IS-A RELATIONSHIP :

- The relationship between two objects which is similar to the parent and child relation is known as the Is-A relationship.
- In an Is-A relationship, the child object will acquire all properties of the parent object, and the child object will have its own extra properties.
- In an Is-A relationship, we can achieve generalization and specialization.

NOTE: 1

- **Parents are called generalized.**
- **Children are called specialized.**

NOTE: 2

private members, constructors and static members are not inherited to the child class.

ASUS Vivobook

EXAMPLE :

- With the help of the child class reference type, we can use the members of the parent's class as well as the child.
- With the help of parent class reference, we can use only the members of a parent but not the child class.

PARENT CLASS :

The parent class is also known as a superclass or base class.

CHILD CLASS :

The child class is also known as a subclass or derived class.

NOTE: Is-A relationship is achieved with the help of inheritance.

EXAMPLE :

- With the help of the child class reference type, we can use the members of the parent's class as well as the child.
- With the help of parent class reference, we can use only the members of a parent but not the child class.

PARENT CLASS :

The parent class is also known as a superclass or base class.

CHILD CLASS :

The child class is also known as a subclass or derived class.

NOTE: Is-A relationship is achieved with the help of inheritance.

INHERITANCE :

The process of one class acquiring all the properties and behavior from the other class is called inheritance.

In java, we can achieve inheritance with the help of

1. **extends keyword**
2. **implements keyword**

extends keyword :

extends keyword is used to achieve inheritance between two classes.

Types of inheritance

Single level inheritance

Multilevel inheritance

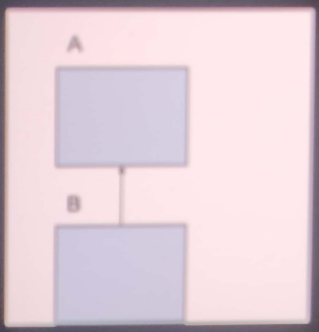
Hierarchical inheritance

Multiple inheritance

Hybrid inheritance

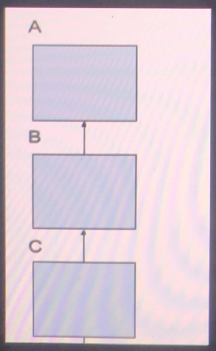
Single level inheritance

One parent class is having one child class is known as single level inheritance



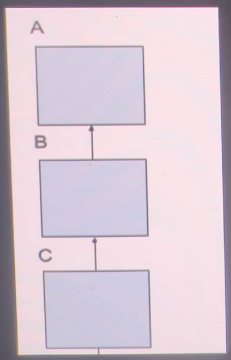
Multilevel inheritance

One superclass is having one subclass and subclass is having another one sub class or one parent class having more than one sub child classes through the child is known as multilevel inheritance

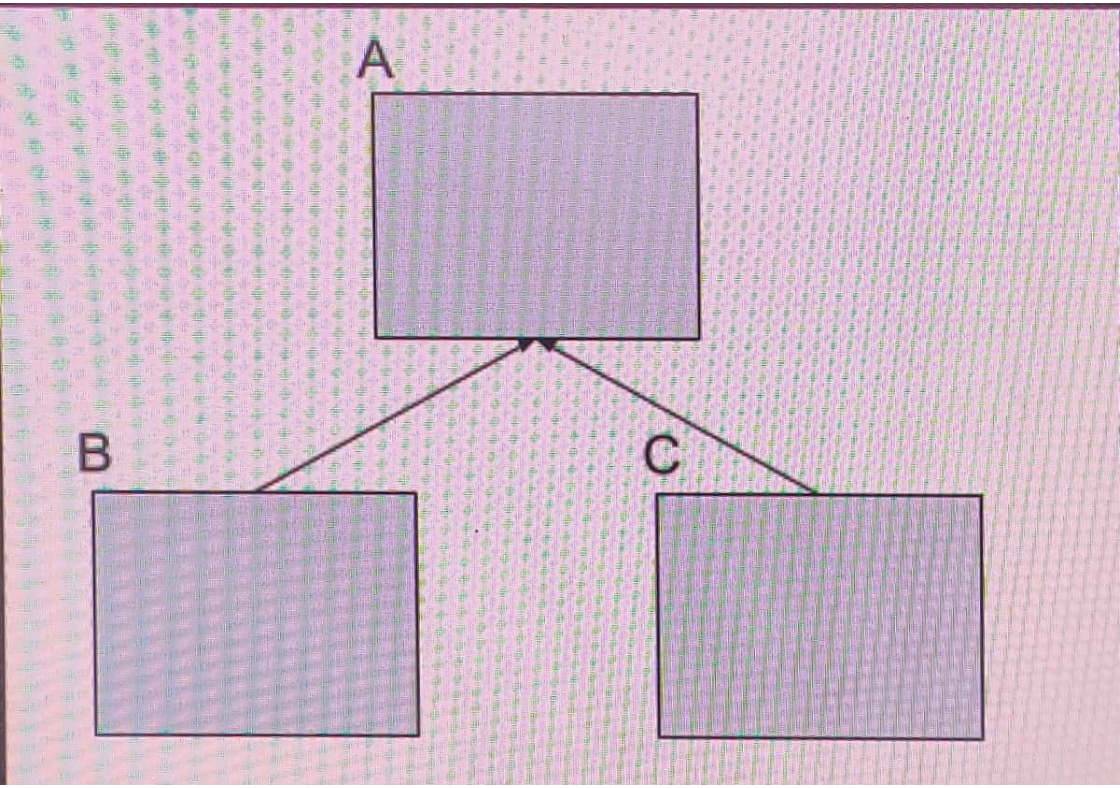


Multilevel inheritance

One superclass is having one subclass and subclass is having another one sub class or one parent class having more than one sub child classes through the child is known as multilevel inheritance

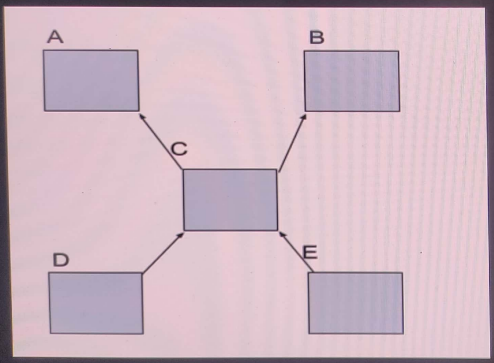


ASUS Vivobook



Hybrid inheritance

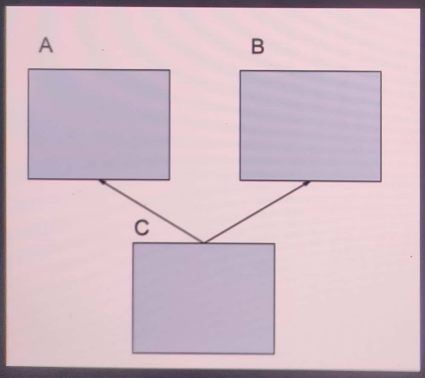
Combination of more than one inheritance is known as hybrid inheritance

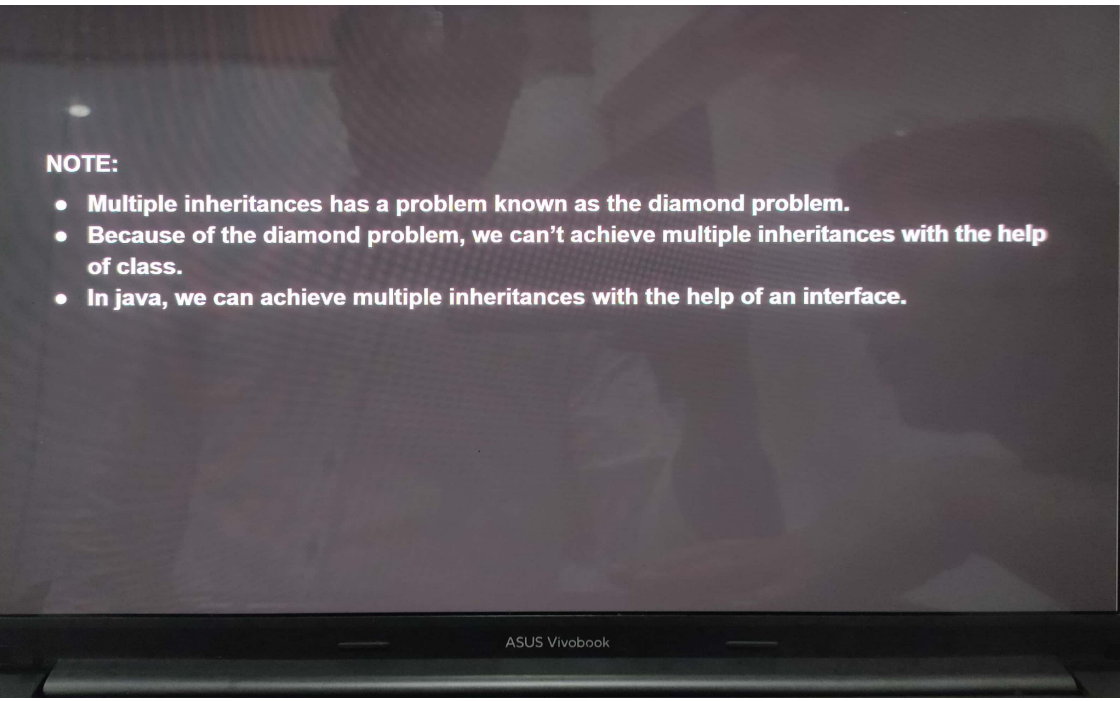


ASUS Vivobook

Multiple inheritance

One child is having more than one parent is known as multiple inheritance





DIAMOND PROBLEM because of Constructor:

Assume that there are two classes A and B. Both are not having user defined constructor . If class C inherits A and B then the time of creating the object of child class , we know that the child class constructor will call the parent class constructor which is called by no argument super() call statement to load the non static member of parent.

