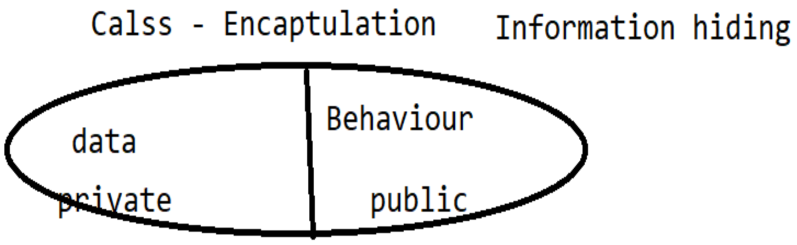
Lesson-3

Class & Objects

Day-1 – Must know the following

1. How to create a class
2. How to add data members/attributes/instance fields/properties
3. How to create a constructor
4. Default constructor, Parameterized constructor and its rules
5. Access Modifiers
6. Constructor Overloading
7. How to create methods / behavior / instance methods
8. this keyword
9. How to create an object using new keyword, assigning objects
10. Array of objects creation
11. Processing the collection of objects to perform some computations
12. Accessing methods
13. Getters and setters
14. Date , Calander Local Date APIs [Mutable and Immutable APIs]
15. Immutable class
16. Record class

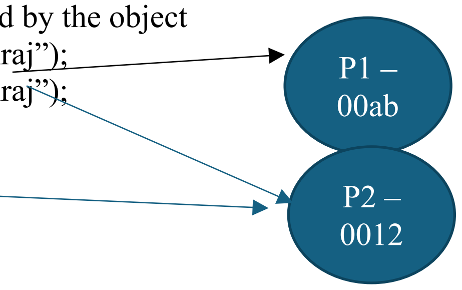
Class -- > data and methods (Template / blueprint)
    Data → members, properties, Instance fields, attributes
    Methods → Service / Behavior / Member function/ Instance methods
Object→ instances of a class / realization of the class template
Security – Encapsulate the data and behavior under a class type



```
class Person{
       private String fname;
       private String lname;
      Person(String fname, String lname)
   {
    this.fname = fname;
    this.laname = lname;
 }
```

```
Main(){
// Status of the object – Current value should by the object
Person p1 = new Person("Renuka","Mohanraj");
Person p2 = new Person("Renuka","Mohanraj");
p1 = new Person("Renu","Mohan");
Person p3 = p2;

}
```



Points about Constructors

Name Matching: The constructor's name must exactly match the class name.

No Return Type: Constructors do not have a return type, not even void.

Implicit Constructor: If no constructor is defined, Java provides a default no-argument constructor.

Multiple Constructors: A class can have multiple constructors, each with different parameter lists (constructor overloading).

Access Modifiers: Constructors can use any access modifier: public, protected, private, or default (package-private).

this(): The this() keyword calls another constructor in the same class. They must be the first statement in a constructor.

this: The this keyword can be used to call the instance fields and methods.

Initialization: Constructors are used to initialize objects when they are created.

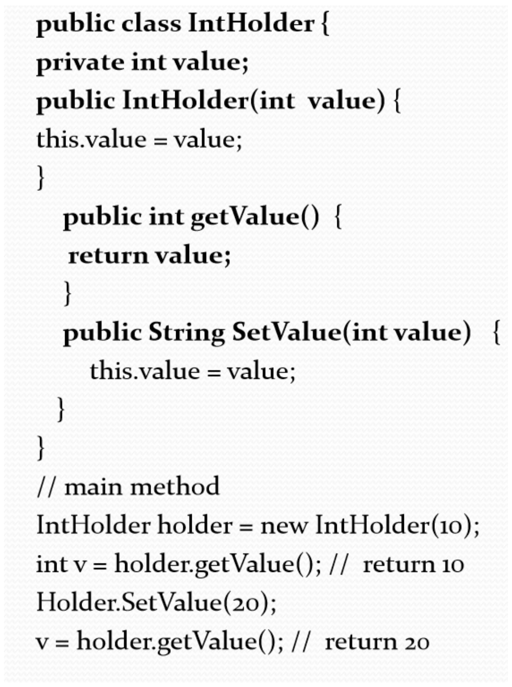
//Static Context: Constructors cannot be static, abstract, or final.

