Agenda

- Revision
- Array
- Variable Arity/Argument Method
- Method Overloading
- Method Arguments
    - pass by value and reference
- Object/Field Initializer

Array (Demo01)

- It is a collection of similar type of data/elements
- Memory to the array elements are given contigously
- Array elements are accessed using index.
- In java, Arrays are of 3 types
    - 1. Single Dinmension Array
    - 2. Multi Dimension Array
    - 3. Ragged Array
        - An multidimension array whose size of second dimension is not fixed is called as Ragged Array
- In java, Array is a reference type
- If we access the index of an array which is greater then the size of an array then jvm throws an exception called as ArrayIndexOutOfBoundsException
- If we access the negative index of an array of an array then jvm throws an exception called as ArrayIndexOutOfBoundsException
- If we declare an array of size with negative value then JVM throws NegativeArraySizeException.

Variable Arity/Argument Method (Demo02)

- If we want to pass multiple no of paramaters of the same type to a method, then we can create an array and pass it as an argument to the method.
- Or we can also define an variable arity method which can accept multiple no of parameters of same type which it internally converts into an array.

Method Overloading (Demo03)

- Defining multiple methods with same name but diffent arguments(Signature) is called as method overloading

- Different arguemts/signature can be in one of the following ways

    1. Changing the count of parameters

```
public static void add(int num1, int num2) {
      System.out.println("Addition = " + (num1 + num2));
  }
```

```
    public static void add(int num1, int num2,int num3) {
        System.out.println("Addition = " + (num1 + num2 +num3));
    }
```

2. changing the type of parameters

```
public static void square(int num1) {
    System.out.println("Square = " + (num1 * num1));
}
public static void square(double num1) {
    System.out.println("Square = " + (num1 * num1));
}
```

3. changing the order/sequence of type of parameters

```
public static void multiply(int num1,double num2) {
    System.out.println("Multiplication = "+(num1*num2));
}

public static void multiply(double num1,int num2) {
    System.out.println("Multiplication = "+(num1*num2));
}
```

- If you apply the logic of method overloading on the construtor it is called as constructor overloading
- retun type is not considered in method overloading

Method Arguments (Demo04)

- In java whatever we pass to the method , it is always pass by value
- If primitive types are passed to the method then their copies are created.
- so whatever changes are done are not reflected into the original varaibles.
- if you want to change the original value of primitive types into other methods then pass their references of wrapper class or create an array of this primitive types and pass it to the method.
- If non primitive/reference types are passed to the methods , then copy of the reference is created.
- however the copy also has the same address of the abject as that of the original/passed reference.
- hence both the references are pointing as same object. so any changes done by any of the reference in that object will be reflected in same way in both the references.

Object/Field Initializer (Demo05)

- In C++/java we initialize the state of an object inside constructor
- Java have provided 3 ways to initialize the object
    - 1. Field initializer
    - 2. Object initializer

    - 3. Constructor
- the initializers have their own priority.
- first the field initializers initialize the fields.
- second if object initializers exists then the gets executed and they initialize the fields. if the field initializers have already initialized the fields then the values will get replaced.
- third initializer that gets called is the constructor.
- if ctor reinitializes the fields which are already initialized by the field or object initializer then all their values gets replaced.

1. Field Initializer

```
class Test{
  private int num1 = 10; // field Initializer
}
```

2. Object Initializer

- It is just a block in which we can initialize our object
- We can initialize our fields of the class inside this block.
- we can write as many object initializers we want.
- all the object initializers will be executed sequentually in the way they appear.

```
public class Test {
    private int num1 = 10; // Field initializers
    private int num2;
    private int num3;
// object initializers
    {
        System.out.println("Inside Object Initializer-1");
        num2 = 20;
    }

    // object initializers
    {
        System.out.println("Inside Object Initializer-2");
        num1 = 100;
        num3 = 30;
    }
}
```

3. Constructor

- It is the last initializer that gets executed depending on how the object is created.
- If object is created without passing arguments parameterless ctor gets called.
- If object is created by passing arguments then parameterized ctor gets called.

Lab Work

- Arrays
    - take input for primitive types using for each and using tradational for.
    - create objects and take input for reference types using for each and using tradational for.
    - spot the difference between them.
- Assignment
- initializerd demo