Now an ambiguity arises when we try to call the superclass no argument constructor with the help of no argument super() call statement.

This problem is known as the **diamond problem.**

DIAMOND PROBLEM because of method:

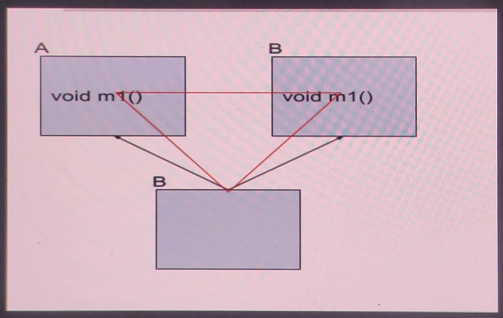
Assume that there are two classes A and B. Both are having the method with the same signature. If class C inherits A and B then these two methods are inherited to C.

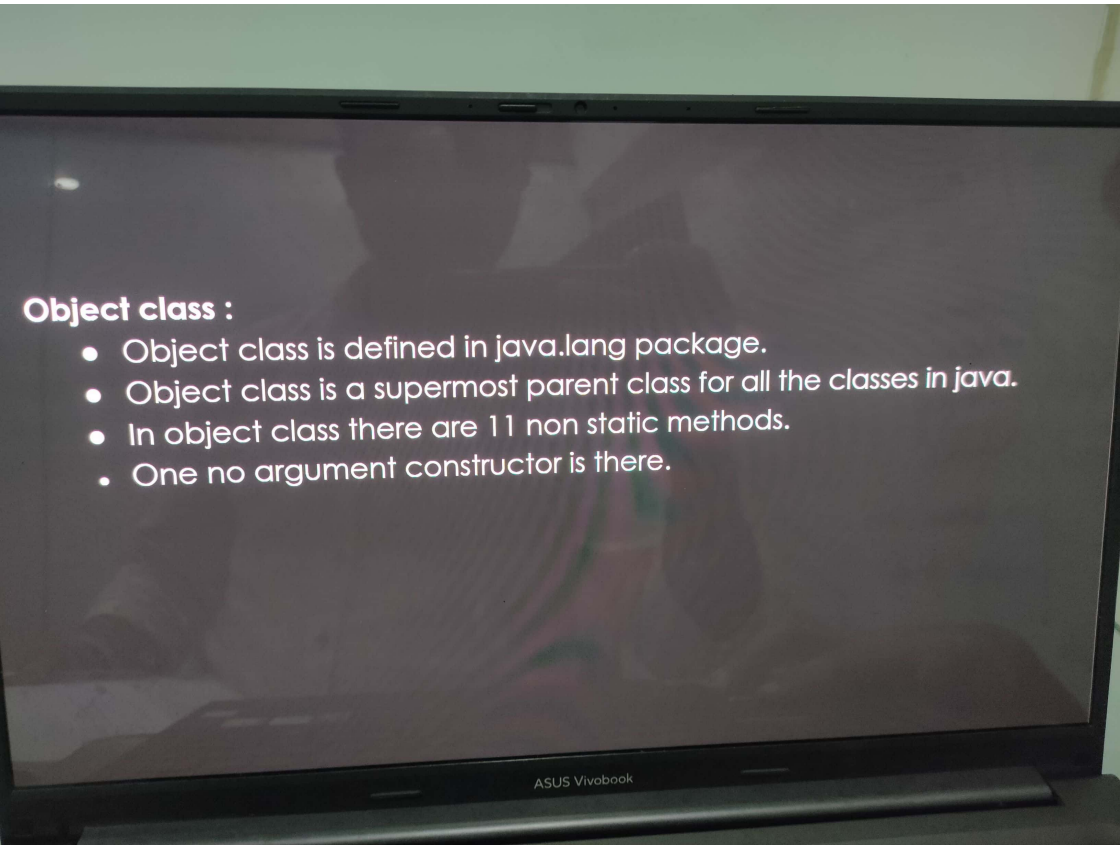
Now an ambiguity arises when we try to call the superclass method with the help of subclass reference.

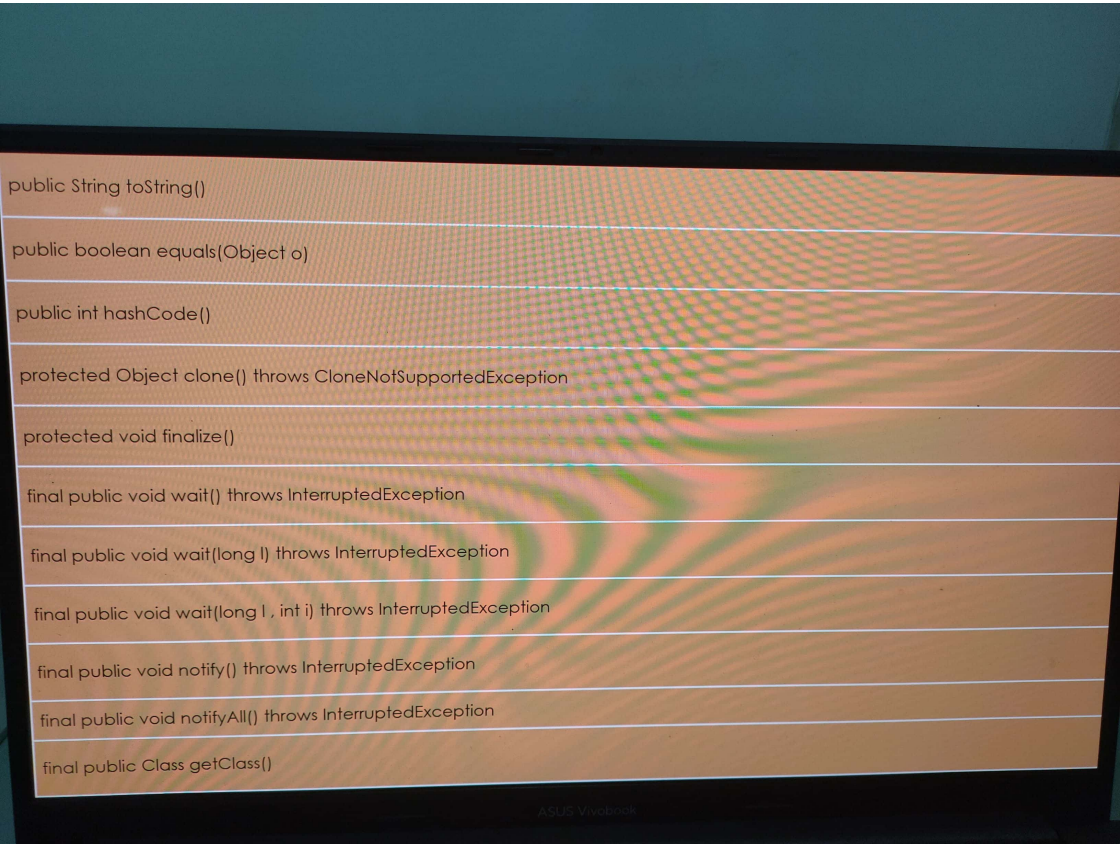
This problem is known as the **diamond problem.**

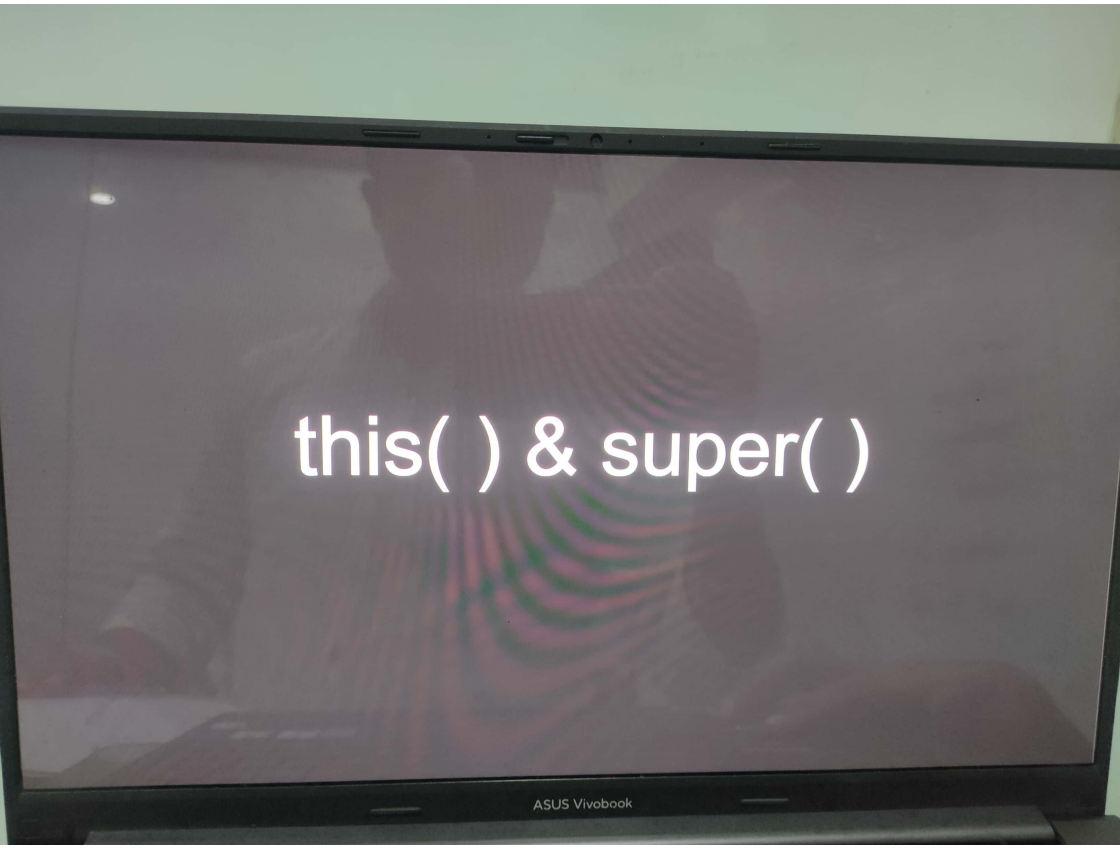
ASUS Vivobook

What is Diamond Ring problem?









