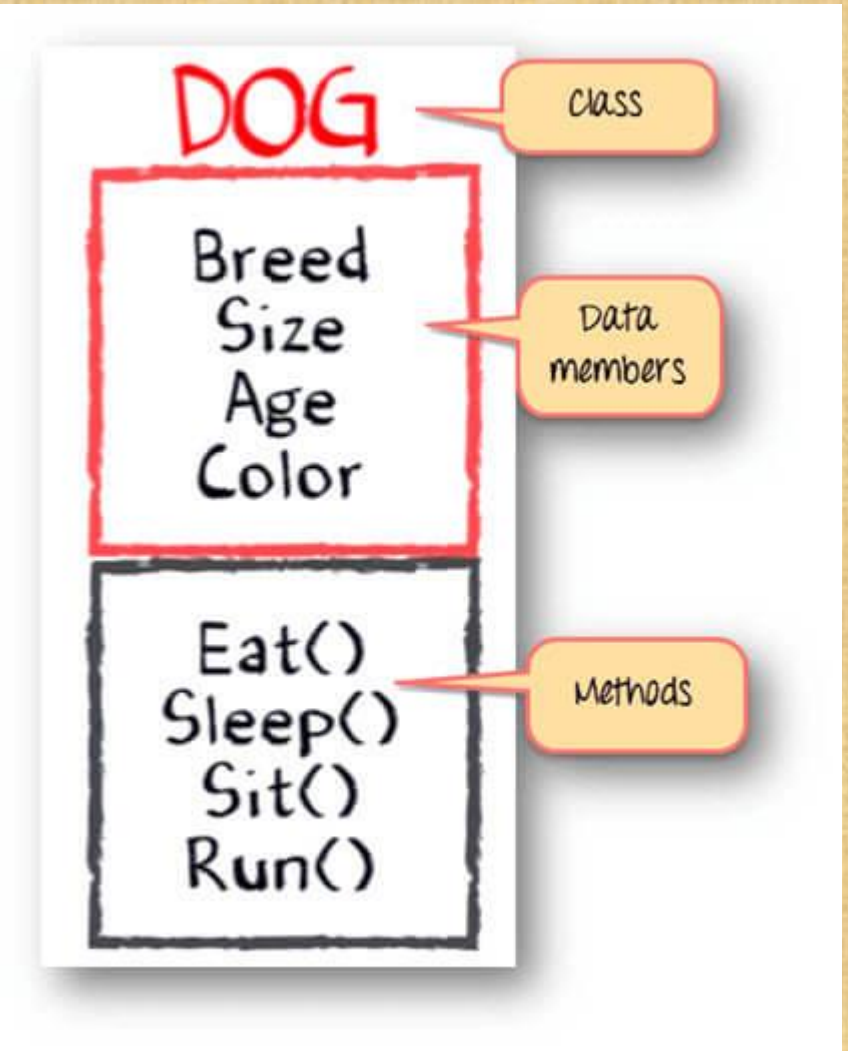
Java: Class, Object and Method Part-1

Prepared By : Minal Shah

Assistant Professor, CSPIT CE, CHARUSAT



Class

- We know the simple form of class
- Till now it is used to encapsulate the main method.
- The actual from of Class is far more powerful than what is presented so far.
- A class is
  - User defined
  - Blueprint or template to create objects
  - Combination of state and behavior of any real world entity

Class Example: Panda

- Panda – attributes
  - Color
  - Weight
- Behavior
  - Eating
  - Sleeping

sleeping

eating

```java
class Panda{
    String colour = "black"; // instance variable
    String weight = "100Kg";

    void eat()  {
        int count = 10;  //local variable
        System.out.println("Panda is eating");
    }
    void sleep() // instance method
    {
        System.out.println("Panda is sleeping");
    }
}
class PandaDemo{
    public static void main(String[] args)
    {
        Panda p1; // reference
        p1 = new Panda(); // Object

        p1.eat();
        p1.sleep();

        Panda p2,p3;
        p2 = new Panda();
        p3 = new Panda();

        p2.eat();
        p3.sleep();
    }
}
```

