Day 4

Class and Instance

- Following elements do not get space inside instance:
    1. Method parameter
    2. Method local variable
    3. Static field(also called class level variable)
    4. Methods(Static /non static)
    5. constructor
    6. Nested type
- Only non static field(also called as instance variable) get space once per instance according to order of their declaration inside class.
- Method do not get space inside instance. Rather, by passing reference, all the instances of same class share single copy of method.

Characteristics of instance

State of the instance

- value stored inside instance is called state.
- Value of the field represent state of the instance.

Behavior of the instance

- Set of operations that we can perform on instance represents behavior of that instance.
- Methods of the class represent behavior of the instance.

Indentity of the instance

- Value of any field which is used to identiy instance uniquely is called identity of the instance.

class

- Definition:
    1. Class is collection of fields and methods.
    2. Structure and behavior of instance depends on class hence class is considered as a template/model/blueprint for instance.
    3. Class is collection of such objects which is having common structure and common behavior.
- Class is a imaginary / logical term.
- Example : Book, Car, Mobile Phone etc.
- Class implementation represents encapsulation.

Instance

- Definition:
    1. In Java, object is also called as instance.

2. An entity, which has physical existance, is called instance.
3. An entity, which has state, behavior and identity is called instance.

- Instance is a real time / physical entity.
- Example : Linux Programming Interface, Tata nexon, iPhone 11 etc.
- Instantiation represents abstraction.

Difference between primitive and non primitive type

Value Type

1. Primtive type is also called as value type.
2. There are 8 value types in Java(boolean, byte, char, short, int, float, double, long).
3. Variable of value type contain value.

```
int num1 = new int( 100 );  //Not OK
int num1 = 100;
```

4. If we assign, variable of value type to the another variable of value type then value gets copied.

```
int num1 = 10;
int num2 = num1;    //value of num1 (10 )will be copied into num2.
```

5. Variable of value type by default contains 0(for the fields).

```
class Employee{
    int empid;  //0
}
```

6. We can not create instance of value type using new operator.

```
public static void main(String[] args) {
    int number = new int( );    //not ok
    int number = 0; //OK
}
```

7. variable of value type get space on Java Stack.

Reference Type

1. Non Primtive type is also called as reference type.
2. There are 4 refernce types in Java(Interface, class, type variable, array)
3. Variable of refernce type contains reference of the instance.

```
Employee emp( "Sandeep", 33, 45000.50f );   //Not OK
Employee emp = new Employee( "Sandeep", 33, 45000.50f );    //OK
```

4. If we assign, variable of reference type to the another variable of reference type then reference gets copied.

```
Employee emp1 = new Employee( "Sandeep", 33, 45000.50f );    //OK
Employee emp2 = emp1;   //reference copy
```

5. Variable of reference type by default contains null(for the fields).

```
class Employee{
    String name;    //null
    Date joinDate;  //null
}
6. It is mandatory to use new operator to create instance of reference
type.
```java
public static void main(String[] args) {
   Employee emp = new Employee( );  //OK
}
```

7. Instance of reference type get space on heap.

reference

```
class Date{
    private int day;
    private int month;
    private int year;
}
class Employee{
    String name;    //Non static field / Instance variable --- get space
inside instance
    int empid;      //Non static field / Instance variable --- get space
inside instance
    Date joinDate;  //Non static field / Instance variable --- get space
inside instance
}
```

- If we declare variable of primitive and non primitive type inside method then it gets space on java stack.

```
public static void main1(String[] args) {
    int empid = 0;  //Method Local Variable : Java Stack
```
```

```
    Date joinDate = null;   //Method Local Reference Variable : Java Stack
    joinDate = new Date();  //Instance : Heap
}
```

- If we declare variable of primitive and non primitive type as a field of the class then it gets space inside instance(Heap).

```
public static void main(String[] args) {
    Employee emp = null;    //Method Local Reference Variable : Java Stack
    emp = new Employee( );  //Instance : Heap
}
```

Comments

- If we want to maintain documentation of source code then we should use comments.
- Types:
    1. Implementation comment
        1. Single line comment(//)
        2. Block / multiline comment(/* */)
    2. Documentation comment
        1. Java doc comment(/** */)

Method Overloading

- Consider following code

```
10 + 20;                    //Addition

10 + 20 + 30                //Addition

10 + 20.5;                 //Addition

10 + 20.5f + 30.2d;       //Addition
```