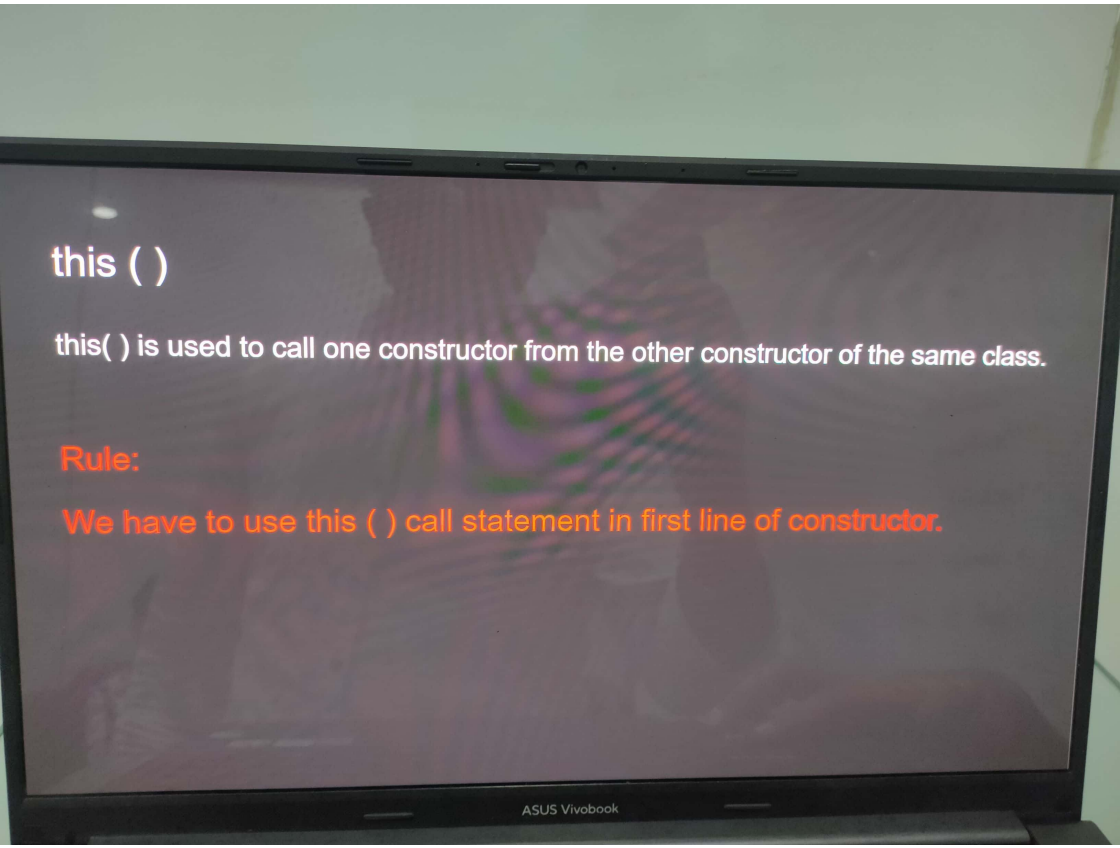
This keyword

If we want differentiate between local variable and non static variable

We have to use this keyword.

note:

It is used to hold the current executing object references.



super keyword

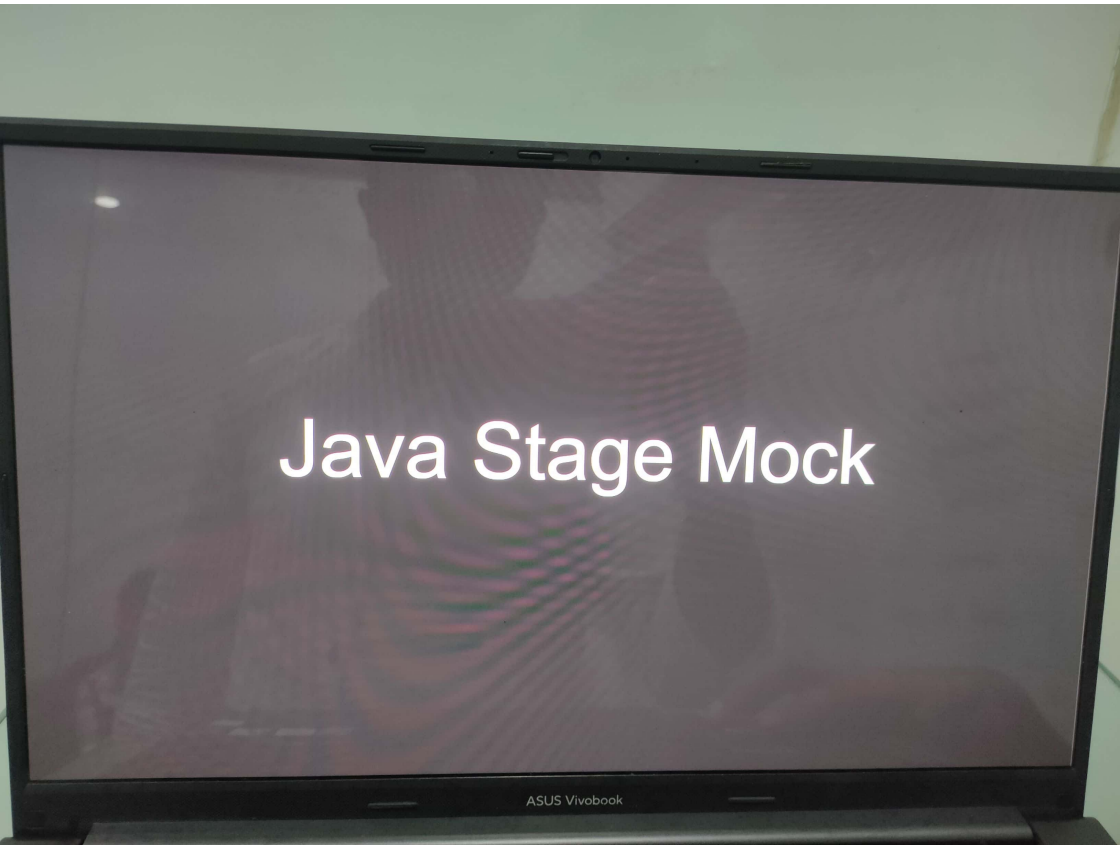
It is used to access the property of super class from the sub class .

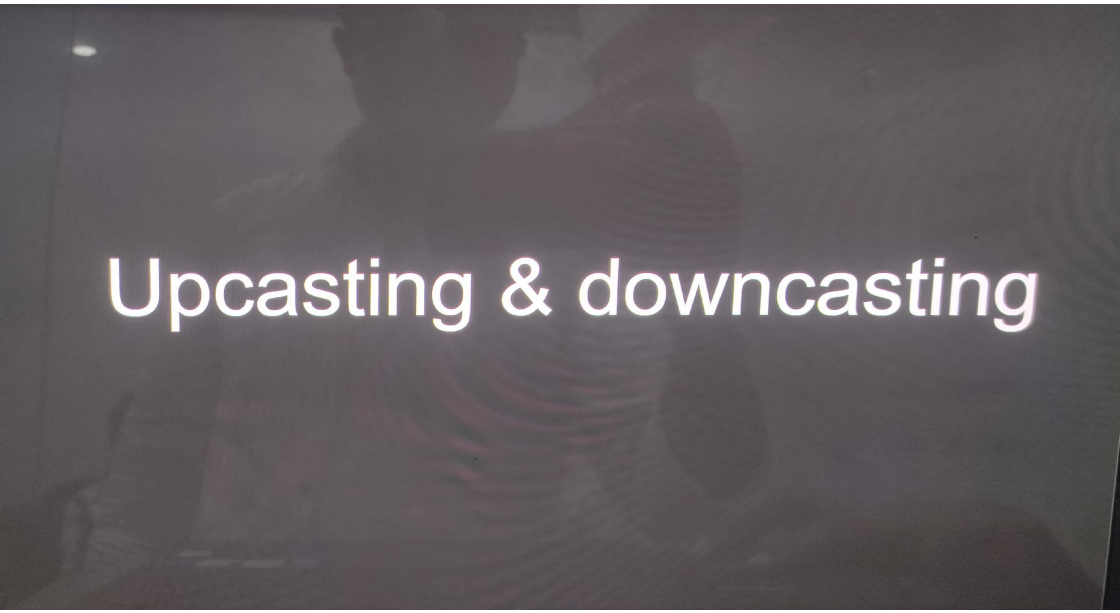
super ()

It is used to Call super class constructor from the sub class .

Rule:

We have to use super ( ) call statement in first line of constructor.