Use new keyword to create an object and allocate memory

Output

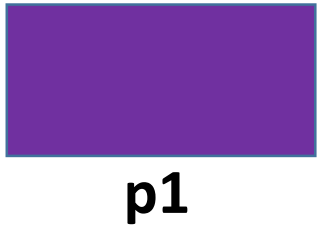
```
E:\One Drive\Charotar University\Programs>javac PandaDemo.java

E:\One Drive\Charotar University\Programs>java PandaDemo
Panda is eating
Panda is sleeping
Panda is eating
Panda is sleeping
```

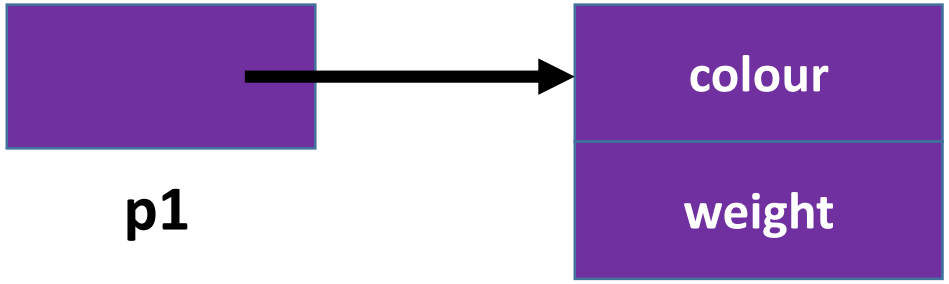
Creating an object

Statement **Effect**

Panda p1;



P1 = new Panda();



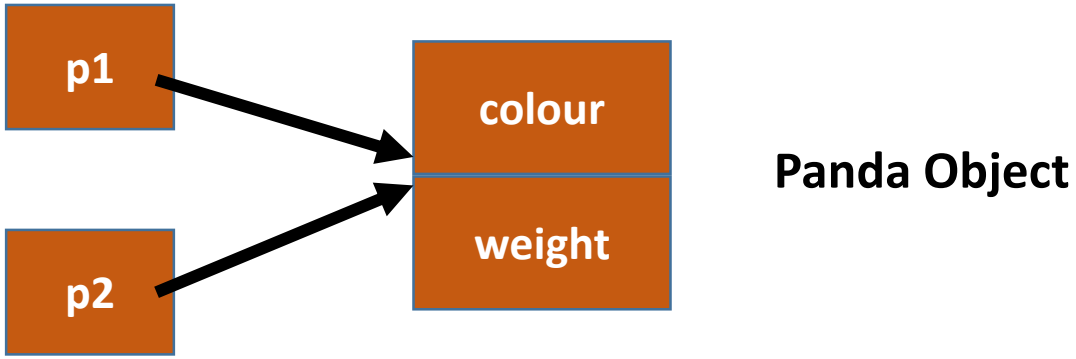
Assigning Object Reference Variables

- Panda p1 = new Panda();
- Panda p2 = p1;

- Here p1 and p2 refers to same object
- p2 = p1 doesn't allocate memory

- It simply makes p2 refer to the same object p1

- Changes made to object p2 will affect p1.



