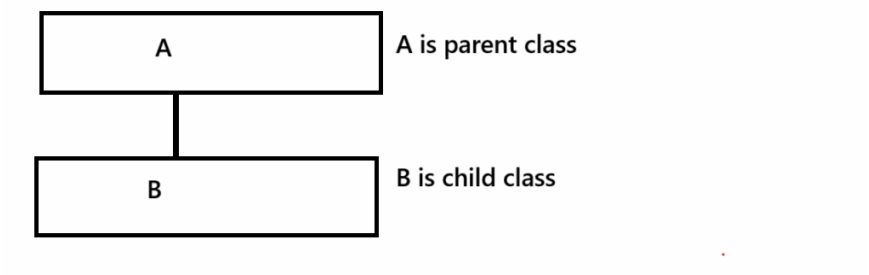
Inheritance

Q. What is inheritance?

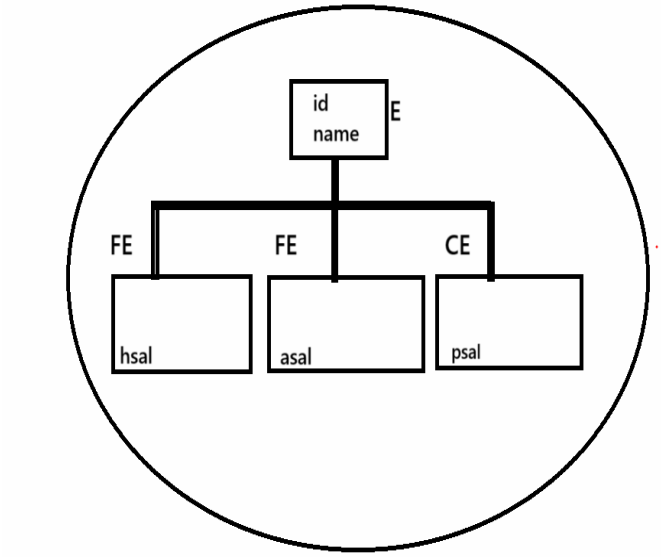
Inheritance means transfer the properties of one class to another class called as inheritance. Property provider class called as parent class and Property receiver class called as child class.



