**Java Class 6: Modifiers and methods**

**Modifiers** are keywords that you add to those definitions to change their meanings.

**Java access modifiers:**

Visible to the package, the **default**. No modifiers are needed.

Visible to the class only - **private**.

Visible to the world - **public**

Visible to the package and all subclasses - **protected**.

Note:

class declaration : class WeAreBoredToday {

}

**Access Modifers: restrict visibility level**

**public:** open visibility - from any other class, package, sub-classes and outside the project (world / global)

**private**: most restricted member – only seen within the class

default (no other declaration or no modifier): visible within class and within other classes of the same package



**B. Non-access modifiers**

Non-access modifiers help achieve many other functionalities - but not the visibility.

·      The ***static*** modifier for creating **class-level** methods and variables.

public **static** void main(String[]args) {

}

·      The ***final*** modifier for finalizing the implementations of classes, methods, and variables.

Public static final String firstDayOf Week = “Monday”;

·      The ***abstract*** modifier for creating abstract classes and methods.

·      *synchronized, transient* and *volatile* modifiers are used for threads.

**Examples:**

**abstract:**

**package** classsix;

**public** **abstract** **class** AccessModifiers {

**abstract** **void** printName();

**public** **void** get() {

            }

}

**final:**

**package** classsix;

**public** **final** **class** AccessModifiers {

**public** **static** **void** main(String[] args) {

**final** String NAME= "KaiserTech";

**final** **int** x = 5;

                           x += 5;

                           System.***out***.println(x);

            }

}

**Notes:**

**static** variables and methods can be accessed from anywhere in the class and can be used inside any **static** method. This is why **static** variables and methods are called **class-level** variables and methods.

**non-static** variables and methods can be used **with the help of an instance variable or object**

How to create an abstract method?

By using the **abstract** keyword before the class or method

An **abstract method** can only be found inside an **abstract class** and Interface**.**

**An abstract class** can contain both **abstract** and **non-abstract methods**

**Abstract methods** have **no body**

**Non-abstract methods have method body**

**package** classsix;

**public** **abstract** **class** AbsDemo1 {

**abstract** **void** getMyMoney();

**void** goHomeAndStudyHard() {

    }

}

Abstract class has **abstract** keyword before class declaration

An abstract class contains **abstract method** and **non-abstract** methods

An abstract class MUST contain at least **one** **abstract method**

**Subclass (child class) and superclass (Parent class):**

**( C**hild comes first on the left hand side, then comes Father or **P**arent class)

**public** **class** BankAccount **extends** Customer{

**public** **static** **void** main(String[] args) {

          System.***out***.println(Father.*skinColor*);

          Father.*getRest*();

          Father.*printValue*();

    }

}

**Starting about method:**

A **method** is

a collection of statements grouped together to perform an action, or

a block of code that performs a specific task.

Methods are also known as **functions** or **sub-routines**. Methods are bound to a class and they define the behavior of a class.

**Simply: methods** do things

For example:

**main** method **runs** the program:

public static void **main( )** {

}

**print** method prints a thing

A method has a pair of parentheses at the end of a method name ( )

We can have one or multiple methods in a class.

main () method runs the entire program

But other methods have different purposes: for example -

Print something

Add 4 numbers: addFourNumbers(){

}

Can you remember the **toUpperCase()** or **length()** method for String class?

Those methods served our purposes

But Java does not cover all the situations, so we need to create methods of our own

Like:

pleaseJoinTheClass(){

}

isPijushPresent() {

}

Are you ok with these?

**Green circles** refer to methods.

**Properties of Java method:**

methods do the actual work, aka functions or sub-routines.

methods are smart containers which can contain other containers.

a method can be called within another method.

but a method cannot contain declaration of another method.

methods generally have a method body marked by { }

methods can have the same name but only with different parameters.

methods have identifiers with a pair of parentheses at the end. e.g. printName()

methods may or may not have parameters

methods may return values or not (in case of a void method)

method names have lowerCamelcCasing style

method signature = methodName + parameter list

methods may contain parameters of the same or different types but returns a single value

**Advantages of Methods:**

methods make code reusable

reduce the code size

avoids redundancy

custom methods make code more readable and meaningful

**Types of methods:**

A. Static method and non-static or instance method:

**class** WeAreBoredToday {

**int** x = 12;

    // static method

**public** **static** **void** main(String[] args) {

    }

    // non-static method

**private** **void** printName() {

    }

}

B. Built-in method and user-defined / custom method

C. method without any parameter, e.g. printMyName(){  } and method with parameters, e.g. getSum(2, 3) { }

**class** WeAreBoredToday {

**public** **static** **void** main(String[] args) {

          System.***out***.println(*getSum*(10, 11));