- So if set p1 = null;
- p2 will still point to original object

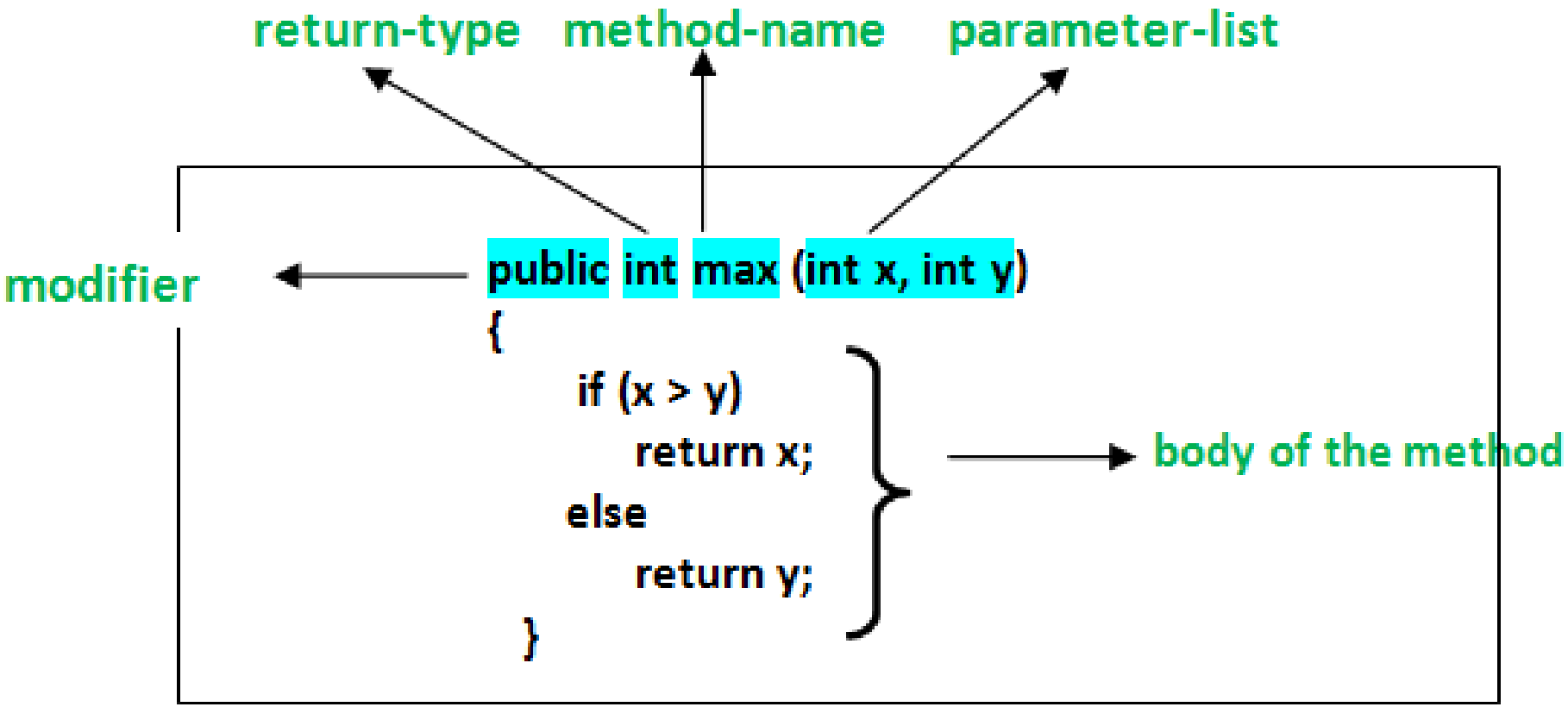
Operator *new* and Object reference

- The **new** dynamically allocates memory to an object during run time
- So the advantage gained here is, program can create as many as required during execution of program
- When you assign one object reference variables to another object reference variable, you are not creating a copy of the object, you are only creating copy of the reference.

Exercise

- Write a program to create a class named Rectangle. Define two attributes width and length. Define an instance method to calculate the area of a rectangle and return it. Define another class having main method and create different objects of Rectangle class and test methods by calling them.
- What is anonymous object?

Method Signature



Valid java main method signature

- public static void main(String[] args)
- public static void main(String []args)
- public static void main(String args[])
- public static void main(String... args)
- static public void main(String[] args)
- public static final void main(String[] a)
- final public static void main(String[] a)
- final strictfp public static void main(String[] a)

Constructor

- It is tedious to initialize all the variables in a class each time an instance is created
- Simpler way to initialize variables is at the time of object is created
- This automatic initialization is performed through constructor
- A *constructor* initializes an object immediately upon creation
- Characteristics of *Constructor*
  - Same name as method name
  - Syntactically similar to method
  - Once defined, it is automatically called when object is created, before *new* completed
  - No return type – not even void [This is because the implicit return type of a class' constructor is the class type itself.]