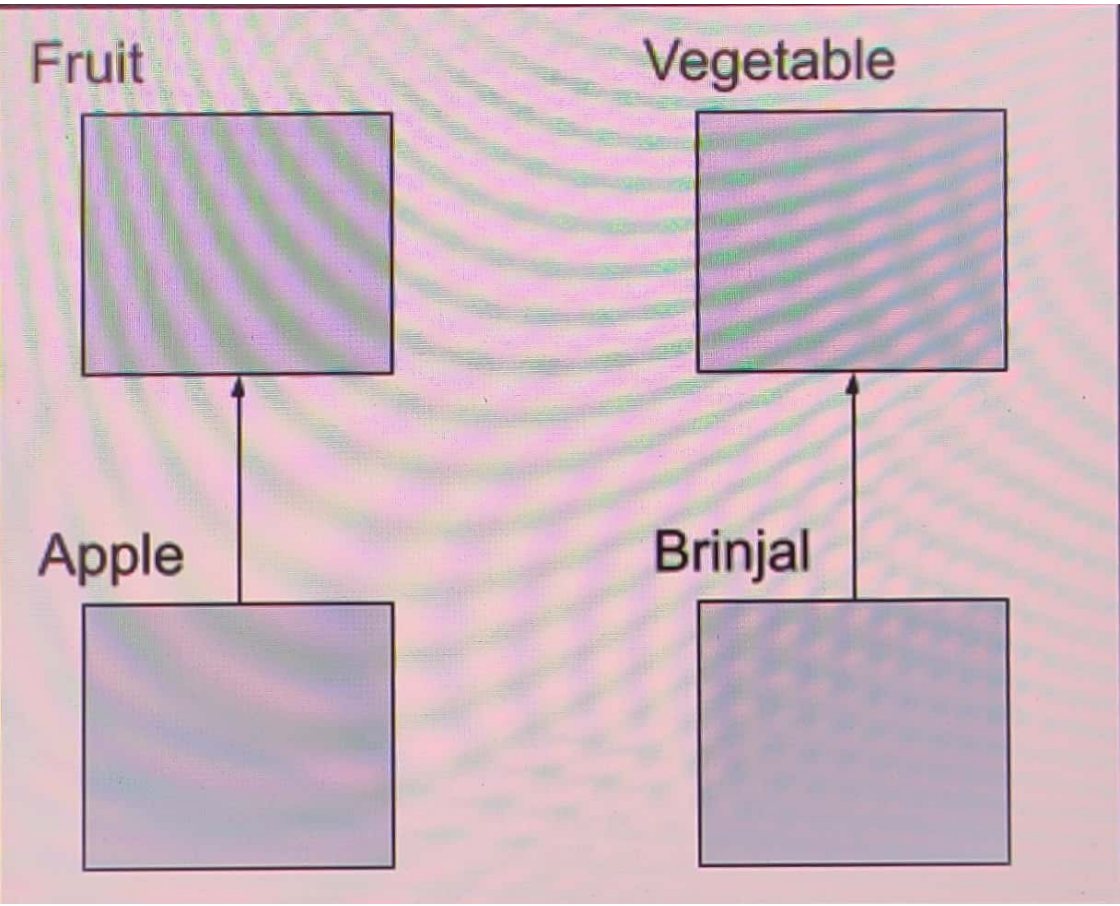
NON PRIMITIVE TYPE-CASTING (DERIVED TYPE CASTING)

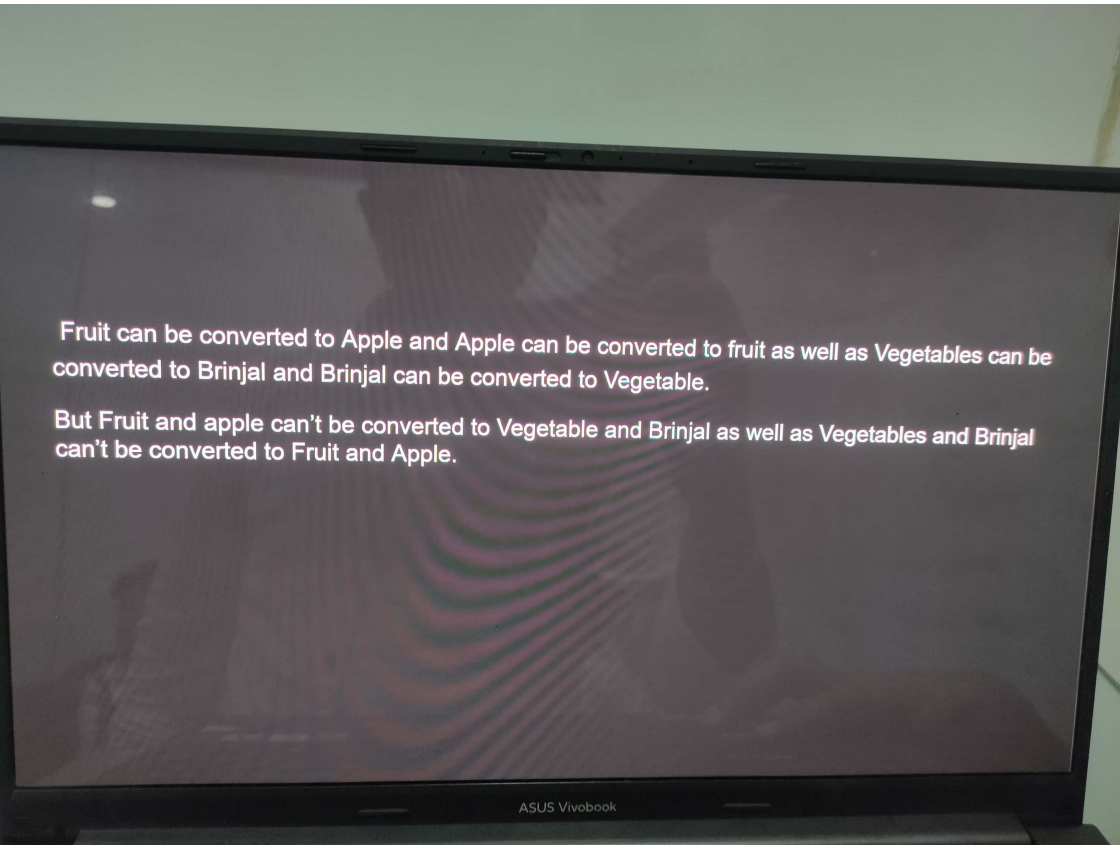
The process of converting one reference type into another reference type is known as non-primitive or derived typecasting.

RULES TO ACHIEVE NON PRIMITIVE TYPE CASTING:

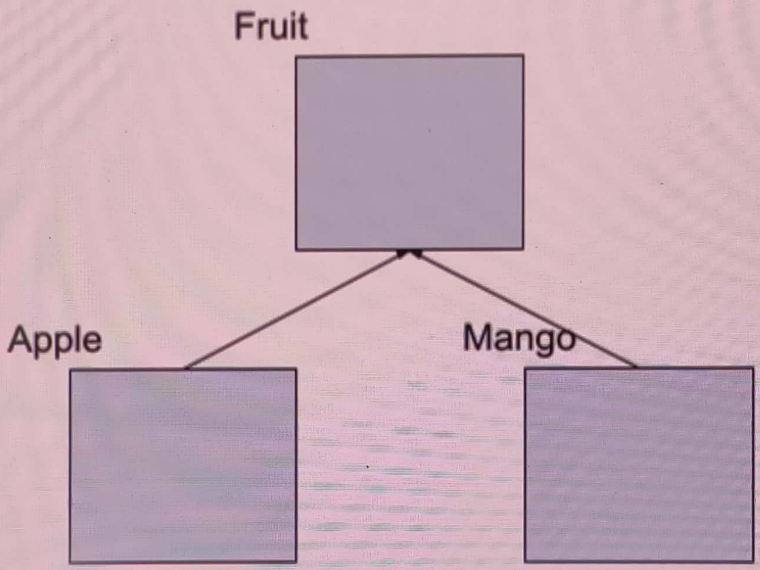
We can convert one reference type into another reference type only if it satisfies the following condition,

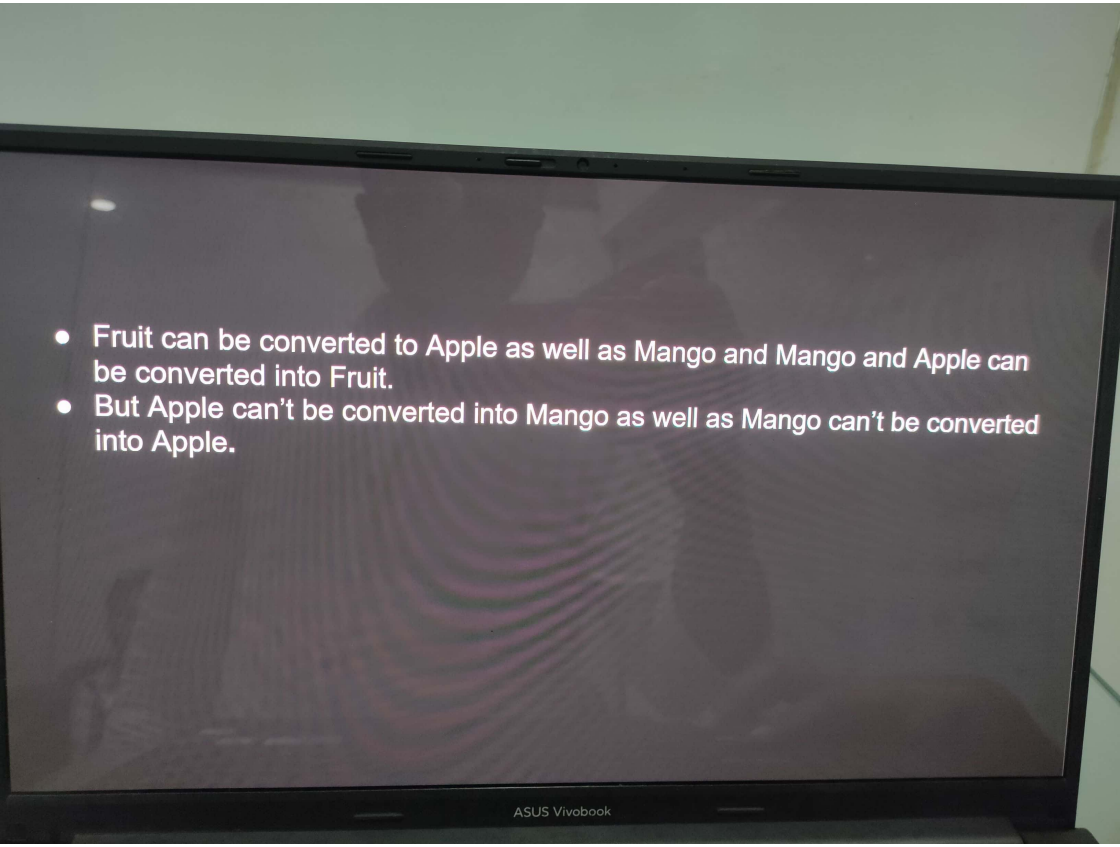
- There must be an Is-A relation (Parent and child) exist between two references.
- If the classes/Interfaces has a common child.





