Day 4

Method Overloading

```
10 + 20          =>  Addition
10 + 20.5        =>  Addition
10 + 20 + 30;        =>  Addition
10 + 20.1f + 30.5d;   =>  Addition
```

- If implementation of method is logically equivalent/same then we should give same name to the method.
- If we want to give same name to the method then we must use following rules:
- Rule 1 : If we want to give same name to the method and if type of all the parameters are same then number of parameters passed to the method must be different.

```
public class Program {
    //Type of all the parameters is int
    private static int sum(int num1, int num2) {    //2 parameters
        return num1 + num2;
    }
    //Type of all the parameters is int
    private static int sum(int num1, int num2, int num3) { //3 parameters
        return num1 + num2 + num3;
    }
    public static void main(String[] args) {
        int result = 0;
        result = Program.sum( 10, 20 );
        System.out.println("Result  :   "+result);

        result = Program.sum( 10, 20, 30 );
        System.out.println("Result  :   "+result);
    }
}
```

- Rule 2 : If we want to give same name to the method and if number of parameters are same then type of at least one parameter must be different.

```
public class Program {
    private static int sum(int num1, int num2) {    //2 parameters
        return num1 + num2;
    }
    private static double sum(int num1, double num2) { //2 parameters
        return num1 + num2;
    }
    public static void main(String[] args) {
        //int result = 0;
```

```
        int result1 = Program.sum( 10, 20 );
        System.out.println("Result  :   "+result1);

        double result2 = Program.sum( 10, 20.5d );
        System.out.println("Result  :   "+result2);
    }
}
```

- Rule 3 : If we want to give same name to the method and if number of parameters are same then order of type of type of parameters must be different.

```
public class Program {
    private static float sum(int num1, float num2) {    //2 parameters
        return num1 + num2;
    }
    private static float sum(float num1, int num2) { //2 parameters
        return num1 + num2;
    }
    public static void main(String[] args) {
        float result = 0;
        result = Program.sum( 10, 20.5f );
        System.out.println("Result  :   "+result);

        result = Program.sum( 10.1f, 20 );
        System.out.println("Result  :   "+result);
    }
}
```

- Rule 4 : Only on the basis of different return type, we can not give same name to the method.

```
public class Program {
    private static int sum(int num1, int num2) {    //OK
        return num1 + num2;
    }
    private static void sum(int num1, int num2) { //Not OK
        int result = num1 + num2;
        System.out.println("Result  :   "+result);
    }
    public static void main(String[] args) {
        int result = 0;
        result = Program.sum( 10, 20 );
        System.out.println("Result  :   "+result);

        Program.sum( 10, 20 );
        System.out.println("Result  :   "+result);
    }
}
```

- Using above rules, proecss of defining method with same name and different signature is called method overloading.
- Method overloading represents compile time polymorphism.
- Methods, which take part in overloading are called overloaded methods.
- print is overloaded method of java.io.PrintStream class

```
public void print(boolean);
public void print(char);
public void print(int);
public void print(long);
public void print(float);
public void print(double);
public void print(char[]);
public void print(String);
public void print(Object);
```

- println is overloaded method of java.io.PrintStream class

```
public void println();
public void println(boolean);
public void println(char);
public void println(int);
public void println(long);
public void println(float);
public void println(double);
public void println(char[]);
public void println(java.lang.String);
public void println(java.lang.Object);
```

- printf is overloaded method of java.io.PrintStream class

```
public java.io.PrintStream printf(java.lang.String, java.lang.Object...);
public java.io.PrintStream printf(java.util.Locale, java.lang.String,
java.lang.Object...);
```

- valueof is overloaded method of java.lang.String class

```
public static java.lang.String valueOf(java.lang.Object);
public static java.lang.String valueOf(char[]);
public static java.lang.String valueOf(char[], int, int);
public static java.lang.String valueOf(boolean);
public static java.lang.String valueOf(char);
public static java.lang.String valueOf(int);
public static java.lang.String valueOf(long);
```

```
public static java.lang.String valueOf(float);
public static java.lang.String valueOf(double)
```

- If implementation of method is logically equivalent/same then we should overload method.
- We can overload static as well as non static method in java.
- We can overload main method in Java. Consider following code:

```
public class Program {
    public static void main(String args) {
        System.out.println("public static void main(String args)");
    }
    public static void main(String[] args) {
        System.out.println("public static void main(String[] args)");
        Program.main("");
    }
}
//Output : public static void main(String[] args)
//public static void main(String args)
```