Q. Why use inheritance or what are benefits of inheritance?

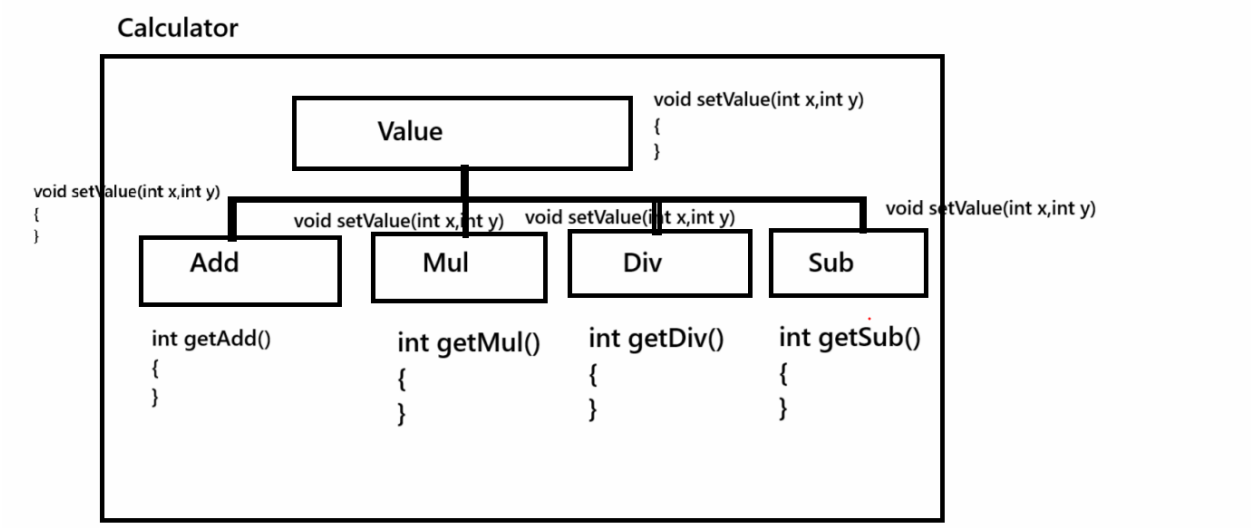
---

1. Reusability of code: If we think about inheritance we can reuse parent content via child object without creating object of parent. Means some time if we have some logic which we required in multiple places or some common data which we required in multiple classes so creating object and use it is not good approach better way we can use it using inheritance.



If we think about above diagram we have three category of employee which contain some common data i.e id and name and we inherit this class in FE, FulltimeEmployee and CE so id and name available in all classes and every class has own property also.

2. Extensibility of code: Child class can acquire properties from parent class and add own properties as per the requirement as well as can modify the parent property if required with the help of overriding called as extensibility.



If we think about above diagram we have parent class name as Value which contain method setValue() and we have four child class and every child has own method like as getAdd() in Add class and getMul() in Mul class means every child has parent property as well as own property called as extensibility

How to implement inheritance practically using JAVA?

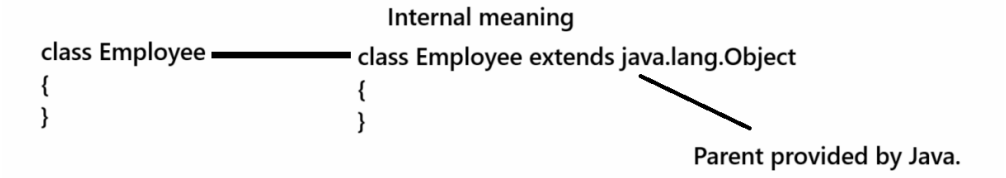
If we want to implement inheritance practically using java we have to use extends keyword

Syntax: class childclassname extends parentclassname{
}

Example:

```
class Value //Value is parent
{
}
class Add extends Value //Add is child class
{
}
```

Note: we cannot create any program in java without inheritance.
Because if we think about java every class has default parent class name as Object class means when user not give the parent class to class then java by default provide parent to class name as Object class and it is member of java.lang package



Q. Why JAVA provide Object class as parent to all classes?

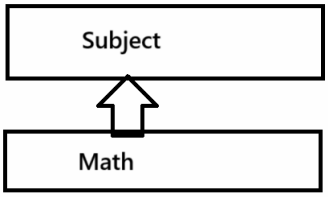
Because Object class contain some inbuilt methods which required in every class for daily operation purpose so object is parent class of every class in java and methods given below.

int hashCode(), boolean equals(), String toString(), Object clone(), void notify(), void notifyAll(),void wait(),void wait(int), void finalize() , Class getClass(), static{ }
Note: Object class method we will discuss later in remaining part of JAVA.

Note: When we perform inheritance between two classes then classes bind in new relationship known as IS-A Relation.

Q. What is IS-A relation?

When we perform inheritance in classes called as IS-A relationship.



MATH is subject because all properties of subject present in Math class.

Q. What is HAS-A Relation?

HAS-A Relationship means when we create object of one class as member of another class called as HAS-A relationship

```
class Chapter
{
}
class Ratio extends Chapter
{
}
class Subject
{
}
class Math extends Subject
{
   Ratio r = new Ratio();

}
```

If we think about this code we can say Math is Subject and Math has-a ratio and Ratio is chapter.