Constructors in Java

- Types of constructors
  - Default Constructor
  - Parameterized Constructor
- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

```java
class Circle{
        int  raduis;
        int  diameter;
        Circle(){
            raduis = 10;
            diameter = raduis * 2;

            System.out.println("Inside Constructor");
       }

       public void display(){
           System.out.println("Radius === "+raduis);
           System.out.println("Diameter === "+diameter);
      }
  }
 class CircleDemo
{
     public static void main(String[] args)
     {
         Circle c1 = new Circle();
         c1.display();
     }
}
```

Constructor Overloading

Beginnersbook.com

```
public class Demo {
   Demo( ) {
   ..
   }
   Demo(String s) {
   ...
   }
   Demo(int i) {
   ...
   }
.....
}
```

Three overloaded constructors - They must have different Parameters list

Method Overloading

- Two or more methods within same class or parent class and derived class – that shares same name but different method signature
- One of the way java supports Polymorphism - One Interface, multiple methods
- How Java determine – which version of method to be called?
  - Based on type and/or number of arguments
  - Overloaded methods must differ in the type and/or parameters
  - Same signature but different return type – alone is insufficient to distinguish two versions of method
- Example – sum method with different signature [OverloadDemo.java]

PASSING OBJECTS As a Parameter

- Objects may be passed to methods
- Example : ObjectAsParameter.java

Argument Passing

- Two ways that computer language can pass an argument
  - Call By Value
  - Call By Reference
- Call By Value
  - Copies value of an argument into the formal parameter
  - So changes made to parameter of subroutine have no effect
- Call By Reference
  - Reference of an argument is passed to parameter
  - Reference is used to access actual argument in call
  - Changes made to parameter will affect the argument used to call subroutine

Argument Passing

- As Java used call by value to pass all arguments – it depends on primitive type or reference type
- Example : CallByDemo.java

Exercise

- Run the following programs and check output
  - CopyObject.java
  - RecTest.java

Recursion

- Something that defines itself
- Recursive versions may execute a bit more slowly than iterative equivalent
- The main advantage
  - Can be used to create clearer and simpler versions of several algorithms than iterative one
  - QuickSort example
  - Some types of AI-related algorithms are most easily implemented using recursive solutions
- Example : RecTest.java

Introducing final

- Prevents its content being modified – constant
- You can initialize it in two ways
  - At the time declaration
  - In constructor
- Examples

Java Final Keyword

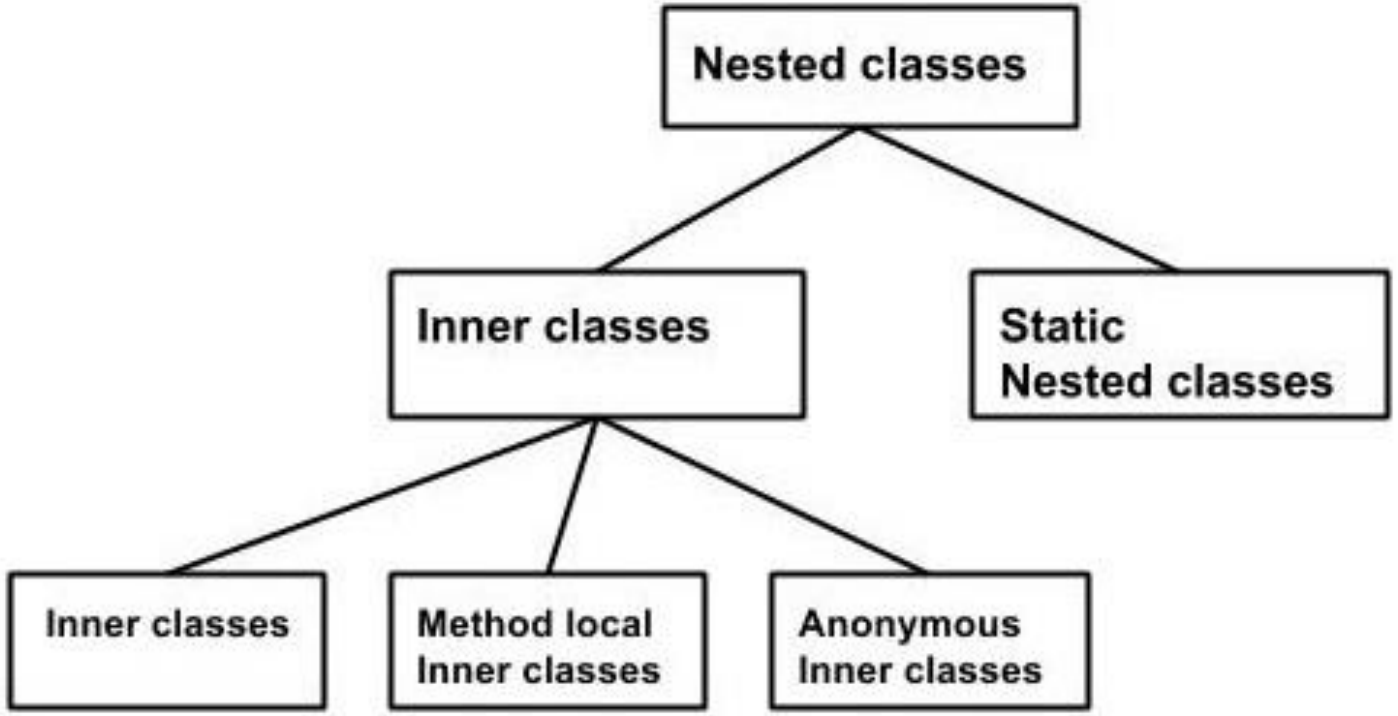
- Stop value change
- Stop method Overriding
- Stop Inheritance