Object Creation: Constructors are called implicitly when an object is created using the new keyword.

Date and Gregorian Calanders are Mutable.
Date→ Current date
```
Date today = new Date();
Date sday = new Date(10005);
Sday.getTime()
```

Gregorian Calander months starts from 0 – 11
0 – January
11 – December

Mutable class



```
public class IntHolder {
private int value;
public IntHolder(int  value) {
this.value = value;
}
   public int getValue()  {
    return value;
   }
   public String SetValue(int value)   {
      this.value = value;
  }
}
// main method
IntHolder holder = new IntHolder(10);
int v = holder.getValue(); //  return 10
Holder.SetValue(20);
v = holder.getValue(); //  return 20
```

Record Class

A record declaration specifies in a header a description of its contents; the appropriate accessors, constructor, equals, hashCode, and toString methods are created automatically. A record's fields are final because the class is intended to serve as a simple "data carrier".

record Rectangle(double length, double width) { }

Rectangle r = new Rectangle(4,5);

- A private final field with the same name and declared type as the record component. This field is sometimes referred to as a *component field*.
- Getters are length() and width().

Technical Terms used with Record class

public record Person(String name, int age) { } // Record class

In this case, Java automatically creates a canonical constructor equivalent to:

```
public Person(String name, int age) {
   this.name = name;
   this.age = age;
}
```

Explicit Canonical Constructor (Compact Form):

```
public record Person(String name, int age) {
   public Person {
      if (age < 0) {
         throw new IllegalArgumentException("Age must be non-negative");
      }
   }
}
```

Here, the explicit canonical constructor (using the compact form) allows you to add custom validation while still automatically assigning the record components.

Final Keyword

Final – fields → You cannot alter the value (Constant)
Final – Class → You cannot do inheritance (Lesson-5)
Final – methods → You cannot override (Lesson-5)

Enum --> Better alternative for public static final constants and to get compile time verification.

Boxed Primitive

Take care of initialization, do the null check to avoid runtime exception

String x = "Java"; -- > String pool memory

String a = new String("Java"); → heap

Swap values

Int x = 10;

Int y = 20;

Int temp = x;

Y = x;

X= temp;

Memory Allocations

```
public class MemoryAllocation {
    int data = 50; // Instance field(heap) not a local variable
    static int int_Count = 0; // static field
    public static void main(String[] args) {
        int x=10; // Local variable
       String s1 = "Java";
       String s2 = new String("Program");
```

```
        DateAPIsMutability ob = new DateAPIsMutability(); // ob
        System.out.println(ob.getD3()); // Calling instance method
        printdoubled(x); // Calling a static method
    }
    public static void printdoubled(int a){ // Local variable a
        System.out.println(a * 2);
    }
}
```

- **Stack Memory:**
  - Local Variables:
    - x = 10 (in main() method)
    - a (in printdoubled() method)
  - Method Calls:
    - main(String[] args)
    - ob.getD3()
    - printdoubled(int a)
  - References:
    - ob (reference to DateAPIsMutability object)
    - s1 (reference to String object)
    - s2 (reference to String object)

- **Heap Memory:**
  - Instance Fields:
    - data = 50 (for each Memory Allocation instance)
  - Instance of DateAPIsMutability class (pointed to by ob)

