

Remember "Reviewing Java" has many more examples and topics.

Tutorial 3 **Java Objects**

Creating Objects

In java, an object is a block of memory that will hold code and perform a specific task(s). This is the foundation of a class that was discussed previous.

An object can be initialized (or as it's said **instantiated**) with the keyword **new**. In order for an object to be used, it must be created first.

Here is a general way to define an object:

Method 1:

```
type name;
```

This way will declare, but not instantiate the object of a specific type. At some point in the program, the name object would have to be instantiated.

Method 2:

```
type name = new type();
```

This is a good way to declare AND instantiate an object. It still uses the assignment operator (=). This will then allow the object to be used as it is now active.

Let's observe method 2 a bit more as this is most common. Similar to a regular variable, this will initialize the object called *name* by invoking what's called a **constructor** for that class. A constructor can be thought of as a construction worker for a specific building. Without them, the building cannot be built. The same applies to a class; without a constructor, there is no class that can be used as an object.

Constructors

A constructor must be part of a class if you are going to use it as a type. It can also be thought of as a method that will perform specific actions.

There are rules about constructors. First, the constructor MUST BE PUBLIC. Second, it MUST MATCH THE CLASS NAME EXACTLY. Thirdly, it contains no other keywords such as void, int, boolean etc.

Remember "Reviewing Java" has many more examples and topics.

Here is a class that contains a constructor:

```
public class Example1{

    public Example1(){
        System.out.println("Welcome!");
    }
    public static void main(String args[]){
        Example1 ex1 = new Example1();
    }
}
```

A small example at that. What this will do is create a new instance of the object Example1. Contained in the constructor is a simple welcome message that will print out "Welcome!" when instantiated.

Instance variables

A class or object can have it's own variables. In Java, these are called **instance variables**. By its definition, an instance variable is unique to each instance of the class; so in general, each time a class is "instantiated" with the new operator, there is another variable associated with it.

These variables are declared outside any methods you may have. These variables are declared either public, private or protected. They are global to the class or object they appear in. Let's see a small example.

Example 1: Instance variables

```
public class Variables{

    private int x = 0;

    public Variables(){
        x++;
        System.out.println("Welcome! " + x);
    }
    public static void main(String args[]){
        Variables var1 = new Variables();
    }
}
```

In this example, the instance variable x is declared private. In addition, it is global to the Variables object.

Remember "Reviewing Java" has many more examples and topics.

The program simply prints out "Welcome! 1" when the program starts.

Static variables

Static variables are quite a bit harder to understand. Do not be confused with a static variable from C++. They are very different.

In java, a **static variable** (also called a **class variable**), is a variable that is given a fixed block of memory. The static keyword tells the compiler that there is exactly one copy of this variable in existence, no matter how many times the class has been instantiated.

Let's just think of a real world example. Say you own a car. The car will always have four wheels in the US. So if you made a class in java called *Car*, a variable called *numWheels* can be made static since it will be the same for every car.

Let's see an example of a static variable.

Example 2: Static variables

```
public class SVariables{

    private static int x = 0;
    private int y = 3;

    public SVariables(){
        x++;
        y += 5;
        System.out.println(y + " -- Welcome -- " + x);
    }
    public static void main(String args[]){
        SVariables s1 = new SVariables();
        SVariables s1a= new SVariables();
        SVariables s1b = new SVariables();
    }
}
```

When you run the above program, this is the output:

```
8 -- Welcome -- 1
8 -- Welcome -- 2
8 -- Welcome -- 3
```

Here is why the output is this way. With the *s1* instance of the object, the value of *x* is 1 and the value of *y* is 8, as per the arithmetic operations in the constructor.

Remember "Reviewing Java" has many more examples and topics.

The next instance of the object *SVariables* is *s1a*. Here, the value of *x* is 2 BECAUSE OF THE STATIC while the value of *y* is again 8, BECAUSE OF THE NON-STATIC.

In the final instance of the object *SVariables*, the *x* is 2 while the *y* is 8.