**object**

The principal building blocks of object-oriented programs. Each object is a programming unit consisting of data (instance variables) and functionality (instance methods). See also class.

**method**

A function defined in a class. See also instance method, [class method](http://java.sun.com/docs/glossary.html#cmethod). Unless specified otherwise, a method is not static.

**class**

In the Java programming language, a type that defines the implementation of a particular kind of object. A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass will implicitly be Object.

**property**

Characteristics of an object that users can set, such as the color of a window.

**profiles**

**instance**

An object of a particular class. In programs written in the Java programming language, an instance of a class is created using the new operator followed by the class name.

**inheritance**

The concept of classes automatically containing the variables and methods defined in their [supertypes](http://java.sun.com/docs/glossary.html#supertype). See also [superclass](http://java.sun.com/docs/glossary.html#superclass), [subclass](http://java.sun.com/docs/glossary.html#subclass).

**constructor**

A pseudo-method that creates an object. In the Java programming language, constructors are instance methods with the same name as their class. Constructors are invoked using the new keyword.

**super**

A Java keyword used to access members of a class inherited by the class in which it appears.

**static**

A Java keyword used to define a variable as a class variable. Classes maintain one copy of class variables regardless of how many instances exist of that class. static can also be used to define a method as a class method. Class methods are invoked by the class instead of a specific instance, and can only operate on class variables.

The *while* statement is used to repeat a block of statements while some condition is true. The condition must become false somewhere in the loop, otherwise it will never terminate.

While(x = 34){

System.out.println(“x still equals 34);

}

The *for* loop groups (1) initialization of a variable, (2) testing a condition, and (3) updating a value before the next iteration together into one statement, making it more readable and less error-prone than the equivalent *while* loop. For repeating code a known number of times, the *for* loop is the right choice.

for(int i = 0; i > 25; I++){

System.out.println(“I is still greater than 25”);

}

When you want to test at the end to see whether something should be repeated, the *do..while* statement is the natural choice.

String ans;

do {

. . .

ans = JOptionPane.showInputDialog(null, "Do it again (Y/N)?");

} while (ans.equalsIgnoreCase("Y"));

**import javax.swing.\*;- so you can use JOptionpane**

**import java.IOException – so you can use exception with buffered reader**