Trying to change final value results in compile time error

```
12  class Car {
13
14      final int speedlimit = 120;//final variable
15
16      void run() {
            speedlimit = 400;
18      }
19
20      public static void main(String args[]) {
21          Car obj = new Car();
22          obj.run();
23      }
24  }
25
```

Introducing Inner and Nested Class

- Class within class – nested class



1. Inner class [Non - static nested class]

- We use inner class to logically group classes and interfaces in place
- It becomes more readable and maintainable.
- It can access all the members of outer class including private data members.
- Unlike a "regular" class, an inner class can be private or protected.
- If you don't want outside objects to access the inner class, declare the class as private.
- Inner Class: This non-static class is created inside a class but outside a method
- Example :
  - PrivateInnerClassDemo.java
  - MyMainClass.java

Syntax:

```
class OuterClass{
        //code
        class InnerClass{
                //code
        }
}
```

2. Method- local Inner class

- We can write a class within a method – inside a block
- This block is a method body or *for* loop inside method or *if* block
- Like local variables, the scope - is restricted within the method.
- A method-local inner class can be instantiated only within the method where the inner class is defined.
- **Note:** Local Inner classes are not a member of any enclosing classes.
- Example : MethodLocalInnerDemo.java

3. Anonymous Inner Class

- An inner class without name is known as **anonymous inner class**.
- Declare and instantiate at the same time - One object created
- Generally, they are used whenever you need to override the method of a class or an interface.

```
AnonymousInner an_inner = new AnonymousInner() {
    public void my_method() {
     ........
     ........
  }
};
```

Anonymous Inner Class Example

Example:

```
abstract class Pizza{

  abstract void addToppings();

}
```

```
class AnonymousInnerExample{

 public static void main(String[] args){

 Pizza p=new Pizza(){

  void addToppings() {

        System.out.println(“I am enjoying Pizza");

     }

  };

  p.addToppings()   } }
```

Output:

I am enjoying Pizza

Static - Inner class

An inner class can also be static, which means that you can access it without creating an object of the outer class:

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 20;
  }
}
```

```
public class MyMainClass {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
  }
}

