
Java Programming
TEK RAJ CHHETRI

Course Objective

- Explain the Java programming environment
- Describe the concepts of programming elements using Java and object-oriented
- programming concepts
- Apply the exception handling and input/output in Java programming
- Apply the event handling, GUI programming using swing, and Java database connectivity

Unit 4: Inheritance and Interface

- Classes, Super classes, and Subclasses
- Polymorphism
- Dynamic Binding
- Final Classes and Methods
- Abstract Classes
- Access Specifies
- Interfaces

Learning Outcome (Unit 6)

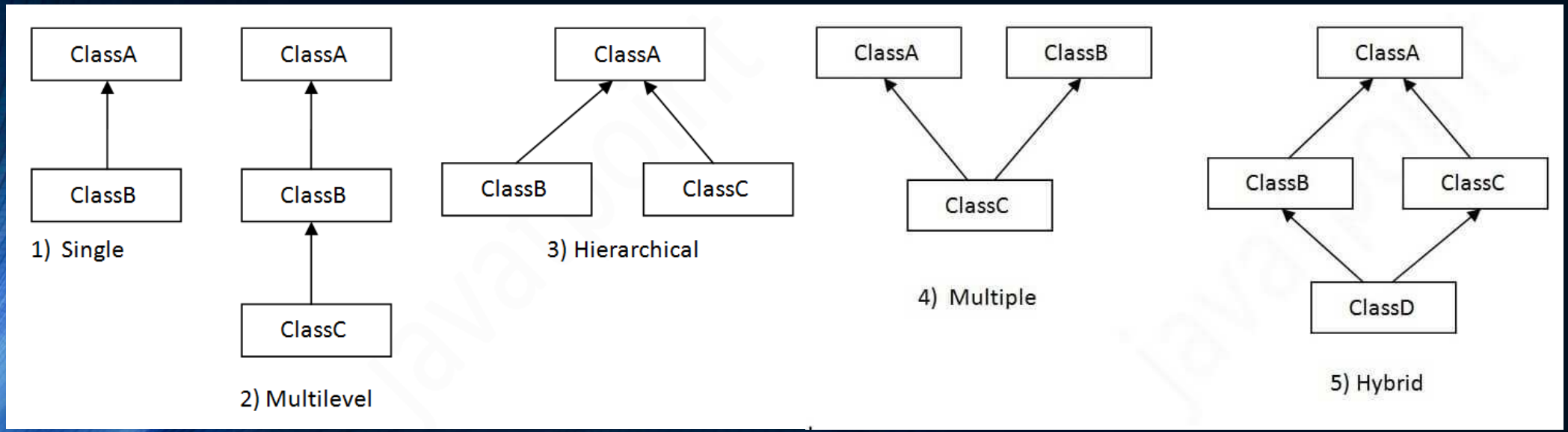
- Develop understanding about inheritance, interface, polymorphism and use of final class and methods and abstract class .
- Able to differentiate between super class and sub class.
- Able to implement dynamic binding, inheritance, use of final class and methods and abstract class.

Classes, Super classes, and Subclasses

- Inheritance is the mechanism where the child inherits the properties and behavior of its parent.
- **Syntax:**
  - class Subclass-name extends Superclass-name
  - {
  - //methods and fields
  - }
- Eg:

```
1. class A{
2.
3. }
4. Class B extends A{
5.
6. }
```

- Sub-class is the class that inherit other class properties. Also called derived class.
- Super class is the class from which sub class inherits the features. It is also called base / parent class.
- In the preceding example A represents super class and B sub class, inherits A's property by extending i.e. using extends keyword.
- **Inheritance Types**:



- **Example Inheritance:**

```
1. class Employee{
2.  float salary=40000;
3. }
4. class SimpleInheritance extends Employee{
5.  int bonus=10000;
6.  public static void main(String args[]){
7.   SimpleInheritance p=new SimpleInheritance();
8.   System.out.println("SimpleInheritance salary is:"+p.salary);
9.   System.out.println("Bonus of SimpleInheritance is:"+p.bonus);
10. }
11. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>javac SimpleInheritance.java

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java SimpleInheritance
SimpleInheritance salary is:40000.0
Bonus of SimpleInheritance is:10000
```