Q. What is meaning of System.out.println ()?

System is class out is reference of PrintStream class and System and PrintStream maintain HAS-A relationship between them and println() is overloaded method for display output on output screen.

```
class Ratio
{ void showFormula(){
    System.out.println("I am ratio formula");
  }
}
class Math
{
   static Ratio r = new Ratio();
}
class Exam
{
   public static void main(String x[])
    {
        Math.r.showFormula();

    }
}
```

Math is class and r is refence of Ratio class and Math and Ratio maintain HAS-A relationship between them and showFormula() is method of Ratio for display the formula

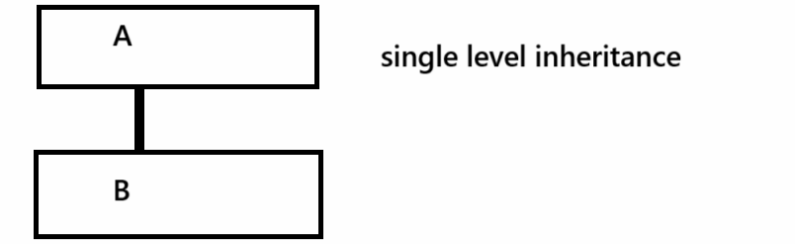
```
class PrintStream
{  void println(int x)
   {
   }
   void println(String x)
   {
   }
   void println(float x){
   }
}
class System
{  static PrintStream out = new PrintStream();
}
public class Test
{
   System.out.println("good");
}
```

System is class and out is static reference of PrintStream class. System and PrintStream maintain HAS-A relationship between println() is overloaded method for display output on output screen.

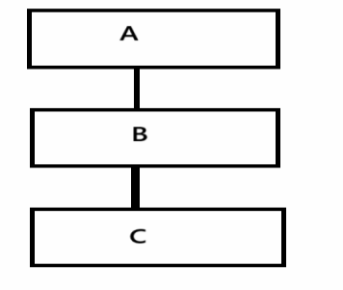
Types of inheritance

---

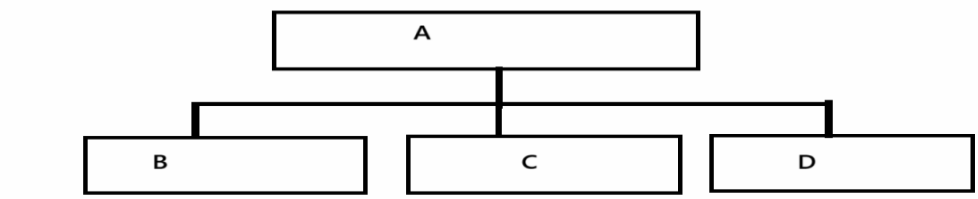
1. Single level inheritance: there is only one parent class and only one child class called as single level inheritance.



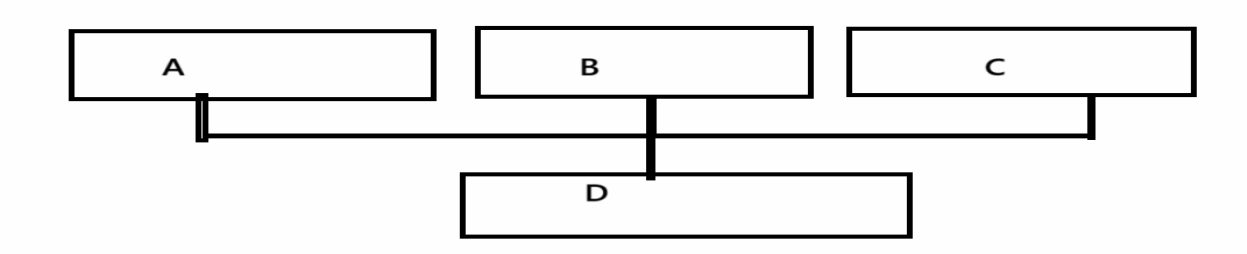
2. Multi level inheritance: one class is parent of another and child of another class called as multi level inheritance.



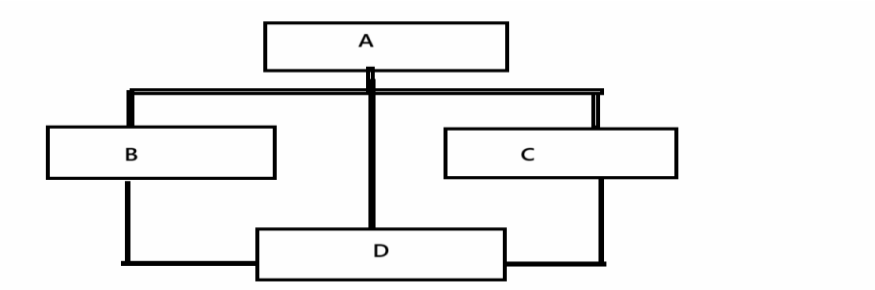
3. Hierarchical Inheritance: Hierarchical inheritance means single parent class can have more than one child classes called as hierarchical inheritance.