          System.***out***.println(*c*

*printMyName*();

    }

**public** **static** **void** printMyName() {

          System.***out***.println("Samir Chamling!");

    }

**public** **static** **int** getSum(**int** x, **int** y) {

**return** x + y;

    }

}

Parameters and typed placeholders: getSum(**int x**, int y)

Arguments are actual values: *getSum*(10, 11)

**Method overloading example:**

            Same method name but with different parameters

**package** classsix;

**public** **class** MethodOverloading {

**public** **static** **void** main(String[] args) {

            }

**public** **static** **int** getSum(**int** x, **int** y, **int** z) {

**int** sum = 0;

**return** sum;

            }

**public** **static** **int** getSum(**int** x, **int** y) {

**int** sum =0;

**return** sum;

            }

}

**syntax of a method:**

**public** **static** **final** **int** getTotal(**int** x, **int** y)

**return** x+y;

}

**Writing the first user-defined method:**

**public class MyFirstTestMethod {**

**static void myFirstMethod() {**

**System.out.println("My first method just got executed!");**

**}**

**public static void main(String[] args) {**

**myFirstMethod ();**

**}**

**}**

**Method writing steps:**

Identify the input (parameters), output (return type) and the serving bucket

Write the method signature

The return container must be similar to the type stated in the method

Create the return container or bucket

Call the method during execution

Methods have **()**

Methods follow lower camel case: **p**lease**J**oin**T**he**C**lass**U**ntil**O**ur**O**ffice**O**pens() { }

**Types of methods:**

**A. Built-in methods** or Pre-built methods: Java provides these ready-made, readily-available methods for us to use and we do not need to create them. So, these are also known as **Standard Library Methods.** We just call them and use them right the way. Example:

**main( )** method,

**println()**,

**toUpperCase()**

But we can create methods, too – we call these methods as **custom** or **user-defined methods:**

**Examples: getSum() { } goHomeAndStudy(){ }**

**Methods** are symbolized by **GREEN circles:**

**s** for static methods

**no s** for non-static methods

**large GREEN C symbolizes a Class**

**void = empty / nothing**

**types of methods:**

A. built-in method and user-defined methods

B. static method and non-static method

C. A method may return the type you want:

Or may not return anything at all: **void methods,** have **void** keyword before the method name

D. Method with or without parameter(s)

**A method may contain no, one, or many parameters. And parameters may be of the same type or different type**

**My request:**

A. Sapin, please go to market and watch the game there: Sapin is not returning anything

System.out.**println**(“My name is Rajendra!”);  -> just prints the name

B. Nepal, please go to market and bring a pen for me: Nepal returning a pen

sumOfTwoNumbers(12, 11) -> returns the sum of two numbers - **23**

calling a method mean writing code for the method **inside** the main method.

Main() **{**   main method starts here

anyName(); -> is a void method and has a print statement (**println**). So when we call the method inside main () – its content gets printed

**}** main method ends here

**public** **static** **void** anyName() {

  System.***out***.**println**("I am from anyName method");

}

When we call a **method with a return type** in the **main method** – we print the **final product** of the method – not just the print already done there inside the method.

When there is a return type for method **getSum()**:

i.  Call the method inside main method

ii. Call the method inside a print statement

Java can understand and run no matter where you place the main method

And the type we define in the method must be the **same** in the **return type**

For example:

public static **int** getSum(int x, int y){

**int** sum = x +y;

return sum;   //is an **int**

Note: the **main() method** is static so only **static methods** can be called inside directly

So, a non-static method = instance method

So, we have to create an instance / object of the class

**Finally, there are 2 types of methods:**

method **with** parameter(s) - printName(){   }

  method **without** parameter(s) - getSum(int x, int y) {   }

do you see the differences?