- Catching value from method is optional.

```
public class Program {
    public static int sum( int num1, int num2 ) {
        int result = num1 + num2;
        return result;
    }
    public static void main(String[] args) {
        int result = sum( 10, 20 ); //OK
        System.out.println("Result  :   "+result);

        sum( 50, 60 );  //OK
    }
}
```

- Since catching value from method is optional, return type is not considered in method overloading.
- For method overloading, method must be exist inside same scope.

```
class A{
    public void sum( int num1, int num2 ) {
        //System.out.println("A.Sum :   "+num1 + num2); //Sum   :   1020
        System.out.println("A.Sum   :   "+(num1 + num2));   //Sum   :   30
    }
}
class B extends A{
    public void sum( int num1, int num2, int num3 ) {
        System.out.println("B.Sum   :   "+(num1 + num2 + num3));    //Sum
:   30
```

```
    }
}
public class Program {
    public static void main(String[] args) {
        B b = new B();
        b.sum(50, 60);
        b.sum(10, 20, 30);
    }
    public static void main1(String[] args) {
        A a = new A( );
        a.sum(10, 20);
    }
}
```

Initialization

- It is a process of storing value inside variable during declaration.

```
int num1 = 10;      //Initialization
int num2 = num1;    //Initialization
String str = null;  //Initialization
```

- We can initialize instance only once.

```
int num1;
int num2 = num1;    //Error : The local variable num1 may not have been
initialized
```

Assignment

- It is a process of storing value inside variable after its declaration.

```
int num1 = 10, num2;
num2 = num1;    //Assignment => 10
num2 = 20;      //Assignment => 20
```

- Assignment can done multiple times.

Constructor

- Java syntax which looks like method but it is not a method.
- If we want to initialize instance then we should use constructor.
- Consider following syntax:

```
public Complex() {  //Constructor
    this.real = 10;
    this.imag = 20;
}
```

- Due to following reasons ctor(constructor) is considered as special syntax of java.

1. Its name is same as class name.
2. It doesn't have return type.
3. It is designed to call implicitly.
4. In the lifetime, it gets called once per instance.

- We can not call constructor on instance explicitly. Consider following code.

```
public static void main(String[] args) {
    //Instantiation : ClassName identifier = new ClassName( );
    Complex c1 = new Complex( ); //Here, Instance is created W/O passing
args
    //c1.Complex( );    //Explict call to the constructor : Not OK
}
```

- We can use any access modifier on constructor.
- If access modifier of constructor is public then we can create instance of a class anywhere(inside method of same class as well as different class).
- If access modifier of constructor is private then we can create instance of a class inside method of same class only.
- Important : Constructor do not create instance. Rather it initializes instance.
- Types of constructor:
    1. Parameterless constructor
    2. Parameterized constructor
    3. Default constructor

Parameterless constructor

- A constructor, which do not take any parameter, is called parameterless constructor.
- Syntax:

```
public Complex() {  //Constructor
    this.real = 10;
    this.imag = 20;
}
```

- If we create instance W/O passing argument then parameterless constructor gets called.

```
Complex c1 = new Complex( );    //Here on instance, parameterless ctor
will call
```

- Parameterless constructor is also called as zero argument constructor / user defined default constructor

Parameterized constructor

- A constructor which takes parameter is called Parameterized constructor.

```
public Complex( int real, int imag) {   //Parameterized Constructor
    this.real = real;
    this.imag = imag;
}
```

- If we create instance by passing arguments then parameterized constructor gets called.

```
Complex c1 = new Complex( 10, 20 ); //Here on instance, parameterized
constructor will call.
```

- Constructor calling sequence depends on order of instance declaration.
- Process of defining multiple constructors inside class is called constructor overloading.

```
class Complex{
    private int real;   //0
    private int imag;   //0
    public Complex( ) { //Constructor
        this.real = 10;
        this.imag = 20;
    }
    public Complex( int real, int imag) {   //Constructor
        this.real = real;
        this.imag = imag;
    }
}
```

```
class Employee{
    private String name;    //null
    private int empid;      //0
    private float salary;   /0.0
    public Employee( ){
        //TODO
    }
    public Employee( String name, int empid, float salary ){
        this.name = name;
```

```
        this.empid = empid;
        this.salary = salary;
    }
}
```