4. Multiple inheritances: Multiple inheritances means more than one parent classes and single child class called as multiple inheritances.



5. Hybrid inheritance: Hybrid inheritance is combination of all types of inheritance called as hybrid inheritance.



How to work practically with inheritance

```
class Num
{
   int no;
   void setValue(int no)
   {  this.no=no;
   }
}
class Square extends Num
{   int getSquare()
    { return no*no;
    }
}
public class SingleInheSepApp
{
    public static void main(String x[])
    {
        Square s1 = new Square();
        s1.setValue(10);
        int result=s1.getSquare();
        System.out.printf("Square is %d\n",result);
    }
}
```

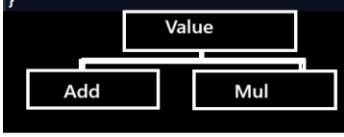
```
Command Prompt

C:\Program Files\Java\jdk-23\bin>javac SingleInheSepApp.java

C:\Program Files\Java\jdk-23\bin>java SingleInheSepApp
Square is 100

C:\Program Files\Java\jdk-23\bin>
```

Note: if we think about above left hand code we create object of child class and using child object we access parent members.

Now we want to implement the Hierarchical inheritance.

```
class Value
{
   protected int x,y;
    void setValue(int x,int y)
    {  this.x=x;
       this.y=y;
    }
}
class Add extends Value
{   int getAdd(){
      return x+y;
    }
}
class Mul extends Value
{   int getMul()
    { return x*y;
    }
}
```

```
public class HierApp
{
    public static void main(String x[])
    {
        Add ad = new Add();
        ad.setValue(10,20);
        int result=ad.getAdd();
        System.out.printf("Addition is %d\n",result);

        Mul  m = new Mul();
        m.setValue(5,4);
        result=m.getMul();
        System.out.printf("\nMultiplication is %d\n",result);
    }
}
```



Constructor in inheritance

When we have default constructor in parent class and constructor in child class and if we create object of child class then parent constructor get executed before child class constructor.

```
class P
{
    P()
    { System.out.println("I am parent constructor");
    }
}
class C extends P
{
    C()
    {
        System.out.println("I am child constructor");
    }
}
public class PCAPP
{
    public static void main(String x[])
    {
            C c1 = new C();
    }
}
```

```
C:\Program Files\Java\jdk-23\bin>javac PCAPP.java

C:\Program Files\Java\jdk-23\bin>java PCAPP
I am parent constructor
I am child constructor
```

Note: when parent class constructor has parameter and if we create object of child class then parent constructor not executed before child class constructor and in this situation child constructor as has responsibility to pass parameter to parent constructor and for that we can use super constructor concept.

```
class P
{       10
    P(int x)
    { System.out.println("I am parent constructor "+x);
    }
}
class C extends P
{
    C()
    {
        super(10);
        System.out.println("I am child constructor");
    }
}
public class PCAPP
{
    public static void main(String x[])
    {
            C c1 = new C();
    }
}
```

```
C:\Program Files\Java\jdk-23\bin>javac PCAPP.java

C:\Program Files\Java\jdk-23\bin>java PCAPP
I am parent constructor 10
I am child constructor

C:\Program Files\Java\jdk-23\bin>
```

Q1. Can we perform constructor chaining using inheritance?

Yes we can perform constructor chaining using inheritance and for that we can use super() constructor

Shown in following code

```
class P
{
  P(int x)
  { System.out.println("I am parent constructor "+x);
  }
}
class C extends P
{  C()
  { super(10);
    System.out.println("I am child constructor");
  }
}
public class PCAPP
{
   public static void main(String x[])
          {
                        C c1 = new C();
          }
}
```

Q2. How many ways to perform constructor chaining?

