1*, Name: *arg2*”. *arg1* is a String with 10 characters and aligned to the left. *arg2* is a String with 10 characters and aligned to the right. Create new instances of *human1* and *human2* and give them the names ”Paul” and ”Astrid” through the constructor with one argument. Call *System.out.println(human1);* in *main()* to test the *toString()* method.

Create an instance *dog* of **ManipulationsB** with the name ”Barky” and species ”Dog” (call the constructor with two arguments). Call *System.out.println(human1);* in *main()* to see what changed.

Define the string

```
String input = "Paul Anki Cesar   Miranda";
```

in *main(..)*. Split the string.

Create a static array of **ManipulationsB** with as many elements as the split string. You send the number of elements *length* into the constructor of the static array:
new ManipulationsB[length];

Initialize the elements of **ManipulationsB** with a loop and take for this purpose the names from *input*. Use a *for loop*, a *do while loop*, and a *while loop*.