Variables and methods

Non-Static variables

- **Local/method local Variables**:
  - A variable defined within a block or method or constructor is called local variable/ method local variables.
  - These variable are created on entry in method and destroyed method exits
  - The scope of these variables exists only within the method/ block in which the variable is declared
  - Initialization of Local Variable is Mandatory. Un-initialized variables produce error in java

Non-Static variables

- **Method local Variables**:

Ex:

```
int  add(int a, int b) {
    int sum = 0;
    // Sum is local variable to method
    sum = a + b;
}
```

Non-Static variables

- **Instance Variables:**
  - Instance variables are non-static variables and are declared in a class outside any method, constructor and block.
  - As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
  - Access specifiers for instance variables define where they can be accessed
  - Initialization of Instance Variable is not Mandatory. They get initialized with default values
  - Instance variable can be accessed only by creating objects.

Non-Static variables

- **Instance Variables example:**

Ex.

```
class Employee {
    private int empNo;
    // empNo is instance variable of class Employee
}
```

Static variables

- **Static Variables (class vars)**
  - A single copy of the static variable is created and shared among all objects at a class level.
  - Static variables are, essentially, global variables. All instances of the class share the same static variable.
  - We can create static variables at class-level only. static block and static variables are executed in order they are present in a program.
  - Initialized to their default values(eg int to 0 ,double to 0.0, boolean to false,ref to null)
  - For static data members(class variables) memory allocated only once @ class loading time.

Static methods

- **Static methods/ Class methods**
  - Static methods are stored in special memory area -- method area (meta space)
  - Static methods are used by ClassName.method()

Ex.
```
class Sample {
    public static int count = 0;
    // count is static variable of class Sample
    // printCount is static method of class Sample
    public static void printCount() {
        System.out.println(count);
     }
    }
```

Static variables vs Instance variables

Static variable	Instance variable
Static variables can be accessed using **class name**	Instance variables must be accessed using instance/object of a class
Static variables can be accessed by **static and non static methods**	Instance variables **cannot** be accessed inside a **static method**.
Static variables are **shared** among all instances of a class.	Instance variables are **specific** to that instance of a class.

Static methods and Instance methods

Static methods	Instance methods
It doesn't require an object of the class. Ex. **main** method, parseInt method	It requires an **object of the class**. Ex. **equals**, hashcode, toString methods of Object
It can access only the **static attribute** of a class.	It can access **all attributes** of a class.
The method is only accessed by **class name.** Ex. ClassName.methodname();	The methods can be **accessed only using object reference**. Ex. objectReference.methodName();

final instance variables

- **final instance Variables**
  - final variable value can not be changed after initialization
  - final variable can be initialized where it is declared or in constructors
  - Static final variables are treated as constants and they can be access with class name. They should be named with all letters CAPs.

Ex.
```
class Sample {
public final int speed = 90;
// count is final variable of class Sample
public static final double PI = 3.14;
// count is final static variable of class Sample
}
```

Demo Programs

- Demo for static methods
- Demo for instance methods