

Agenda

- Array
- Variable Arity/Argument Method
- Method Arguments
 - pass by value and reference
- Method Overloading
- final Keyword
- static Keyword
- ~~Singleton Design Pattern~~

Method Overloading

- Defining methods with same name but different arguments(signature) is called as method overloading
- Arguments can differ in one of the following ways
 1. No of parameters should be different
 2. If no of parameters are same then their type of parameters should be different
 3. If no and type are same then the order of parameters should be different
- Count (no of parameters)

```
static int multiply(int a, int b) {  
    return a * b;  
}  
static int multiply(int a, int b, int c) {  
    return a * b * c;  
}
```

- type of parameter

```
static int square(int x) {  
    return x * x;  
}  
static double square(double x) {  
    return x * x;  
}
```

- Order of parameters

```
static double divide(int a, double b) {  
    return a / b;  
}  
static double divide(double a, int b) {  
    return a / b;  
}
```

- Note that return type is NOT considered in method overloading.

Variable Arity/Argument Method

- It is a method which can take variable no of arguments.
- We can also pass array to this method.
- If we want to pass different types of variables to this arity method then we can use the object as the type.

Method Arguments

- In Java, primitive values are passed by value and objects are passed by reference.
- Pass by reference stores address of the object. Changes done in called method are available in calling method.
- Pass by value -- Creates copy of the variable. Changes done in called method are not available in calling method.
- Pass by reference for value/primitive types can be simulated using array.

final

- In Java, const is reserved word, but not used.
- Java has final keyword instead.
- It can be used for
 - variables
 - fields
 - methods
 - class
- if variables and fields are made final, they cannot be modified after initialization.
- final fields of the class must be initialized using any of the following below
 - field initializer
 - object initializer
 - constructor
- final methods cannot be overridden, final class cannot be extended(we will see at the time of inheritance)

static Keyword

- In OOP, static means "shared" i.e. static members belong to the class (not object) and shared by all objects of the class.
- Static members are called as "class members"; whereas non-static members are called as "instance members".
- In Java, static keyword is used for
 - 1. static fields

- 2. static methods
- 3. static block
- 4. static import
- Note that, static local variables cannot be created in Java.

1. static Fields

- Copies of non-static/instance fields are created one for each object.
- Single copy of the static/class field is created (in method area) and is shared by all objects of the class.
- Can be initialized by static field initializer or static block.
- Accessible in static as well as non-static methods of the class.
- Can be accessed by class name or object name outside the class (if not private). However, accessing via object name is misleading (avoid it).
- eg :
 - Integer.SIZE
- Similar to field initializer, static fields can be initialized at declaration.

2. Static methods

- These Methods can be called from outside the class (if not private) using class name or object name. However, accessing via object name is misleading (avoid it).
- When we need to call a method without creating object, then make such methods as static.
- Since static methods are designed to be called on class name, they do not have "this" reference. Hence, they cannot access non-static members in the static method (directly), However, we can access them on an object reference if created inside them.
- eg:
 - Integer.valueOf(10);
 - Factory Methods -> to create object of the class

static Field Initializer

- Similar to field initializer, static fields can be initialized at declaration.

```
static double roi = 5000.0;  
// static final field -- constant  
static final double PI = 3.142;
```

static Initializer Block

- Like Object/Instance initializer block, a class can have any number of static initialization blocks, and they can appear anywhere in the class body.
- Static initialization blocks are executed in the order their declaration in the class.
- A static block is executed only once when a class is loaded in JVM.