  - New String object "Program" created by new String("Program"), referenced by s2

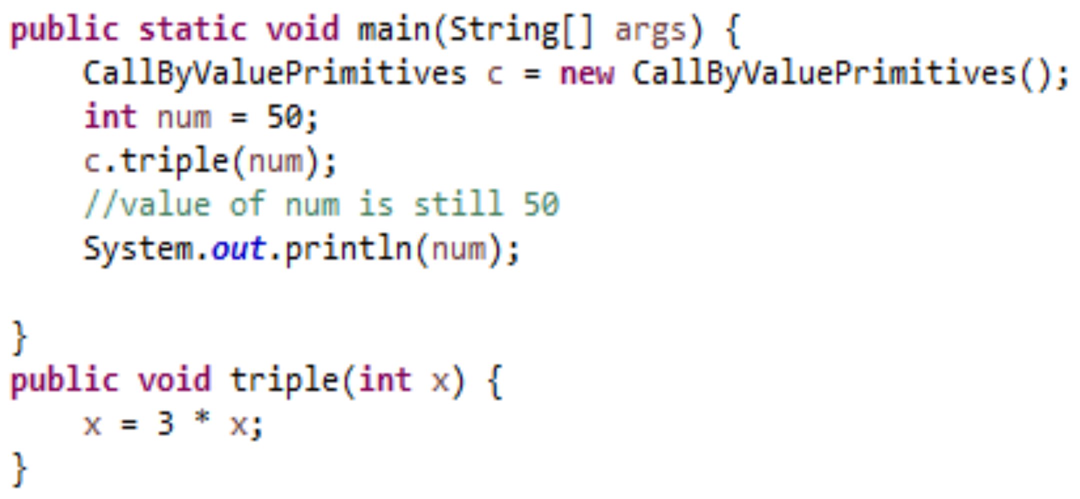
  - **String Pool:**

    - String literals "Java" stored in the string pool referenced by s1

- **Meta Space**

  - **Static Fields:**
    - static int_Count = 0;

Sample Code and its Memory allocation



```
public static void main(String[] args) {
    CallByValuePrimitives c = new CallByValuePrimitives();
    int num = 50;
    c.triple(num);
    //value of num is still 50
    System.out.println(num);

}
public void triple(int x) {
    x = 3 * x;
}
```

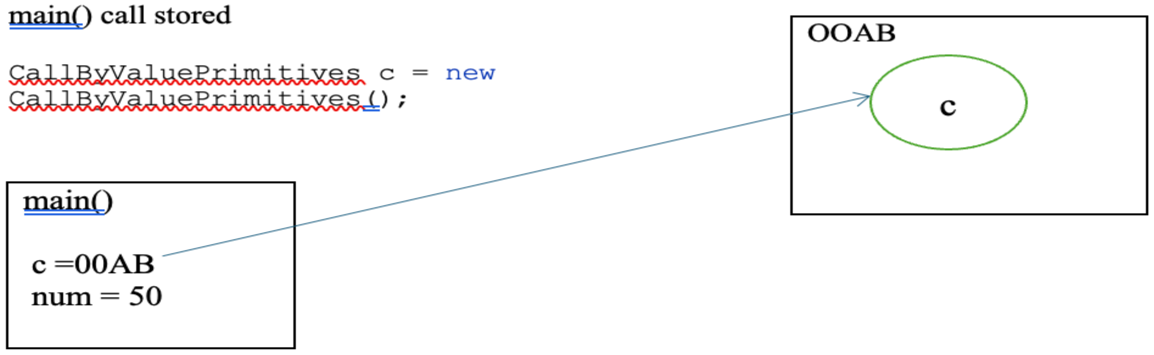
Stack Memory [Stores Primitives, method calls and Address of objects references, local var]

Heap Memory [Stores the Objects]

Step: 1

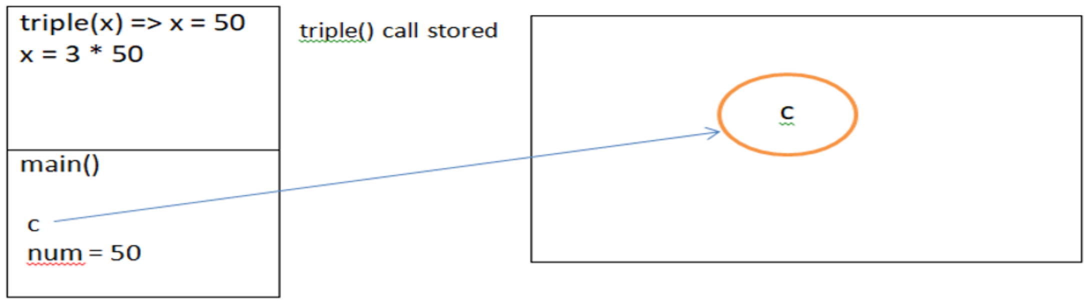
main() call stored

```
CallByValuePrimitives c = new
CallByValuePrimitives();
```



```
c.triple(num);
```