EXAMPLE 2:





TYPES OF NON-PRIMITIVE OR DERIVED TYPE CASTING:

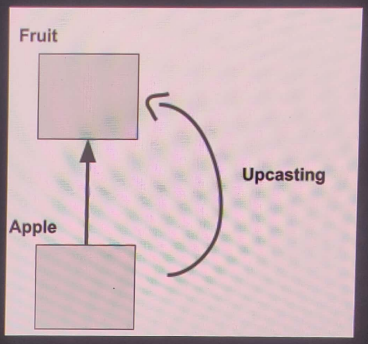
Non-primitive type casting can be further classified into two types,

I. **Upcasting**
II. **Downcasting**

UPCASTING:

The process of storing the child class object into a parent class reference type is known as upcasting.

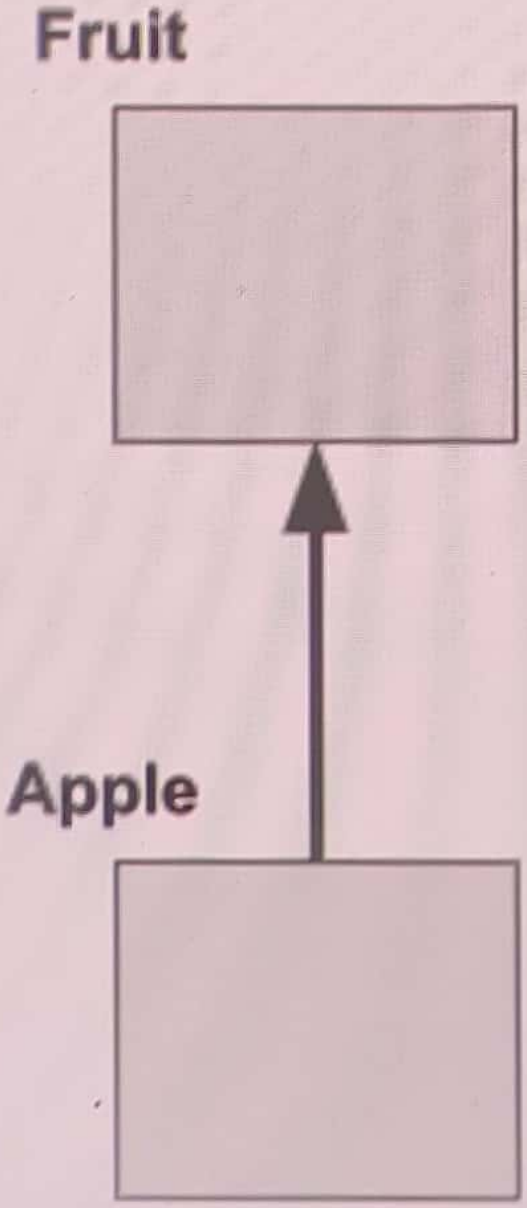
ASUS Vivobook



NOTE:

- The upcasting is implicitly done by the compiler.
- It is also known as auto upcasting.
- Upcasting can also be done explicitly with the help of a typecast operator.
- Once the reference is upcasted we can't access the members of the child.

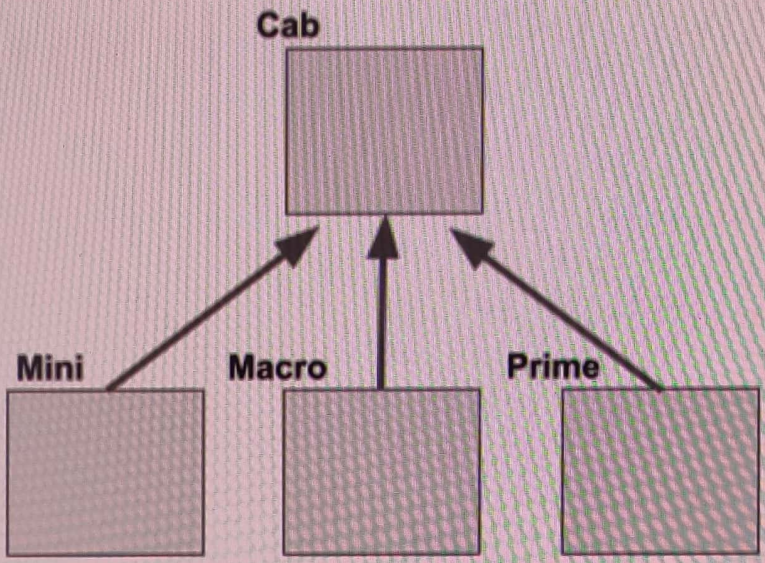
EXAMPLE :



WHY DO WE NEED UPCASTING?

- It is used to achieve generalization.
- It helps to create a generalized container so that the reference of any type of child object can be stored.

EXAMPLE :




```
Cab c ;
c = new Mini();
c = new Macro();
c = new Prime();
```

DISADVANTAGE :

There is only one disadvantage of upcasting that is, once the reference is upcasted its child members can't be used.

NOTE:

In order to overcome this problem, we should go for downcasting.

DOWNCASTING:

The process of converting a parent (superclass) reference type to a child (subclass) reference type is known as downcasting.

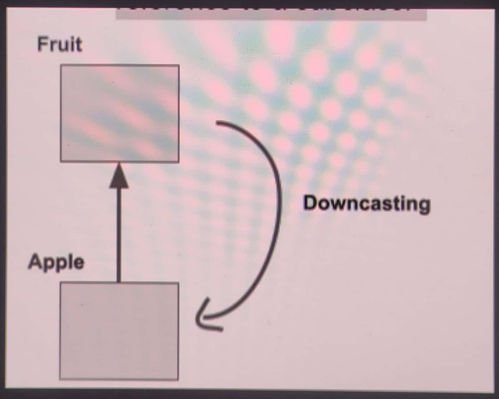
NOTE:

- **Downcasting is not implicitly done by the compiler.**
- **It should be done explicitly by the programmer with the help of a typecast operator.**

ASUS Vivobook

WHY DO WE NEED A DOWNCASTING?

- If the reference is upcasted, we can't use the members of a subclass.
- To use the members of a subclass we need to downcast the reference to a subclass.



ClassCastException :

- It is a RuntimeException.
- It is a problem that occurs during runtime while downcasting.

When and why do we get a ClassCastException?

When we try to convert a reference to a specific type(class), and the object doesn't have an instance of that type then we get ClassCastException.

EXAMPLE:

Case 1:

```
Child c = (Child)new Parent();//ClassCastException
```

Case 2:

```
Parent p=new Parent( );
Child c=(Child)p;//ClassCastException
```

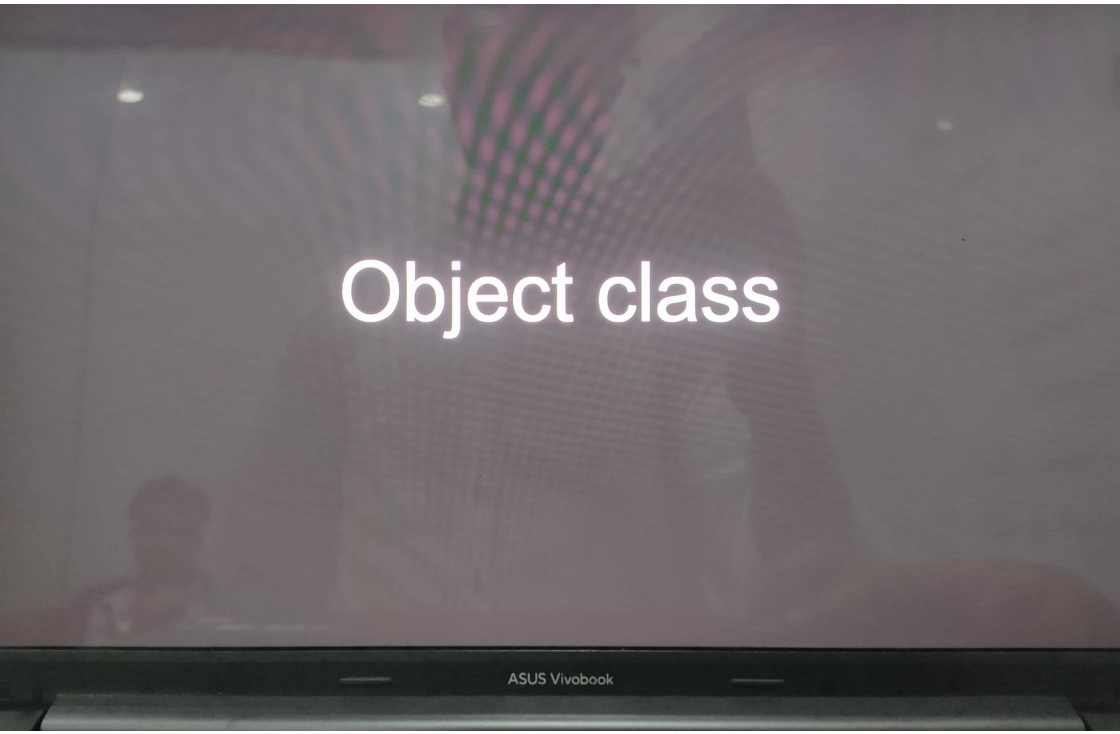
INSTANCEOF OPERATOR:

instanceof operator:

- It is a binary operator
- It is used to test if an object is of the given type.
- The return type of this operator is boolean.
- If the specified object is of a given type then this operator will return true else it returns false.