_____

There are two ways to perform constructor chaining in java

a) Using super () constructor: super constructor can use perform constructor chaining using inheritance

b) Using this() constructor: this () constructor can use perform constructor chaining using constructor overloading.

Q3. Can we use this() and super() at same time?

_____

No we cannot use super() and this() constructor at same time because both required first line of code.

Q4. Is it true super() constructor can call immediate parent?

_____

super() constructor can call only to immediate parent not call to grand parent or parent of parent

Q5. is it true super() constructor implicitly by child class constructor?

_____

Note when we have default constructor in parent class and if we create object of child class then child class constructor implicit call parent constructor using super() but when parent contain parameterized constructor then child class cannot decide the parameter value to parent constructor so it should pass by developer so we need to call super explicitly with parameter value.

Example with source code

```
class P
{ P()
  {   System.out.println("I am parent constructor ");
  }
}
```

```
class C extends P
{ C()
  {  super();
    System.out.println("I am child constructor");
  }
}
public class PCAPP
{
   public static void main(String x[])
          {   C c1 = new C();
          }
}
```

Final keyword

Final keyword is work as non-access specifier in JAVA

There are 8 non-access specifiers in JAVA

1. **Static 2. Final 3. Abstract 4. Synchronized 5. Volatile 6. Transient 7. Strictfp 8. Assert**

Final: is keyword we can use with variable, function and class

Final variable: final variable means variable cannot modify its value once we assign it means we can say final variable is specially design for declare the constant in JAVA.

```
public class FinalVarApp
{
   public static void main(String x[])
   {
        final int a=100;
        ++a;
        System.out.printf("A is %d\n",a);
   }
}
```

Note: if we think about left hand side code we get compile time error because we try to modify the value of final variable a and which is not possible so we get compile time error.

```
C:\Program Files\Java\jdk-20\bin>javac FinalVarApp.java
FinalVarApp.java:6: error: cannot assign a value to final variable a
        ++a;
          ^
1 error

C:\Program Files\Java\jdk-20\bin>
```

Final Method in JAVA

Final Method means method cannot override in child class

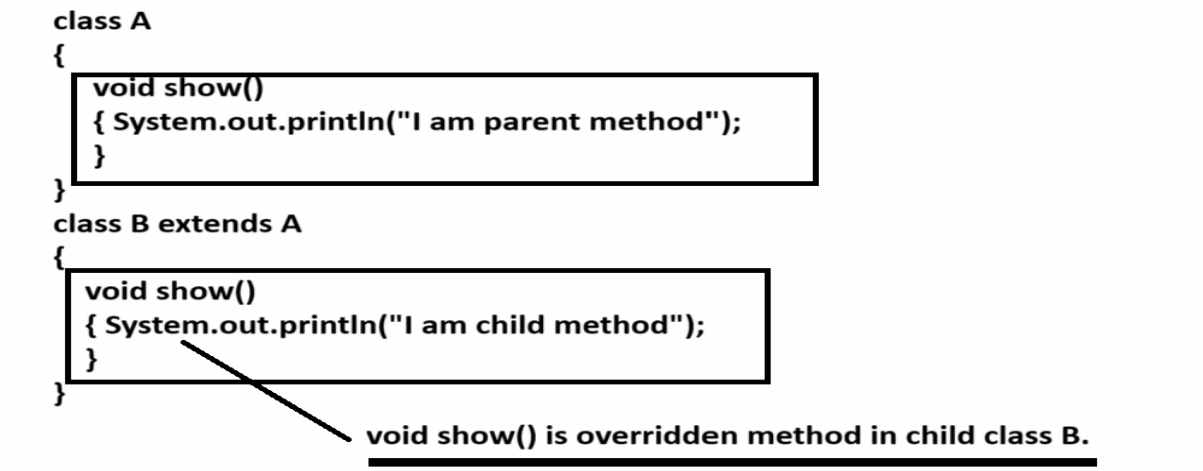
Note: Before learn final method, we need to know

Q. What is method overriding?

Method overriding means when we define method in parent class and redefine again method in child class called as method overriding.