- If implementation of method is logically same/similar/equivalent 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 parameter is same then number of parameters passed to method must be different.

```
public class Program {
    private static int sum(int num1, int num2) {    //2 parameters
        return num1 + num2;
    }
    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 = sum(10, 20 );
        System.out.println("Result  :   "+result);

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

        double result2 = sum(10, 20.5 );
        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 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 result1 = sum(10, 20.1f );
        System.out.println("Result  :   "+result1);

        float result2 = sum(10.1f, 20 );
```

```
        System.out.println("Result  :   "+result2);
    }
}
```

- 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) {    //2 parameters
        return num1 + num2;
    }
    private static void sum(int num1, int num2) {   //2 parameters
        return num1 + num2;
    }
    public static void main(String[] args) {
        int result = sum(10, 20 );
        System.out.println("Result  :   "+result);
        sum(50, 60 );
    }
}
```

- If we define methods using above rules then it is called method overloading. In short, process of defining method with same name and different signature is called method overloading.
- Methods, which take part in overloading are called overloaded methods.
- If implementation of method is logically same/similar/equivalent then we should overload method.
- print is overloaded method, declared in 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, declared in 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(String);
public void println(Object);
```

- printf is overloaded method, declared in java.io.PrintStream class.

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

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

```
public static String valueOf(Object);
public static String valueOf(char[]);
public static String valueOf(char[], int, int);
public static String valueOf(boolean);
public static String valueOf(char);
public static String valueOf(int);
public static String valueOf(long);
public static String valueOf(float);
public static String valueOf(double);
```

- We can overload main method in Java.

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

- Consider following example

```
public static void sum( int a, int b ){ //a, b => Method parameters /
parameters
    int result = a + b;
    System.out.println("Result  :   "+result);
}
public static void main(String[] args) {    //args =>  Method parameters /
parameters
    //Method Call
    Program.sum( 10, 20 );  //10, 20 => Method arguments / arguments

    int x = 10, y = 20;
    //Method Call
```

```
    Program.sum( x, y  );  //x, y => Method arguments / arguments
}
```

- For method overloading, method must exist inside same scope.

```
class A{
    public void sum( int num1, int num2 ) {
        //System.out.println("Result    :   "+num1 + num2 );    //Result
:   1020
        System.out.println("A.Result    :   "+(num1 + num2));   //Result
:   30
    }
}
class B extends A{
    public void sum( int num1, int num2 ) {
        System.out.println("B.Result    :   "+(num1 + num2));   //Result
:   30
    }
    public void sum( int num1, int num2, int num3 ) {
        System.out.println("B.Result    :   "+(num1 + num2 + num3));
    }
}
public class Program {
    public static void main(String[] args) {
        B b = new B( );
        b.sum(10, 20);
        b.sum(10, 20, 30 );
    }
    public static void main1(String[] args) {
        A a = new A( );
        a.sum(10, 20);
    }
}
```

- return type is not considered in method overloading.
- Since catching value from method is optional, return type is not considered in overloading.

Initialization

- Intialization is the process of storing value inside variable during its declaration.

```
int num1 = 10;      //Initialization
int num2 = num1;    //Initialization
```

```
int num1;
int num2 = num1;    //Error
```

- We can intialize any variable only once.

Assignment

- Assignment is the process of storing value inside variable after its declaration.

```
int num1 = 10, num2;
num2 = num1;    //Assignment    :   10
num2 = 20;    //Assignment    :   20
num2 = 30;    //Assignment    :   30
```

- We can do assignment multiple times.

Constructor

- Java syntax, which look like method but which is not a method and which is designed to initialize instance is called constructor(ctor) of the class.
- Due to following reason, constructor is considered as special syntax of java
    1. Its name is same as class name.
    2. It doesn't have any return type.
    3. It is designed to call implicitly.
    4. In the lifetime of instance, it gets called only once.
- Compiler do not call constructor on reference. Constructor gets called once per instance.
- We can not call constructor on instance explicitly.

```
public static void main(String[] args) {
    Complex c1 = new Complex( );    //OK
    c1.Complex( );  //Not OK
}
```

- Constructor do not create instance rather it initializes instance.
- We can use any access modifier on constructor.
- If constructor is public then we can create instance of a class inside method of same class as well as different class.
- If constructor is private then we can create instance of a class inside method of same class only.

Object Class

toString method