- **Single level Inheritance:**

```
1. /*
2.     * This example is from javatpoint.com
3. */
4. class Animal{
5.     void eat(){
6.         System.out.println("eating...");
7.     }
8. }
9. class Dog extends Animal{
```

```
1.      void bark(){
2.              System.out.println("barking...");
3.      }
4.  }
5.  class SingleInheritance{
6.      public static void main(String args[]){
7.              Dog d=new Dog();
8.              d.bark();
9.              d.eat();
10.     }
11. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>javac SingleInheritance.java

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java SingleInheritance
barking...
eating...
```

- **Multi level Inheritance:**

```
//This program is taken from javatpoint.com
class Animal{
        void eat(){
                System.out.println("eating...");
        }
}
class Dog extends Animal{
        void bark(){
                System.out.println("barking...");
        }
}
```

```
class BabyDog extends Dog{
        void weep(){
                System.out.println("weeping...");
        }
}
class MultiLevelInheritance{
        public static void main(String args[]){
                BabyDog d=new BabyDog();
                d.weep();
                d.bark();
                d.eat();
        }
}
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>javac MultiLevelInheritance.java

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java MultiLevelInheritance
weeping...
barking...
eating...
```

- **Hierarchical Inheritance:**

```
//This program is taken from javatpoint.com
class Animal{
        void eat(){
                System.out.println("eating...");
        }
}
class Dog extends Animal{
        void bark(){
                System.out.println("barking...");
        }
}
```

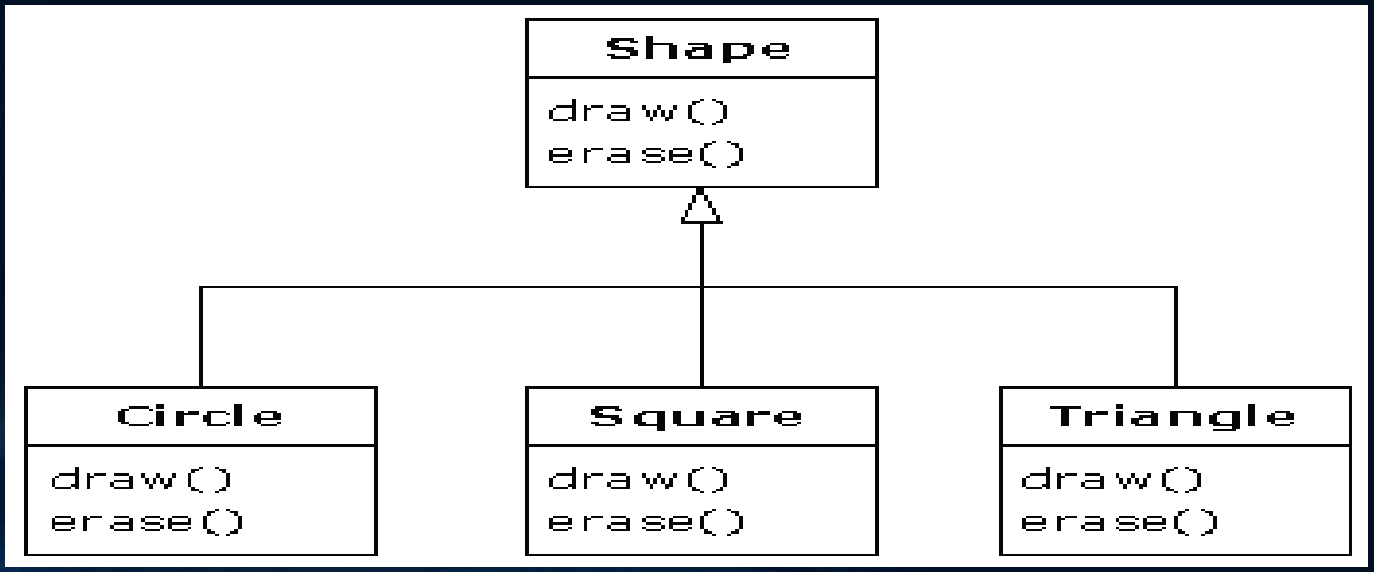
- Multiple and hybrid inheritance are not supported by java.

