Classes & Objects

1 General Form of a Class

Remember, when you define a class, you declare its form and nature. You do this by declaring its instance variables and the methods that operate on them.

The basic structure is therefore:

```
public class  classname {

     //declare instance variables

     private int price;
     private int balance;
     …

     // declare constructor

     public classname(){

     //set default values of variables, if necessary
     }

     // declare methods

     public void getBalance(){

     …..
     }
}
```

Instance Variables:

These store the data for each object to use. They tell us the characteristics of an object. (“Eigenschaften”) .

Constructor:

Allows each object to be set up properly when it is first created. A class can have more than one constructor. You can also pass parameters through a constructor.

Methods:

They implement the behaviour of the objects.

2 Types of Methods:

There are two types of methods:

Accessor methods = "getter"-Methods. We can get information about the object.
Mutator methods = they change the state of the object.

➔ An *accessor* method will typically have a return type. Example:

```
public String getStudentName(){

     return studentName;

}
```

A *mutator* method doesn't necessarily have to return anything. Its return type can be **void**.

3 How Objects are created

Look at following code:

```
Vehicle minivan = new Vehicle();
```

This declaration does two things.

1. It declares a variable called **minivan** of the class type **Vehicle**.
2. It creates a physical copy of the object and assigns to **minivan** a *reference* to that object.

➔ The **new**-operator allocates memory for an object and returns a reference to it.

We can now do things with the object minivan, like change its state or ask for info.

This declaration can also be written like:

```
Vehicle minivan;          //declare reference to object
```

`minivan = new Vehicle();` //allocate a Vehicle object= minivan is linked with an object

The new-operator is therefore important to make sure that a reference is actually linked to a physical object. Otherwise you will encounter an exception (Null-Pointer-Exception).

4 Two types of Variables

As we have seen in some of the BlueJ examples, we can have variables that are a reference to an object or we can have primitive types.
We therefore have two types of variables:

- reference
- primitive

A reference variable always refers to an object.
A primitive variable does not refer to an object. They have normal binary values.

Java has following **primitive types:**

```
boolean
byte
char
double
float
int
long            (long integer)
short           (short integer)
```

Please check the Java API for details.

5 Reference Variables and Assignment

Look at following example:

```
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
```

Do **car1** and **car2** refer to different objects?

➔ No! car1 and car2 refer to the **same object**.

Any changes you do on car1 will also affect car2. → see Example.

6 Why do we need classes? Why can't we just use objects?

➔ A class is a *template*. It only defines the form of an object.
➔ Objects are *instances* of a class.

We therefore need both: with a class, we only have the *plan* how to build an object. It is a logical abstraction. When an object has been created, we have a physical representation of that class in memory.

7 When do we use *static*?

We have seen the **static** declaration in the **main** method. This means, the main method can be accessed without having an object. So wouldn't it make sense, to have all methods static?

Static methods only make sense if the behaviour is not dependent on an instance variable.

➔ What does that mean?
It means, sometimes we have utility methods ("nützliche Methoden") which do not need a specific object.

Example: check out the **Math** class in your Java class library. The Math class has a lot of methods which are static.

So we only really need static when we are dealing with very general utilities (math calculation that are not object related).

By the way: Variables can be static as well, which means, the value is the same for all instances of the class.

8 The *this* keyword

The reference to the object is called **this**. It can be very useful when you have parameters of the same name like the variables of an object. See Vehicle example.