Syntax to use instanceOf operator :

(Object_Ref) instanceof (type)



POLYMORPHISM

Polymorphism is derived from two different Greek words 'Poly' means Numerous , 'Morphs' means form Which means Numerous form. Polymorphism is the ability of an object to exhibit more than one form with the same name.

For Understanding :

One name -----> Multiple forms

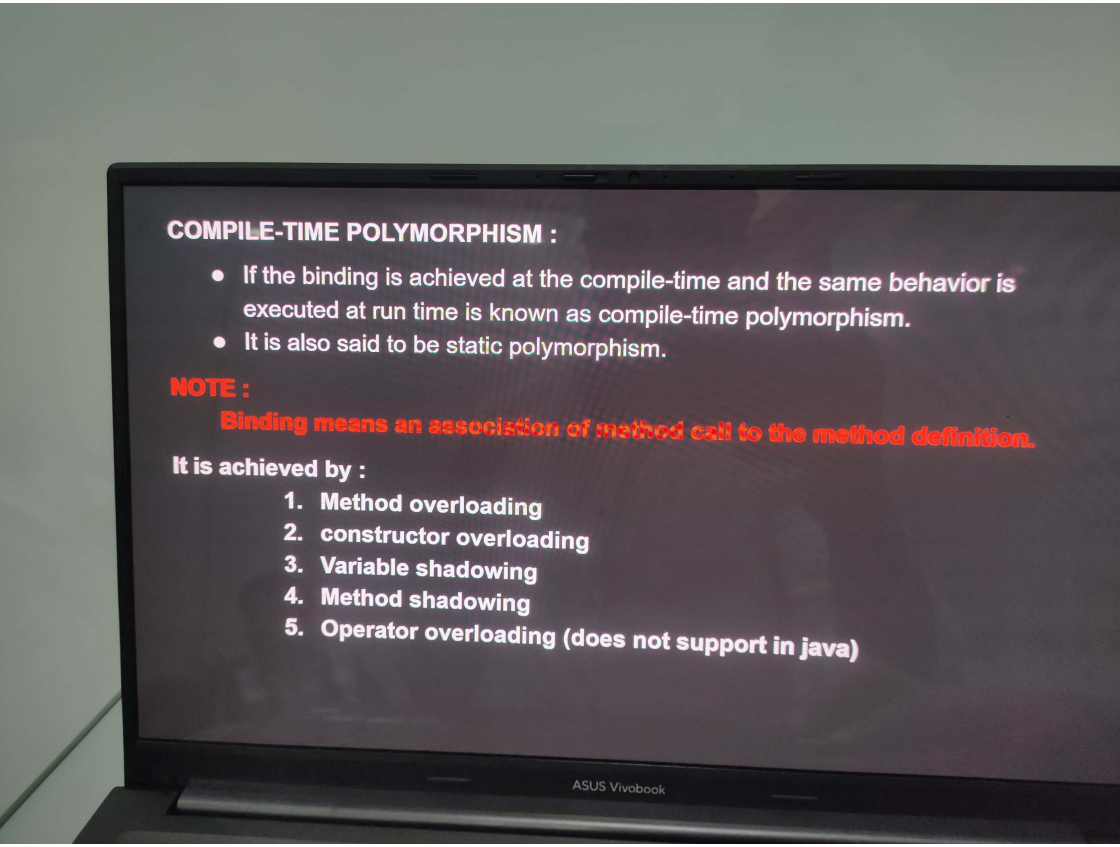
One variable name ------> Different values

One method name -----> Different behavior

TYPES OF POLYMORPHISM :

In java, we have two types of polymorphism,

1. **Compile-time polymorphism**
2. **Runtime Polymorphism**



METHOD OVERLOADING :

If more than one method is created with the sa
same class are known as method overloading.

Different formal arguments means :

Differ in no. of FA

Differ in type of FA

Differ in order of FA

EXAMPLE : java.io.PrintStream;

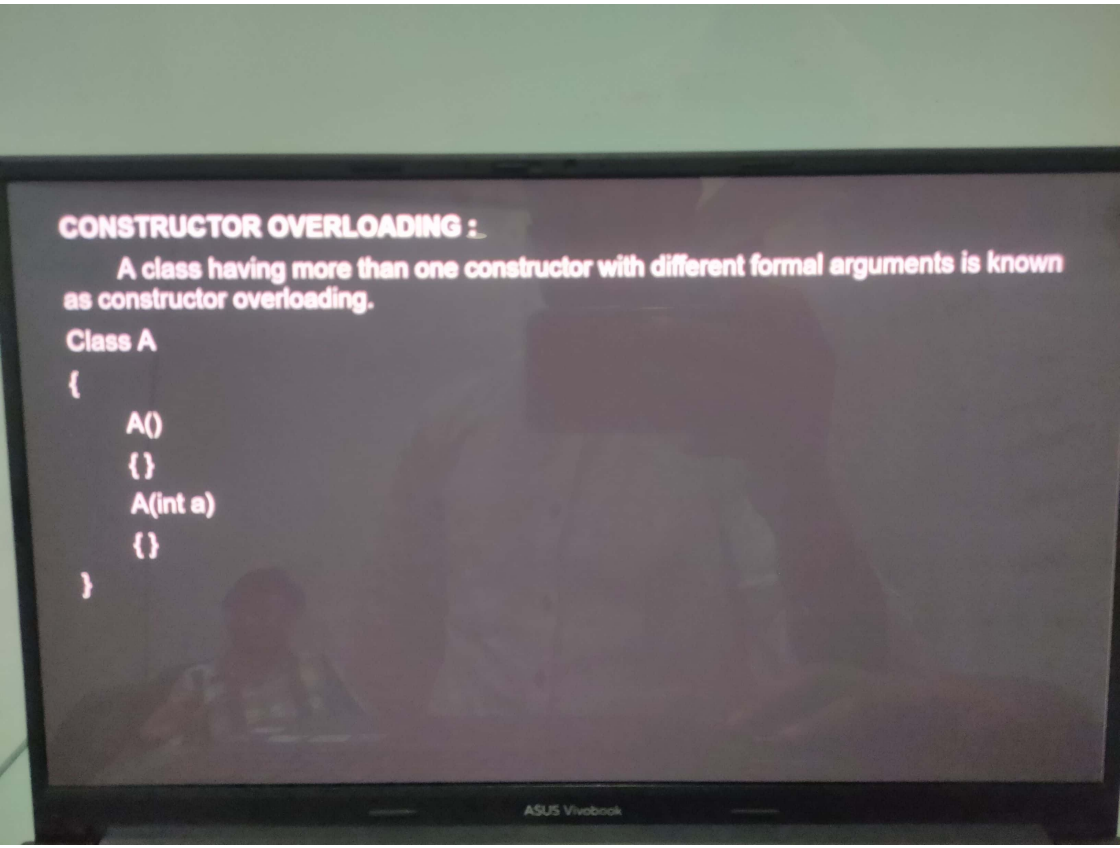
println()

println(int a)

println(double d)

println(String s)

These are some of the overloaded methods
formal arguments) implemented in java.io.PrintSt



METHOD SHADOWING :

If a subclass and superclass have the static method with the same declaration, but different in implementation is known as method shadowing.

Which method implementation gets executed, depending on what?

In method shadowing binding is done at compile-time, hence it is compile-time polymorphism. The execution of the method depends on the reference type and does not depend on the type of object created.

