Day 6

Method Overloading

- Consider code in C

```
void sum( int num1, int num2 )
{
    int result = num1 + num2;
    printf("Result  :   %d\n", result);
}
void add(int num1, int num2, int num3) {
    int result = num1 + num2 + num3;
    printf("Result  :   %d\n", result);
}
int main( void ) {
    sum( 10, 20 );
    add( 10, 20 , 30 );
    return 0;
}
```

- In C, we can give same name to the multiple functions.
- If implemenation of method is logically same then we should give same name to the method. It helps developer to reduce maintenance of code.
- If we want to give same name to the method then it is necessary to follow some rules.

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 {
    public static void sum( int num1, int num2 ) {
        int result = num1 + num2;
        System.out.println("Result  :   "+result);
    }
    public static void sum(int num1, int num2, int num3) {
        int result = num1 + num2 + num3;
        System.out.println("Result  :   "+result);
    }
    public static void main(String[] args) {
        Program.sum( 10, 20 );
        Program.sum( 10, 20 , 30 );
    }
}
```

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

```
public class Program {
    public static void sum( int num1, int num2 ) {
        int result = num1 + num2;
        System.out.println("Result  :   "+result);
    }
    public static void sum(int num1, double num2) {
        double result = num1 + num2 ;
        System.out.println("Result  :   "+result);
    }
    public static void main(String[] args) {
        Program.sum( 10, 20 );
        Program.sum( 10, 20.5 );
    }
}
```

3. If we want to give same name to the method and if number of parameters passed to the method are same then order of type of parameters must be diffrent.

```
public class Program {
    public static void sum( int num1, float num2 ) {
        float result = num1 + num2;
        System.out.println("Result  :   "+result);
    }
    public static void sum(float num1, int num2) {
        float result = num1 + num2 ;
        System.out.println("Result  :   "+result);
    }
    public static void main(String[] args) {
        Program.sum( 10, 20.1f );
        Program.sum( 10.1f, 20 );
    }
}
```

- Method can return value but catching that value is optional.

4. On the basis of only diffrent return type we can not give same name the method.

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

        Program.sum(50, 60);
    }
}
```

- Process of defining method using above rules is called as method overloading. In simple words, process of defining method with same name and diffrent signature is called method overloading.
- Methods, which take part in overloading are called overloaded methods.
- 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 Dac");
    }
}
```

- We can define multiple constructors inside class. It is called constructor overloading.

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

- We can overload static as well as non static methods.
    - valueOf() is overloaded static method of java.lang.String class.
    - print, println, printf are non static overloaded methods of java.io.PrintStream class.
- To overload method, minimum 2 definitions are required.
- If implemenation of methods are logically same/equivalent then we should overload method.
- For method overloading, methods must be exist inside same scope.
- Since catching value from method is optional, return type is not considered in method overloading.

Final

- final is keyword in Java

```
public class Program {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        final int num1;
        System.out.print("Num1  :   ");
        num1 = sc.nextInt();
        //num1 = 200;
        System.out.println(num1);
    }
    public static void main3(String[] args) {
        final int num1;
        num1 = 10;  //OK
        //num1 = 20;    //NOT OK
        System.out.println(num1);
    }
    public static void main2(String[] args) {
        final int num1 = 10;    //OK
        //num1 = 20;    //Not OK
        System.out.println(num1);
    }
    public static void main1(String[] args) {
        final int count = 10;
        //count = count + 1;    //Not OK
        System.out.println("Count   :   "+count);
    }
}
```

- If we dont want to modify value of method local variable then we should declare it final.
- Once we store value inside final variable then we can not modify it.
- We can provide value to the final variable at compile time as well as runtime.

Final Field

- If we dont want to modify state/value of any field inside any method of the class then we should declare such field final.
- If we want to declare any field final then we should declare it static too.

```
class Test{
    public static final int number = 10;
    public void showRecord( ) {
        //this.number  = this.number + 1;   //Not OK
        System.out.println("Number  :   "+Test.number);
    }
    public void displayRecord( ) {
        //this.number  = this.number + 1;   //Not OK
        System.out.println("Number  :   "+Test.number);
    }
}
public class Program {
    public static void main(String[] args) {
        Test t = new Test( );
        t.showRecord();
        t.displayRecord();
    }
}
```

- We can declare reference final but we can not declare instance final.

```
final Complex c1 = new Complex(10, 20);
c1.setReal(11);
c1.setImag(22);
c1.printRecord();

c1 = new Complex(50, 60);   //Not OK
```