// Outputs 20
```

System.arraycopy() in Java

```
public static void arraycopy(Object source_arr, int
sourcePos, Object dest_arr, int destPos, int len)
```

Parameters : source_arr : array to be copied from
sourcePos : starting position in source array from where to copy
dest_arr : array to be copied in
destPos : starting position in destination array, where to copy in
len : total no. of components to be copied.

Garbage Collection

- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

An automatic garbage collector essentially performs two tasks:

- Decides if and when memory needs to be reclaimed
- Finds objects that are no longer needed by the program and reclaims their storage

- we were using free() function in C language and delete() in C++. But, in java it is performed automatically or Manually using System.gc() method.

Finalize() method

- Finalize() is the method of Object class.
- Run command javap java.lang.Object on command prompt
- This method is called just before an object is garbage collected.
- It overrides to release system resources perform clean-up activities and minimize memory leaks.
- Example : FinalizeDemo.java

Finalize() method continued…

- if o1.equals(o2), then o1.hashCode() == o2.hashCode() should always be true.
- If o1.hashCode() == o2.hashCode is true, it doesn’t mean that o1.equals(o2) will be true.

How can object be unreferenced?

1. By assigning null

   Example:

   ```
   Aircraft a = new Aircraft();
   a = null;
   ```

2. By assigning a reference to another

   Example:

   ```
   Aircraft a1 = new Aircraft();
   Aircraft a2 = new Aircraft();
   a2 = a1 //now the first object referred by a1 is ready for garbage collection
   ```

3. By anonymous object

   ```
   new Employee();
   ```

Questions

- What will be the result of compiling following code?

```
public class MyClass {
    public static void main(String args[]) {
        System.out.println("In first main()");
    }
    public static void main(char args[]) {
        System.out.println('a');
    }
}
```

Output: Code will compile correctly and will print In first main()

Find Output

```
class Test {
  int i;
}
class Main {
   public static void main(String args[]) {
     Test t;
     System.out.println(t.i);
}
```

t is just a reference, the object referred by t is not allocated any memory. Unlike C++, in Java all non-primitive objects must be explicitly allocated and these objects are allocated on heap. The following is corrected program.

```java
class demo
{
    int a, b;

    demo()
    {
        a = 10;
        b = 20;
    }

    public void print()
    {
        System.out.println ("a = " + a + " b = " + b + "n");
    }
}

class Test
{

    public static void main(String[] args)
    {
        demo obj1 = new demo();
        demo obj2 = obj1;

        obj1.a += 1;
        obj1.b += 1;

        System.out.println ("values of obj1 : ");
        obj1.print();
        System.out.println ("values of obj2 : ");
        obj2.print();

    }
}
```

values of obj1:
a = 11 b = 21
values of obj2:
a = 11 b = 21

```java
class Test
{
    int a = 1;
    int b = 2;

    Test func(Test obj)
    {
        Test obj3 = new Test();
        obj3 = obj;
        obj3.a = obj.a++ + ++obj.b;
        obj.b = obj.b;
        return obj3;
    }

    public static void main(String[] args)
    {
        Test obj1 = new Test();
        Test obj2 = obj1.func(obj1);

        System.out.println("obj1.a = " + obj1.a + " obj1.b = " + obj1.b);
        System.out.println("obj2.a = " + obj2.a + " obj1.b = " + obj2.b);

    }
}
```

obj1.a = 4 obj1.b = 3
obj2.a = 4 obj2.b = 3

obj1 and obj2 refer to same memory address.

Topic Wise - Video Links available on MS Stream

- https://web.microsoftstream.com/video/5dca645c-7e5e-40d7-9b13-7db9e0efe478
- https://web.microsoftstream.com/video/f675288e-135d-429d-8964-d19230ff880d
- https://web.microsoftstream.com/video/7837708d-5456-4d51-b475-e23187e840dc
- https://web.microsoftstream.com/video/90324834-5fe3-4f9f-a061-b2280af5ffd7
- https://web.microsoftstream.com/video/f959a867-4c9f-42c8-ae1b-9ac7ff4b43e6
- https://web.microsoftstream.com/video/ee28f2f1-9161-4132-834d-a28552986674
- https://web.microsoftstream.com/video/117a36cc-e09a-4b1c-be3e-9cce9c36d5bc
- https://web.microsoftstream.com/video/ab4f8463-2d08-48b8-814e-1b3d18b3ecce
- https://web.microsoftstream.com/video/4acb9808-67ef-4bfc-afb3-30518694ca09
- https://web.microsoftstream.com/video/0d07c659-69fd-4e83-b979-48f1f1bc1344

References

- Java: The Complete Reference, Eleventh Edition by Herbert Schildt
- https://www.tutorialspoint.com/java/
- https://www.javatpoint.com/
- https://www.geeksforgeeks.org/java/
- https://www.javatpoint.com/java-mcq


thank you