NOTE :

- **The return type should be the same or it should be a covariant return type.**
- **Access modifier should be same or higher visibility than superclass method.**
- **Method shadowing is applicable only for the static method.**
- **It is compile time polymorphism**
- **Execution of implemented method depends on the reference type of an object.**

ASUS Vivobook

EXAMPLE : Method Shadowing

```
class Parent
{
  public static void test()
  {
        System.out.println("From parent");
  }
}
class Child extends Parent
{

  public static void test()
  {
        System.out.println("From child");
  }
  public static void main(String[] args)
  {
        Parent p = new Child();
        p.test(); // from parent  // this is based on the ref variable type

        Child c = new Child();
        c.test(); // from child     // this is based on the ref variable type

        Parent p1 = new Parent();
        p1.test(); // from parent   // this is based on the ref variable type
  }
}
```

VARIABLE SHADOWING :

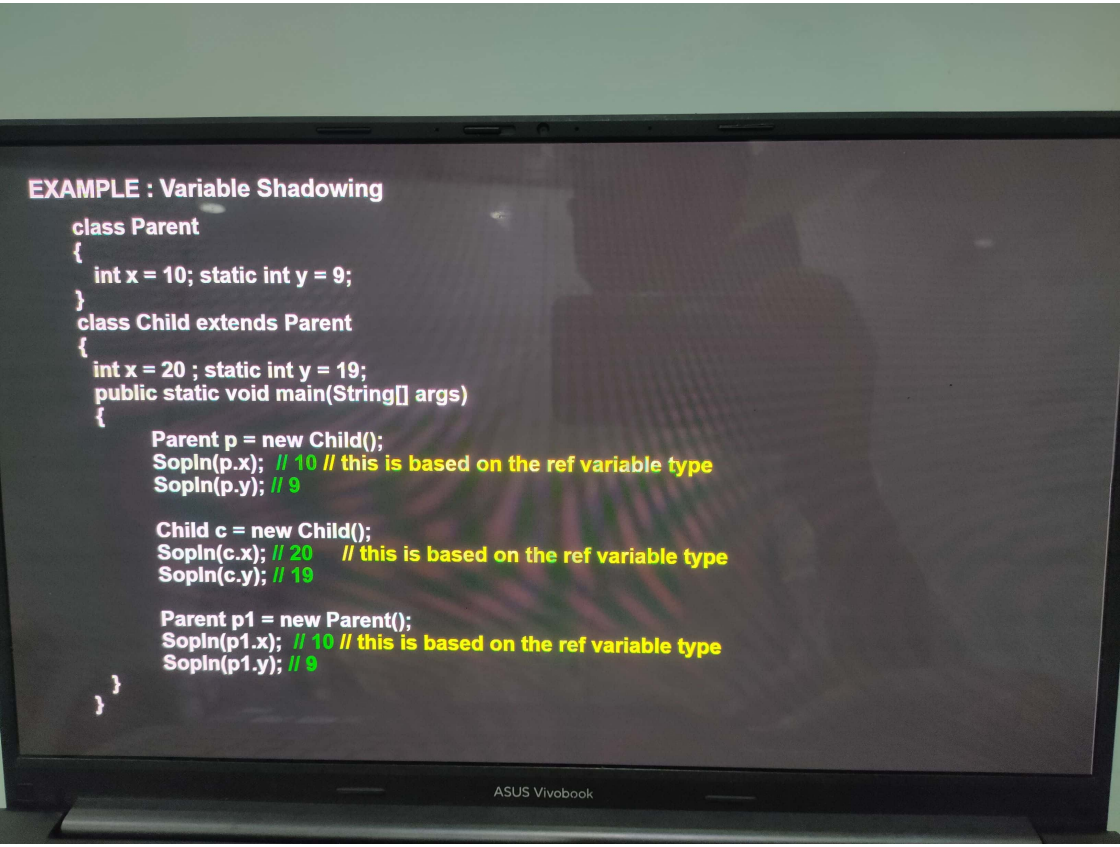
If the superclass and subclass have variables with the same name but change in values then it is known as variable shadowing.

Which variable is used, depending on what?

In variable shadowing binding is done at compile-time, hence it is a compile-time polymorphism. The Variable used depends on the reference type and does not depend on the type of object created.

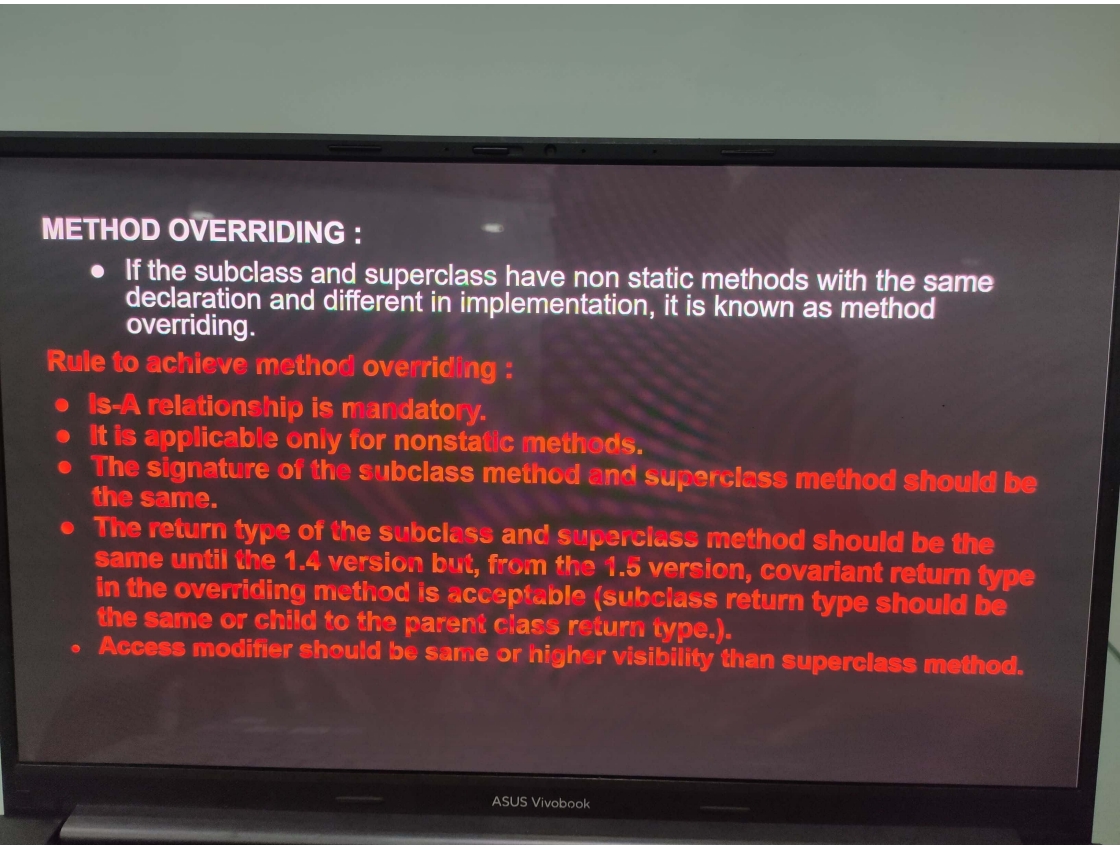
NOTE :

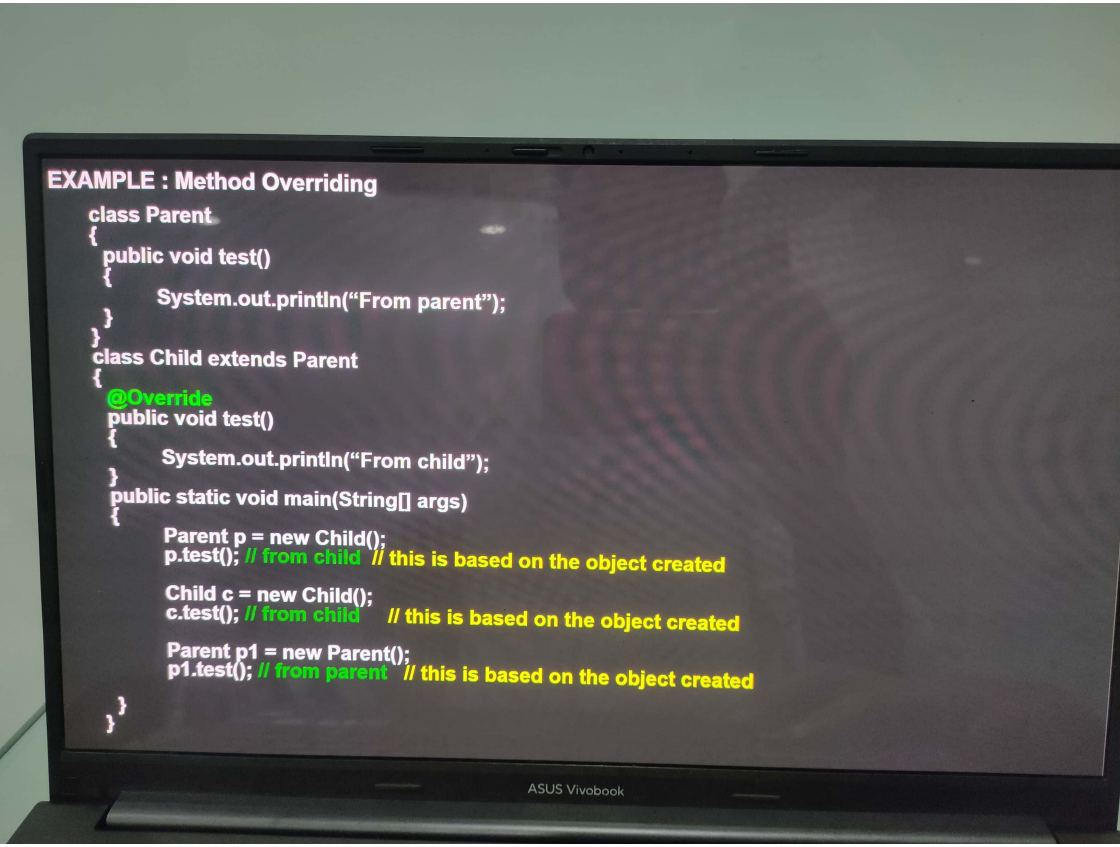
- **It is applicable for both static and non-static variables.**
- **It is a compile-time polymorphism.**
- **Variable usage depends on the type of reference and does not depend on the type of object created.**