I want **biriyani**:

main()**{**

System.out.println(getBiriyani(1 lb rice, 2 lbs chicken, 3 spoon spices));

**}**

getBiriyani(rice, chicken, spices){

boil rice;

  cook chicken;

  add spices;

biriyani = mix all these together;

return biriyani;

return is a serving container. When the method executes, we get whatever is placed inside it

**PLEASE take a picture in your camera and practice the code in your Eclipse**

**public** **static** **int** getResult(**int** x, **double** y) { // another method with parameters of different types

**int** result = x + (**int**) y;

**return** result;

  }

**Note:** many methods may have the same name – but the parameters must be different. This is called **Method Overloading**

**etSum(int x, int y) { }**

**getSum(int x, int y, int z) { }**

**getSum(int x, int y, double w) { }**

**getSum(int x, int y, float s) { }**

**Coding examples:**

**Methods in general:**

**package** classthree;

**public** **class** TodaysClass {

**public** **static** **void** pleaseJoinTheClass() {

       System.***out***.println("Hello, Brothers! Please join the class!");

  }

**public** **void** printMyName() { // this is non-static - so no s

  }

**public** **static** **void** anyName() {

  System.***out***.println("I am from anyName method");

}

**public** **static** String printName() { // this method must be called inside a print statement in main()

  String FirstName = "Donald";  // this methods has no print statement

  System.***out***.println(FirstName);

**return** FirstName;

}

**public** **static** **void** main(String[] args) { // method

  String name ="NEW YORK IS THE BEST";

  System.***out***.println(name.toLowerCase());   // 2 methods are: println()

  System.***out***.println(*printName*());

*anyName*();

*anotherMethod*();

}

**static** **void** anotherMethod() {

  System.***out***.println("Another method for Samir Chamling :)");

}

}

**Method overloading:**

**package** classthree;

**public** **class** MethodsWithTheSameName {

**public** **static** **void** main(String[] args) {

       System.***out***.println(*getSum*(2, 5));

       System.***out***.println(*getSum*(2, 4, 6));

       System.***out***.println("The third method: " + *getSum*(2, 4, (**int**) 2.444555));

  }

**public** **static** **int** getSum(**int** x, **int** y) {

**int** sum1 = x + y;

**return** sum1;

  }

**public** **static** **int** getSum(**int** m, **int** n, **int** o) {

**int** sum2 = m + n + o;

**return** sum2;

  }

**public** **static** **int** getSum(**int** e, **int** f, **float** g) {

**int** sum = e + f + (**int**) g; // type casting

**return** sum;

  }

}

**Methods with parameters and no parameter:**

**package** classthree;

**public** **class** MethodExamples {

**public** **static** **void** main(String[] args) {

       System.***out***.println(*getSumOfThreeNumbers*(10, 20, 30));

  }

**public** **static** **int** getSumOfThreeNumbers(**int** a, **int** b, **int** c) { // a method with 3 parameters

**int** sum = a + b +b c; // the sum is ready but we need a serving bucket or container

**return** sum;

  }

**public** **static** **void** methodWithNoParameter() { // a method without parameter -

       System.***out***.println("I am sorry, I have no parameter");

  }

**public** **static** **int** getResult(**int** x, **double** y) { // another method with parameters of different types

**int** result = x + (**int**) y;

**return** result;

  }

}

**Static and non-static Methods:**

**package** classthree;

**public** **class** AnotherClassForMethodDemo {

  // lets create 2 methods: one static and one non-static

  // and a main()

**public** **static**  **void** letGoForMovie() {                          // static method

       System.***out***.println("Lets watch the latest movie!");

  }

**public** **void** getTicketForMe() {

       System.***out***.println("Please get movie ticket for me!");   // non-static method

  }

  // main() method

**public** **static** **void** main(String[] args) {

       AnotherClassForMethodDemo abc = **new** AnotherClassForMethodDemo();

*letGoForMovie*();

       abc.getTicketForMe();

  }

}

**Methods with return type or not:**

**package** classthree;

**public** **class** MethodExamples {

**public** **static** **void** main(String[] args) {

       System.***out***.println(*getSumOfThreeNumbers*(10, 20, 30));

  }

**public** **static** **int** getSumOfThreeNumbers(**int** a, **int** b, **int** c) { // a method with 3 parameters

**int** sum = a + b + c; // the sum is ready but we need a serving bucket or container

**return** sum;

  }

**public** **static** **void** methodWithNoParameter() { // a method without parameter -

       System.***out***.println("I am sorry, I have no parameter");

  }

**public** **static** **int** getResult(**int** x, **double** y) { // another method with parameters of different types

**int** result = x + (**int**) y;

**return** result;

  }

}

**Java Recursive method**

Java **recursive method** is a function that calls itself and this technique is known as **recursion**.