- To achieve reusability, we can call constructor from another constructor. It is called as constructor chaining.
- For constructor chaining, we should use this statement.
- this statement must be first statement inside constructor.

```
class Complex{
    private int real;   //0
    private int imag;   //0

    public Complex() {  //parameterless constructor
        //Constructor call must be the first statement in a constructor
        this( 10, 20 ); //Constructor chaining
    }
    public Complex(int real, int imag) {    //Parameterized constructor
        this.real = real;
        this.imag = imag;
    }
}
```

- Consider construtors of integer class

```
public Integer(int value);
public Integer(String s) throws NumberFormatException;
```

Default constructor

- If we do not define any constructor inside class then compiler generates on constructor for the class by default. It is called default constructor.
- Compiler do not generate default parameterized constructor. If we want to create instance by passing arguments then we should define parameterized constructor inside class.

java.lang.Object class

- It is non final(it means that we can extend it) and concrete class(it means that we can instantiate it) declared in java.lang package.
- java.lang.Object class do not extend any class or do not implement any interface. In other words it is super class of all the classes(not interefaces) in java language.
- It is also called as super cosmic base class / ultimate base class / root of java class hierarchy.

```
class Complex{
    //TODO
}
//is equivalent to
class Complex extends Object{
    //TODO
}
//Super class : java.lang.Object
//Sub class : Complex
```

- In Java, All the classes are directly/indirectly exetended from java.lang.Object class.
- Object class do not contain nested type(Interface/class/enum).
- Object class do not contain any field.
- Object class contains, compiler generated default constructor.

```
Object o1 = new Object( 125 );  //NOT OK
Object o2 = new Object( "Hello" );  //NOT OK
Object o3 = new Object( );  //OK
```

- Object class contains 11 methods:(5 non final methods + 6 final methods)

5 non final methods (2 native + 3 non native)of java.lang.Object class

```
1. public java.lang.String toString();
2. public boolean equals(Object);
3. public native int hashCode();
4. protected native Object clone() throws CloneNotSupportedException;
5. protected void finalize() throws Throwable;
```

6 final methods(4 native + 2 non native) of java.lang.Object class

```
6. public final native java.lang.Class<?> getClass();
7. public final void wait() throws java.lang.InterruptedException;
8. public final native void wait(long) throws
java.lang.InterruptedException;
9. public final void wait(long, int) throws
java.lang.InterruptedException;
10. public final native void notify();
11. public final native void notifyAll();
```

- In the context/environment of Java, C/C++ code is called native code.
- native method means, it is C++ language member function which is designed to call from java language.

```
native method <---> |JNI|<---> C++ code.
```

- Java Native Intereface(JNI). It is readymade framework to access native code in java.

toString method

- If we want tot represent/return state of java instance in String format then we should use toString() method.
- It is a method of java.lang.Object class.
- Syntax:

```
public String toString();
```

- If class do not contain toString() method then super class's(parent class) toString() method gets call. If any super class do not contain toString() then java.lang.Object class's toString gets call.
- Hashcode is a logical integer number that can be generated by processing state of the object.

```
public static int getCode( int number ) {
    int PRIME = 151;
    int result = 1;
    result = result * number + PRIME * number;
    return result;
}
public static void main(String[] args) {
    int number = 1983;
    int code =  Program.getCode( number );
    System.out.println("Numebr  :   "+number);  //1983
    System.out.println("Code    :   "+code);    //301416 => hashcode
    System.out.println("Code    :   "+Integer.toHexString(code));
//49968 => hashcode
}
```

- Consider implementation of toString() method of Object class
- The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object.

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

- In short, toString() methid of object class returs String in followinh format:

```
F.Q.ClassName@HashCode
```

- According to business logic, if implementation of super class method is logically incomplete then we should redefine/override method in sub class.
- The result in toString() method should be a concise but informative that is easy for a person to read. It is recommended that all subclasses override this method.

```
@Override
public String toString() {
    return "Employee [name=" + name + ", empid=" + empid + ", salary=" +
salary + "]";
}
```

"public static String format(String format,Object... args)" is a method of java.lang.String class.

```
String str = String.format( "%-20s%-5d%-10.2f", this.name, this.empid,
salary );
```

```
@Override
public String toString() {
    return String.format( "%-20s%-5d%-10.2f", this.name, this.empid,
salary );
}
```