TND002 Object-oriented programming

Lecture 3: Developing a class

Mark Eric Dieckmann,
MIT division,
ITN, Linköping University.

January 21, 2023

Outline of the lecture

- Definition of classes and objects in object-oriented programming.
- Instance and class variables.
- Instance and class methods.
- Constructors and the method toString().

Introducing classes

We are familiar with numbers (int, double) and arithmetic operations (+-/*). Operators make sense in this context.

How about a similar framework for example for humans?

People have attributes like eye color, the mother tongue, weight etc. They can speak with each other and respond to questions.

Attributes can be defined by variables, while things humans do can be described by methods.

Object-oriented programming bundles attributes and methods that belong together into a "sub-program" called a class.

The class Human

We create a new class **Human** without a main method.

The file "Human.java" should be placed into the same package as the class with the main method (MainClass.java).

```
package lecture3;
public class MainClass {
public static void main(String[] args) {} }
```

```
package lecture3;
public class Human { }
```

All attributes and methods related to a person will be put into **Human** while **MainClass** contains code that uses humans.

This is what is called encapsulation.

Creating an instance of Human

We can create an instance of **Human**. Instantiation happens while the code is running. The instance goes into the heap.

```
public class MainClass {
public static void main(String[] args) {
String s = new String("Hello world");
System.out.println(s); // Writes "Hello world".
Human h = new Human();
System.out.println(h);// Writes heap address of the instance }}
```

An address exists $\Rightarrow$ "h" is connected to code in the heap.

The instance of the class **Human** is called an object.

Variables and access modifiers

We give the object (instance of **Human**) a name.

```
public class Human { public String name; }
```

We can create instances of **Human** as follows:

```
public class MainClass {
public static void main(String[] args) {
Human h = new Human(); h.name = "Mark";
System.out.println(h.name); }}
```

h contains an address or reference to its content in the heap.

The operator . allows us to access the object's members (variables or methods) with the access modifier *public*.

Variables and access modifiers

```
public class Human { private String name;
public String getName(){ return name; }
public void setName(String arg){ name = arg; }}
```

name is not visible in *main(..)* if its access modifier is *private*.

Method names start with a lowercase letter.

getName() and *setName(arg)* are not *static* and go into the heap. They have access to private variables in the same class.

The return type matches the variable type behind *return*.

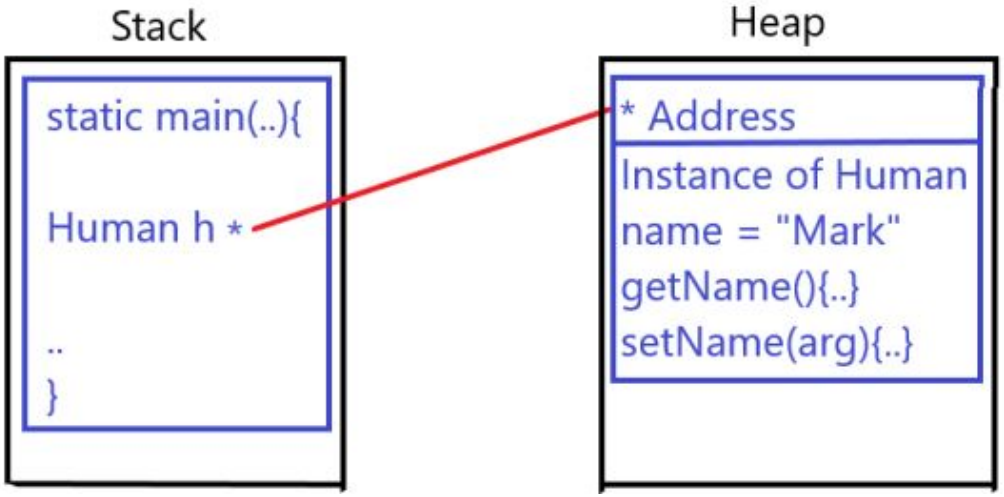
Global variables like *name* are declared before any method.

arg is a local variable valid inside the body of *setName(arg)*.

Memory arrangement

When you create an instance of **Human**, the class is compiled and the executable goes into the heap.

h stores the address of the instance of **Human**.



getName(), and *setName(arg)* are not declared static and go into the heap as a part of the object named "Mark".

The line *name = "Mark";* creates a String object and the content goes into a separate heap slot.

