METHODS IN JAVA

A class represent the properties and behabiour of an object. An methods represent these behaviours.

Methods in Java are blocks of code with

- a name
- a paranthesis that can accept zero or more parameters
- can optionally return a value.
- has a body block enclosed within two curly braces {}.

Signature and body of a method

The first line of a method with its **access specifier** (who all can access), **return type** (what will this method return to its caller), name and the paranthesis with zero or more parameters, is called the **method signature**.
The body follows the signature starting with { and ending with }.

Example: The main method
When you execute a Java class (using java command) in console, JVM first calls a predefined method called main with a particular signature and then we can start calling other methods from there.

```
public static void main(String[] args)

{

//method body

}
```

Explanation of various terms used:

- public - is the access specifier that tells how visible the method is within and outside the class. Possible values are public, private, protected and default (no access modifier).
- static - this is a static method, not an instance method.
- void - return type. void means nothing is returned.
- main - name of the method.
- args - A String array to accept command line arguments.

Rules for declaring a method

- It is mandatory to specify **return type** while declaring a method; if nothing is returned, specify it as void.
- We should **return a value compatible** with the return type and do any casting if required.
- If you have specified the return type as **void** you can either omit the return statement or simply have the return statement as return;.
- We can have any primitive type like int, float etc or any reference type (class or interface) as the return type.
- Like variables, methods can also be static or instance. Static methods will have a static keyword in its signature line (as in the case of main method).
- From a static context (e.g. static method) you cannot access non-static members (e.g. non-static variables or methods). However, from non-static context (e.g. non-static method), you can access static members (e.g. static variables or methods).

Invoking methods

- Methods are not automatically invoked like initialization blocks and constructors, but need to be invoked explicitely from within the code; only exception is the main method which is invoked automatically by the JVM.
- Invocation is similar to the usage of instance and static variables, but need to pass any required arguments.
    - For instance if your method signature says add(int a, int b), you need to invoke it as add(1,2) or add(x,y) where x and y are integers.

Using object reference for instance methods

Instance methods are invoked using an object reference.

Example: An instance method with signature public void myMethod(int i) can be invoked as:

```
MyClass mc = new MyClass();

mc.myMethod(5);
```

Saving the return value after calling a method

If the method return a type, we can assign it to a variable of the same type. An instance method with signature public void myMethod() can be invoked and return type assigned to a variable as:

```
MyClass mc = new MyClass();

int i = mc.myMethod();
```

Invoking static methods

Like static variables, static methods can be invoked either using the object reference or the class name.

```
MyClass mc = new MyClass();

mc.myStaticMethod();

MyClass.myStaticMethod();
```

It is always preferred to use class names with static variables and methods.

Method Overloading

We can have multiple methods with same name, but different signatures.

```
public static void main(String[] args){}

public static void main({}
```

- We can overload even the main method, but Java will only call the one with predefined signature when a class is invoked. We can however invoke the other from within the main method.
- Return type is no part of signature and hence two methods with same name and argument types, but different return types are not valid overloading which is not allowed.
- Correct version of the method will be invoked based on the types you specify while invoking. Hence whenever you see a declaration and/or invocation, make sure you can invoke without confusion as Java will not allow any declaration or invocation which can lead to ambiguity.

Blocks within methods

When a block without any name appear inside a method body block, the block is considered as a single statement and executed when the containing method is executed.

A block within the main method will look as:

```
public static void main(String[] args) //outer block
```

```
{
  int var1=0;
  int var2;
  //inner block within method
  {
    int var3=0;
    System.out.println("var1="+var1);
    System.out.println("var3="+var3);
  }
  System.out.println("var1="+var1);
  /* Below line won't compile as var2 is not initialized.  */
  //System.out.println("var2="+var2);
  /* Below line won't compile as var3 is not visible outside the block in which it was defined.  */
  //System.out.println("var3="+var3);
}
```

Source : http://javajee.com/methods-in-java