Polymorphism

- **Super**:
- The super keyword is used to access the super class constructor.
- Syntax for accessing super class constructor:
  - super(arg-list)
  - arg-list : specifies the parameter of super class constructors.
- Note: It must be the first statement to be execute.
- The other use is to access members of the super class that are hidden by subclass.

```
class Child extends Parent{
    int i;

    Child(int a, int b){
        super(); //call to Parent class constructor
        super.i = a;
        i = b;
    }

    void show(){
        System.out.println("Parent i="+super.i);
        System.out.println("Parent i="+i);
    }

}
```

- **Polymorphism**:
- Many forms
- Are two types:
  1. Compile time: implemented using method overloading
  2. Run time: implemented using dynamic dispatch method



- **Compile time Polymorphism / Method Overloading**:
- Method overloading is features that allows class to have methods with same name and different parameter.

```
void add(int a, int b){
    System.out.println("Result1 = "+ (a+b));
}

void add(int a, int b, int c){
    System.out.println("Result2 = "+ (a+b+c));
}
```

- **Eg:**

```
1.  //java program to demonstrate compile time (static) polymorphism
2.  class StaticPolymorphism{
3.          void add(int a, int b){
4.                  System.out.println("Result1 = "+ (a+b));
5.          }
6.          void add(int a, int b, int c){
7.                  System.out.println("Result2 = "+ (a+b+c));
8.          }
9.  }
10. class CompilePolymorphism{
11.         public static void main(String[] args) {
12.                 StaticPolymorphism compile = new StaticPolymorphism();
13.                 compile.add(3,4);
14.                 compile.add(4,5,2);
15.         }
16. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>javac CompilePolymorphism.java

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java CompilePolymorphism
Result1 = 7
Result2 = 11

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>
```

- **Method Overriding:**
- When a method in sub class has same name and type signature as super class, it overrides the super class one.
-

```java
//java program to demonstrate method overrirding
class A{
    void show(){
        System.out.println("This is super class");
    }
}
class B extends A{
    void show(){
        System.out.println("This is child class");
    }
}
class Overriding{
    public static void main(String[] args) {
        B b = new B();
        b.show();
    }
}
```

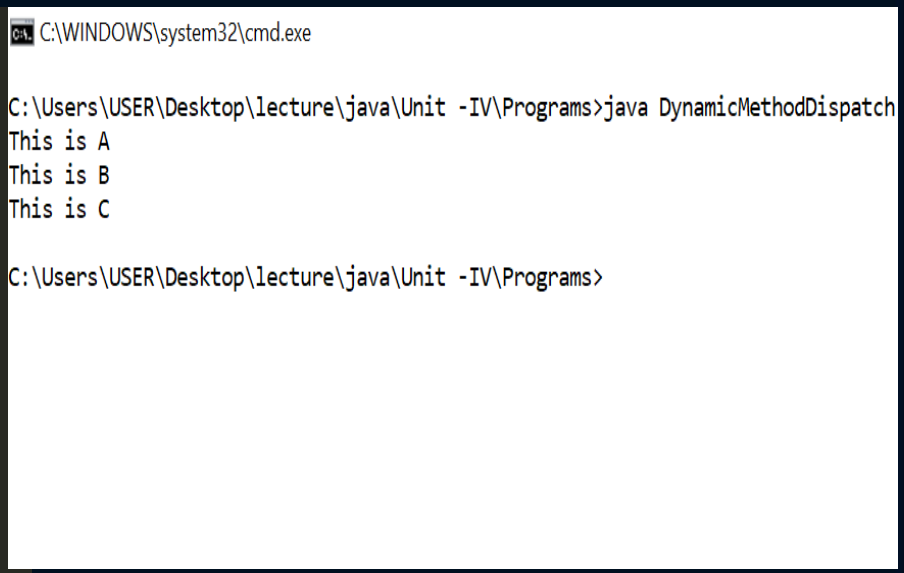
```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>javac Overriding.java

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java Overriding
This is child class

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>
```

- Use super to access override method

```
class A{
    void show(){
        System.out.println("This is super class");
    }
}
class B extends A{
    void show(){
        super.show();
        System.out.println("This is child class");
    }
}
class Overriding{
    public static void main(String[] args) {
        B b = new B();
        b.show();
    }
}
```

- **Run time Polymorphism / Dynamic method Dispatch:**
- Dynamic method dispatch is a mechanism by which call to overridden method is resolved at runtime.

```java
//java program to demonstrate run time polymorphism
class A{
    void show(){
        System.out.println("This is A");
    }
}
class B extends A{
    void show(){
        System.out.println("This is B");
    }
}
class C extends A{
    void show(){
        System.out.println("This is C");
    }
}
class DynamicMethodDispatch{
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        A r;//obtain a reference of type A
        r = a; //refer to b object
        r.show(); //call A classs show() method
        r = b;
        r.show();
        r = c;
        r.show();
    }
}
```