Means in the case of method overriding parent method name and child method name must be same, parent return type and child return type must be same and parent parameter type and number of parameter list must be same

Means signature / syntax of parent method and child method must be same.



Note: If we think about method overriding when we create object of child class and try to call overridden method then by default child logics get executed.

```
class A
{
   void show()
   { System.out.println("I am show method in A");
   }
}
class B extends A
{
    void show()
    { System.out.println("I am show method in B");
    }
}
public class OverrideApp
{   public static void main(String x[])
    {
        B b1 = new B();
        b1.show();
    }
}
```

Note: if we think about left hand side code we create object of class B and call show() method then we get by default logic of child because in overriding child logic get executed.

```
C:\Program Files\Java\jdk-20\bin>javac OverrideApp.java

C:\Program Files\Java\jdk-20\bin>java OverrideApp
I am show method in B

C:\Program Files\Java\jdk-20\bin>
```

Q1. Why use method overriding?

a) **To Customize parent logic in child class According to child requirement**
b) **To Achieve dynamic polymorphism**

Q2. Is Method overriding beneficial or not?

It is dependent on the scenario some time method overriding is beneficial and sometime not.

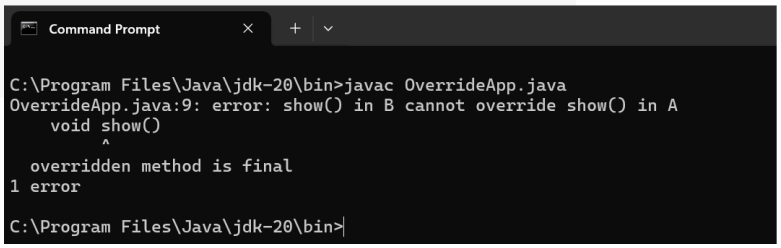
Q. How we can avoid method overriding?

Or how we can secure parent logic from child modification by using overriding?

If we want to avoid method overriding in java, we need to define parent method as final means when we define parent method as final then we cannot override the final method in child and cannot modify the parent logic in child class so we can say with the help of final method we can avoid method overriding and secure parent logic from child class.

```
class A
{
   final void show()
   { System.out.println("I am show method in A");
   }
}
class B extends A
{
    void show()
    { System.out.println("I am show method in B");
    }
}
public class OverrideApp
{   public static void main(String x[])
    {
        B b1 = new B();
        b1.show();
    }
}
```

Note: if we think about left hand side code we get compile time error because we have final void show() in parent class and we try to override the final method in child class B and which is not possible so we get compile time error.



Q. is it true final method can inherit?

Yes final method can inherit means we can call the final method by using child object

Q. is it true final method can override?

No final method cannot override because using overriding we try to modify the logic of parent method and final cannot allow modify the parent logic in child class so this is major reason we cannot override final method.

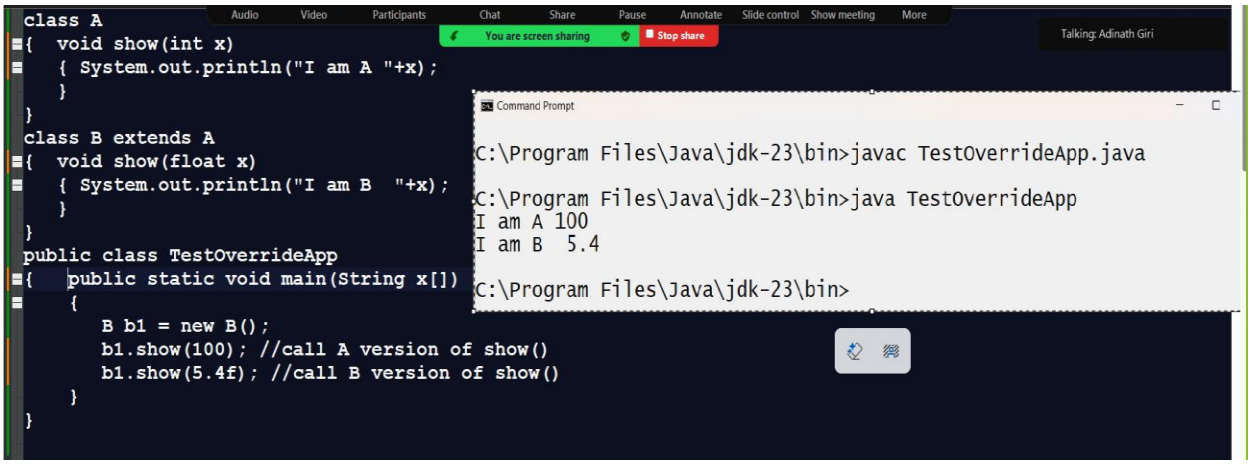
Q. Can we give scenario where we can implement overriding and avoid overriding?