Using the methods getName() and setName(arg)

We can access and modify the *private* variable *name* through the *public* methods.

```
public class MainClass {
public static void main(String[] args) {
Human h = new Human();
h.setName("Mark");
System.out.println(h.getName());}}
```

We use again the operator . to access the methods in the instance of **Human**.

The instance of **Human** attached to *h* is a folder for all attributes and methods that belong to the object named "Mark".

The standard method toString()

System.out.println(h); in *main* returns the address of *h*.

toString() allows us to replace it by a user-defined value.

```
public class Human {
private String name;
public void setName(String arg){ name = arg; }
public String toString(){ return "The name is: " + name;} }
```

```
public class MainClass {
public static void main(String[] args){
Human h = new Human(); h.setName("Mark");
System.out.println(h); // writes out the return value of toString }}
```

"+" concatenates both strings giving "The name is: Mark".

Constructors

We presently create an instance of human as

```
Human h = new Human();
```

The parenthesis tells us that *new* precedes a method call.

Constructors are methods, which have the class name.

We have not implemented yet any method *Human()*;

Java comes with a default constructor.

```
public Human(){ } // This constructor does nothing.
```

The *new* command creates an instance of **Human** and the constructor *Human()* tells it how to do it.

Implementing a default constructor

```
public class Human {
private String name;
public void setName(String arg){ name = arg; }
public String toString(){ return "The name is: " + name; } }
```

private String name; is just a declaration.

If we call *System.out.println(h)* in *main(..)* before we use *setName(arg)* then the code crashes.

We implement our own default constructor

```
public class Human {
private String name;
public Human(){ name = "Mark";}
public void setName(String arg){ name = arg; }
public String toString(){ return "The name is: " + name; } }
```

Implementing a second constructor

We also want a constructor that takes user-defined values.

```
public class Human {
private String name;
public Human(){ name = "Mark"; }
public Human(String arg){ name = arg; }
public void setName(String arg){ name = arg; }
public String toString(){ return "The name is: " + name; } }
```

```
public class MainClass {
public static void main(String[] args) {
Human h1 = new Human(); Human h2 = new Human("Eric");
System.out.println(h1); System.out.println(h2); }}
```

main() writes "The name is: Mark" and "The name is: Eric".

Instance methods and instance variables

```
public class MainClass {
public static void main(String[] args) {
Human h1 = new Human(); Human h2 = new Human("Eric");
h1.setName("Karl"); h2.setName("Britt");
System.out.println(h1); System.out.println(h2);}}
```

Every instance of **Human** (here, *h1* and *h2*) has an own value for the instance variable *name* and an own copy of the instance method *setName(arg)*.

Calling *h1.setName(arg)* changes the value of *name* in *h1*.

The console output is thus "The name is: Karl" followed by "The name is: Britt".

Motivating class variables

We want to set *name* to a default value "Britt" in *main()*.

```
public class Human {
private String name, defaultN; // default is a reserved name
public Human(String arg1, String arg2)
{ name = arg1; defaultN = arg2; }
public void setToDefault(){ name = defaultN; }
public String toString(){return "The name is: " + name;}}
```

```
public static void main(String[] args){
Human h1 = new Human("Mark","Britt");
Human h2 = new Human("Eric","Britt");
h1.setToDefault(); h2.setToDefault();
System.out.println(h1); System.out.println(h2);}
```

All instances have an own copy of the same default "Britt". If we want to change it, we need to do it for all instances.

Class variables

A class variable is shared by all instances of this class.

We change an instance variable into a class variable with the keyword static between the access modifier and the type.

```
public class Human {
private String name; // Instance variable
private static String defaultN = "Karl"; // Class variable
public Human(String arg){ name = arg;}
public void setToDefault(){ name = defaultN; }
public String toString(){ return "The name is: " + name;} }
```

We should not initialize the class variable in the constructor since *defaultN* has the same value for all instances of **Human**.

Class methods

Values of variables should be changed with methods.

Class variables like *defaultN* are changed with a class method.

Class methods are declared static.

```
public class Human {
private String name; // Instance variable
private static String defaultN = "Karl"; // Class variable
public Human(String arg){ name = arg; }
public void setToDefault(){ name = defaultN; }
public static void changeDefault(String arg){ defaultN = arg; }
public String toString(){ return "The name is: " + name; } }
```

```
Human.changeDefault("Britt"); // call in main().
```

Summary

We have discussed

- what classes and objects are in object-oriented programming.
- Instance and class variables.
- Instance and class methods.
- Constructors and the method *toString()*