Dynamic Binding

- Binding which can be resolved at compile time is called static binding.
- Binding of static, final and private methods are always static.
- Binding at runtime is called dynamic binding.
  - Eg: Method overriding.

```
/*        @ Static Binding Example
          @Source: geeksforgeeks.com
*/
public class StaticBinding
{
   public static class superclass
   {
      static void print()
      {
         System.out.println("print in superclass.");
      }
   }
   public static class subclass extends superclass
   {
```

```
    static void print()
    {
       System.out.println("print in subclass.");
    }
}
 public static void main(String[] args)
{
    superclass A = new superclass();
    superclass B = new subclass();
    A.print();
    B.print();
}
}
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java StaticBinding
print in superclass.
print in superclass.

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>
```

Final Classes and Methods

- The final when used with method prevents it from overriding.

```
class A{
    final void show(){
        System.out.println("this is final method");
    }
}

class B extends A{
    void show(){ // Error Can't override
        System.out.println("Overriding final method show():: class A");
    }
}
```

- The final when used with class prevents inheritance.

```
final class A{
    //...
}

class B extends A{ //Error: can't inherit A
    // ...
}
```

Abstract Classes

- Let's say you want to define parent class Shapes with general structure for its child.

- Every child that inherits Shapes should have method area with their own implementation.

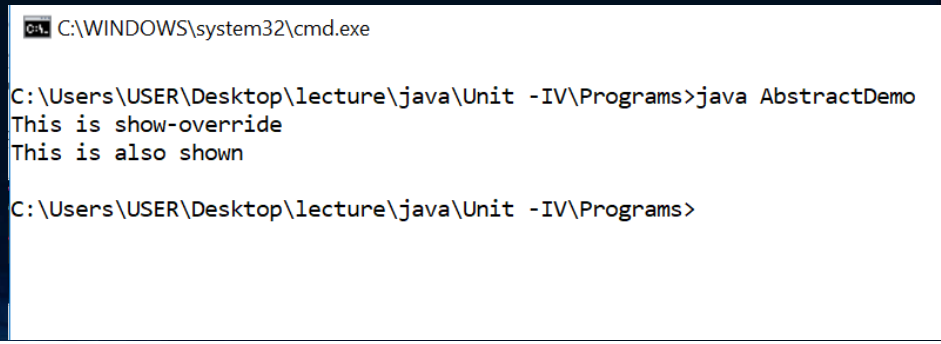
How to do this?

The answer to this is abstract class and methods.

- The class can be declared abstract by placing abstract keyword in front of the class name.
- The abstract class can have abstract method as well as concrete method.
- The abstract method must be overridden by subclass.
- Syntax Abstract method:

  abstract method-name(parameter-list); //no implementation
- The abstract class is used to define general structure that is implemented by sub class.
- Any sub class of an abstract class must implement all abstract methods or itself be abstract class.
- Abstract constructors and abstract static methods are not allowed.

```
1.  abstract class A{
2.      abstract void show();
3.      void showthis(){ //concrete methods are also allowed in abstract class
4.          System.out.println("This is also shown");
5.      }
6.  }
7.  class B extends A {
8.      void show(){
9.          System.out.println("This is show-override");
10.     }
11. }
12. class AbstractDemo{
13.     public static void main(String[] args) {
14.         B b = new B();
15.         b.show();
16.         b.showthis();
17.     }
18. }
```



Access Specifier

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Interface

- A blue print of class.
- Syntax:

```
access interface name{
    return-type method-name1(paramater);
    return-type method-name2(paramater);
    type final-variable1 = value;
    type final-variable2 = value;
}
```

- Interface can be implemented using implements keyword.

```
class B implements NameInterface{

}
```

- The file name must be same name as the interface.
- All methods and variables are implicitly public.
- Variables are implicitly final and static – can't be changed during implementation.
- While implementing multiple interface, they are separated by comma (,).

```
class B implements NameInterface,CalculateInterface{

}
```

- Partial implementation can be made using abstract class.
- Interface can be extended only by other interface.
- Using interface we can implement multiple interface.