Step : 2



Step 3 : triple() call is removed from the stack. Also tripled x value is not in the memory. So changes does not affect the original

The JVM's garbage collector (GC) is highly optimized and manages memory efficiently without manual intervention. Avoid System.gc() in most cases.

Avoid Memory Leakages

- Avoid keeping unnecessary **static references**, as they persist throughout the application's lifecycle.

- Make Null or remove references to objects that are no longer needed.

- Use **try-with-resources** (AutoCloseable) to ensure resources like files, database connections, are closed automatically, if you are not using try-with-resources close it manually.

Call by Value vs Call by Reference in Java

Call by Value/pass by value:

- **Definition:** When a method is called, a copy of the actual parameter's value is passed to the method.
- **Behavior in Java:** Java always uses call by value. This means that the method receives a copy of the variable's value.
- **Effect on Primitive Types:** Changes to the parameter inside the method do not affect the original variable.
- **Effect on Objects:** The reference to the object is passed by value, meaning the method receives a copy of the reference. Changes to the object's fields by calling it's methods will affect the original object, but reassigning the reference will not.

Call by Reference/pass by reference:

- **Definition:** The method receives a reference to the actual parameter, allowing it to modify the original variable directly.

- **Java's Approach:** Java does not support call by reference. Even with objects, the reference is passed by value, not the actual object.

Homework –Hints

```
class Customer{
 Fname;
Lanmae;
Ssn;
Address badd; // has a relationship
Address sadd;

}
class Address{
  street, city, state and zip
}
class Main ()
{
Customer c1 = new Customer(“fn”,”nl”,”ssn”);
Customer c2 = new Customer(“fn”,”nl”,”ssn”);
Customer c3 = new Customer(“fn”,”nl”,”ssn”);

Address a1 = new …..

C1.setBadd(a1);
}
```

Apply your creativity to solve the Event problem.

Example Task 3 you can choose your way of nice formatting must understand by the user.

Another Suggestion: You can go with only one class. Or you can create an Event class and Test class. All is up to you.

Inside a class

1. Static block
   Static{
   }
2. Static initialization
   Static int x = 10;

 Will initialized only once for class level

3. Instance fields
   int k = 10;
4. Instance methods
5. Instance block / Anonymous block

```
{
}
```
Will execute before the constructor execution

6. Constructor block

   Instance fields and instance block have same priority, in which order you specified

Can have one or more static, instance blocks

```
public class Baby {

   // First Static block
   static {
      String x = "Baby Class";
      System.out.println("1.Static Fields String x = " + x);
   }
   // Second static block
   static {
        int rn = getNumber();
        System.out.println("2. Static Fields Integer rn " + rn);
      }
   // static method
   public static int getNumber() {
      // Random r = new Random();
      // int a = r.nextInt();
        return new Random().nextInt();
      }
   // Instance Fields
      private String name;
      private String gender;
   // Default Constructor
      public Baby() {
        System.out.println("I am exited to see my Baby");
      }
      // Instance block - Anonymous block - will execute for each constructor
      {
        System.out.println("Hearing Baby Crying - Happy");
      }
      public Baby(String name, String gender) {
        this.name = name;
        this.gender = gender;
        System.out.println("I am happy to see " + name + " " + gender + " baby");
      }
}
```

```
public class TestBaby {

   public static void main(String[] args) {
      new Baby(); // Nothing wrong, just object created in the memory
      new Baby("Pinky","Female");
   }
}
```