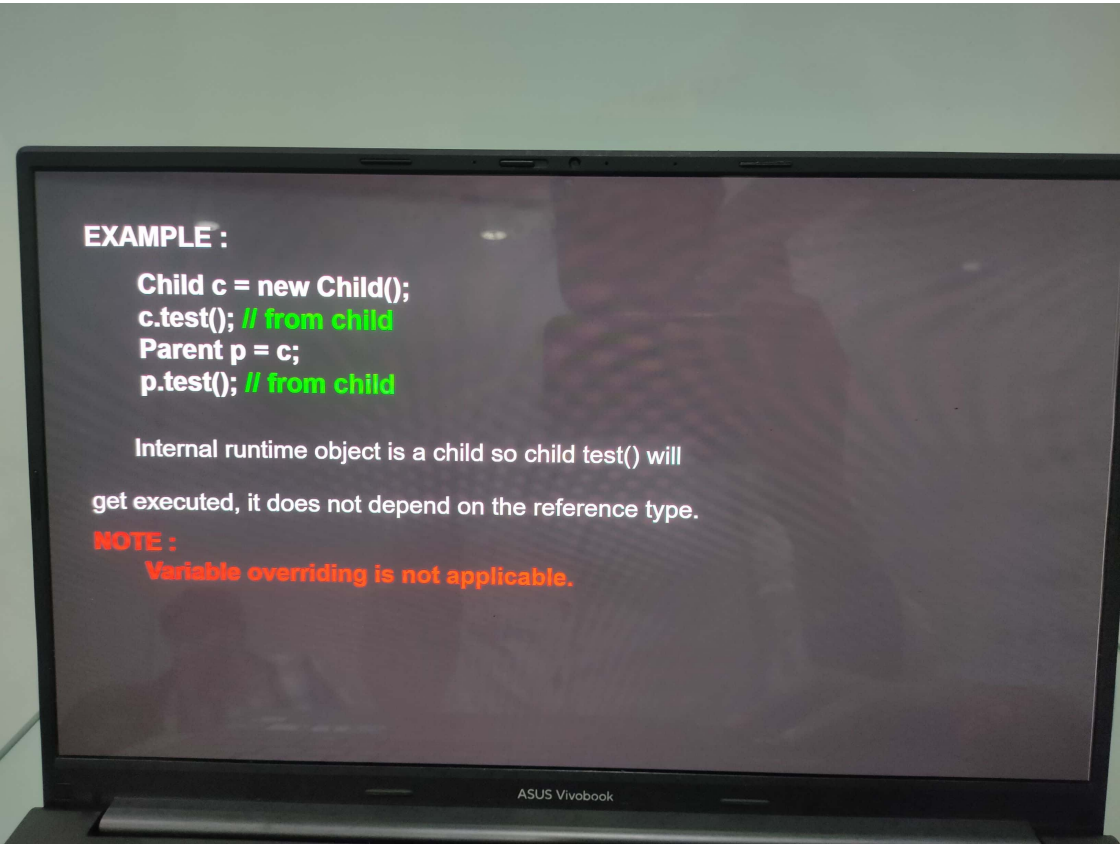


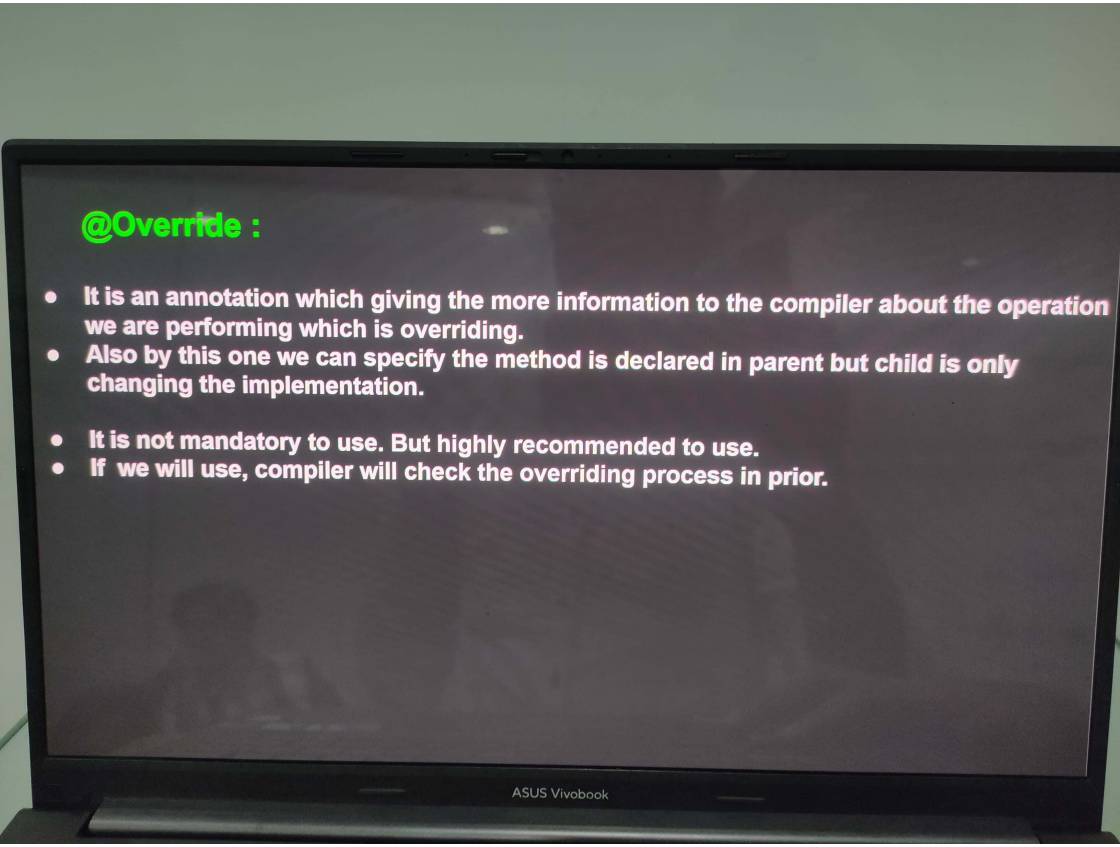
RUNTIME POLYMORPHISM :

- If the binding occure at compile time but different behaviour is achieved at the runtime then it is known as runtime polymorphism.
- It is also known as dynamic binding.
- It is achieved by method overriding.









Abstraction?

It is a design process of hiding the implementation and showing only the functionality (only declaration) to the user is known as abstraction.

HOW TO ACHIEVE ABSTRACTION IN JAVA?

- In java, we can achieve abstraction with the help of abstract classes and interfaces.
- We can provide implementation to the abstract component with the help of inheritance and method overriding.

ABSTRACT MODIFIER :

- The abstract is a modifier, it is a keyword.
- It is applicable for methods and classes.

ABSTRACT METHOD :

- A method that is prefixed with an abstract modifier as well as does not have the implementation and end with semicolon is known as the abstract method.
- This is also said to be an incomplete method.
- It is generally non static in nature.
- It can't be private or static or final .

Syntax to create abstract method :

```
abstract [access modifier] returnType methodName([For_Arg]) ;
```

NOTE :

Only child class of that class is responsible for giving implementation to the abstract method.

Abstract Class

Abstract Class : If the class is prefixed with an abstract keyword / modifier then it is known as abstract class. We can't create object (instance) for the abstract class.

Characteristics :

1. We can't create an instance of an abstract class.
2. We can have abstract class without an abstract method.
3. An abstract class can have both abstract and concrete method.
4. If a class has at least one abstract method which is either declared or inherited but not overridden , then it is mandatory to make the class as abstract class.

Abstract Class

— — —

EXAMPLE :

```
abstract class Atm
{
        abstract public double withdrawal();
        abstract public void getBalance();
        abstract public void deposit();
}
```

// hiding implementation by providing only functionality

Atm a = new Atm() // CTE

NOTE :

Only subclass of Atm is responsible for giving implementation to the methods declared in an Atm class.

When you have to go for Abstract Method

1. When we don't have the clear idea about the implementation of the method
2. If want to leave the implementation of the method for the child who should override the method and give the implementation.

Implementation of abstract method :

- If a class extend abstract class then it should give implementation to all the abstract method of the superclass.
- If inheriting class doesn't like to give implementation to the abstract method of superclass then it is mandatory to make subclass as an abstract class.
- If a subclass is also becoming an abstract class then the next level child class is responsible to give implementation to the abstract methods.

STEPS TO IMPLEMENT ABSTRACT METHOD :

STEPS TO IMPLEMENT ABSTRACT METHOD :

STEP 1:
Create a class.

STEP 2:
Inherit the abstract class/ component.

STEP 3:
Override the abstract method inherited (Provide implementation to the inherited abstract method).

Example of giving the implementation to abstarct method

EXAMPLE :

```
abstract class WhatsApp
{
    abstract public void send();
}
class Application extends WhatsApp
{
    public void send()
    {
        System.out.println("Send() method is implemented");
    }
}
```

Creating object and calling the abstract method :

```
WhatsApp w = new WhatsApp(); // not possible
Application a = new Application();
a.send();
```

Note : We can't create the object of abstract class but we can create the reference variable of abstract class and store the child reference which is concrete

```
WhatsApp w = new Application();
w.send();  //O/P : Send() method is implemented
```

Here send() is not the child class method , it is declared in the parent and only implementation is given by child.So it is possible to access the member by the parent reference variable.

Concrete Class

CONCRETE CLASS :

The class which is not prefixed with an abstract modifier and doesn't have any abstract method, either declared or inherited is known as concrete class.

NOTE :

In java, we can create objects only for the concrete class.