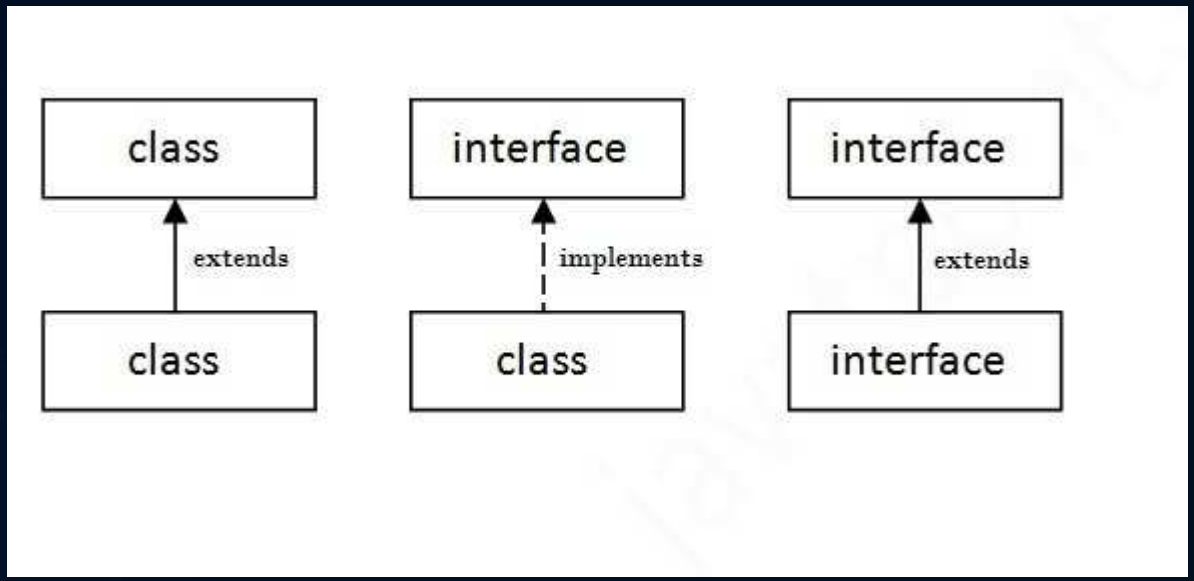


Fig: relation between class and interface

- Eg: Interface Declaration

```
1. /* File name : Animal.java */
2. interface Animal {
3.    public void eat();
4.    public void travel();
5. }
```

- Eg: Interface Implementation
- // @Source: https://www.tutorialspoint.com/java/java_interfaces.htm

```
1. public class Animals implements Animal {


2.    public void eat() {
3.      System.out.println("Mammal eats");
4.    }


5.    public void travel() {
6.      System.out.println("Mammal travels");
7.    }
```

```
1.    public int noOfLegs() {
2.      return 0;
3.    }

4.    public static void main(String args[]) {
5.      Animals m = new Animals();
6.      m.eat();
7.      m.travel();
8.    }
9.  }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java Animals
Mammal eats
Mammal travels

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>
```

Eg: Multiple Inheritance using interface.

```
interface A{
        void print();
}
interface B{
        int show(int Number);
}
interface C{
        void sayHello();
}
class MultipleInheritance implements A,B,C{
        public void print(){
                System.out.println("Hello");
        }
```

```
1.      public int show( int number){
2.              return number*2;
3.      }
4.      public void sayHello(){
5.              System.out.println("Hello C");
6.      }
7.
8.      public static void main(String args[]){
9.              MultipleInheritance obj = new MultipleInheritance();
10.             obj.print();
11.             int received = obj.show(45);
12.             System.out.println(received);
13.             obj.sayHello();
14.     }
15. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java MultipleInheritance
Hello
90
Hello C

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>
```

```
1.  //inheritance using interface
2.  interface A{
3.      void callA();
4.  }
5.  interface B extends A{
6.      void callB();
7.  }
8.  class InheritanceInterface implements B{
9.      public void callA(){
10.         System.out.println("Hello A");
```

```
1.      }
2.      public void callB(){
3.              System.out.println("Welcome B");
4.      }
5.      public static void main(String args[]){
6.              InheritanceInterface obj = new InheritanceInterface();
7.              obj.callB();
8.              obj.callA();
9.      }
10. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>javac InheritanceInterface.java

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>java InheritanceInterface
Welcome B
Hello A

C:\Users\USER\Desktop\lecture\java\Unit -IV\Programs>
```

Suggested Readings

- The respective topics in The complete Reference Java 7 (or any higher edition) by Hebert Schildt
- Related Topics in Oracle official java documentation
- Javatpoint.com

References

- The complete Reference Java 7 by Hebert Schildt
- https://www.javatpoint.com/ and tutorialspoint.com
- Java 8 in Action by Dreamtech press.
- Mit Opencourseware
- http://ee402.eeng.dcu.ie/
- https://docs.oracle.com/javase/tutorial/?sess=16e492aba137894101940f7f88d9f51f
- https://www.geeksforgeeks.org
- https://images.google.com for Images