Suppose consider we are working on banking application and we have one parent class name as RBI and we have two child classes name as Cooperative and Nationalize suppose consider RBI provide some rules and norms for account opening document which is not customizable by any bank means bank must be follow that instruction provided by RBI but RBI provide flexibility to bank for modify the interest rate according their profit and expenses so here every bank customize the interest rate on loan amount as per their requirement means here we can say account opening document method must be final means avoid override it but interest rate method can customizable by child means can override in child class shown in following code.

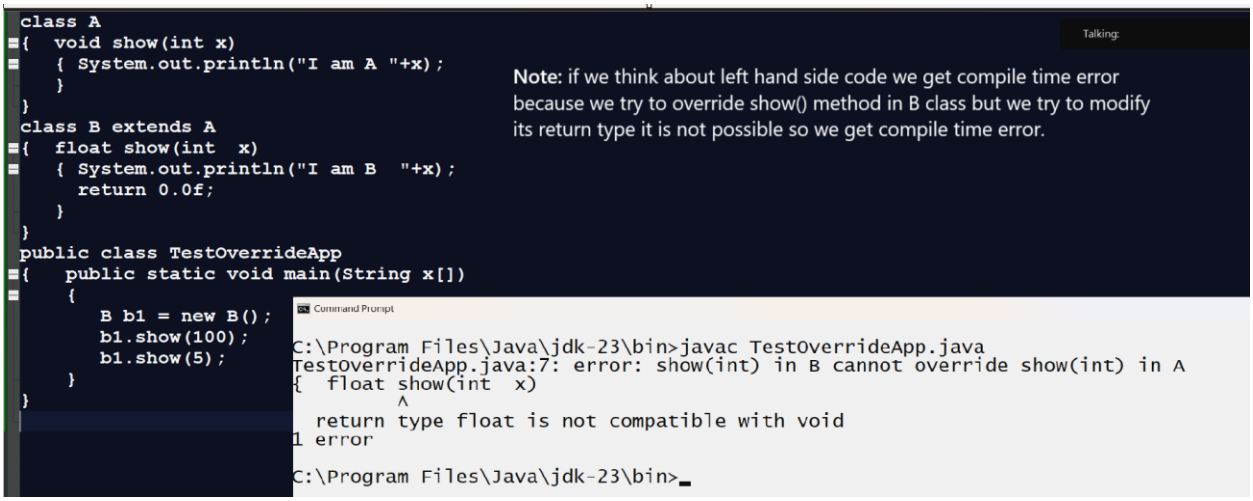
```
class RBI
{ int irate;
  final void accOpenDoc() //avoid overriding
    {  System.out.println("Adhar/PAN/VOTER Id/Driving licence mandatory");
    }
   void initializeMaxInterestRate()
   { irate=16;
   }
}
class Cooperative extends RBI
{  void initializeMaxInterestRate() //we implement overridng
   { irate=8;
   }
}
class Nationalize extends RBI
{ void initializeMaxInterestRate()
   { irate=7;
   }
}
```

If we want to work with method overriding in Java we need to know the some important points related with method overriding

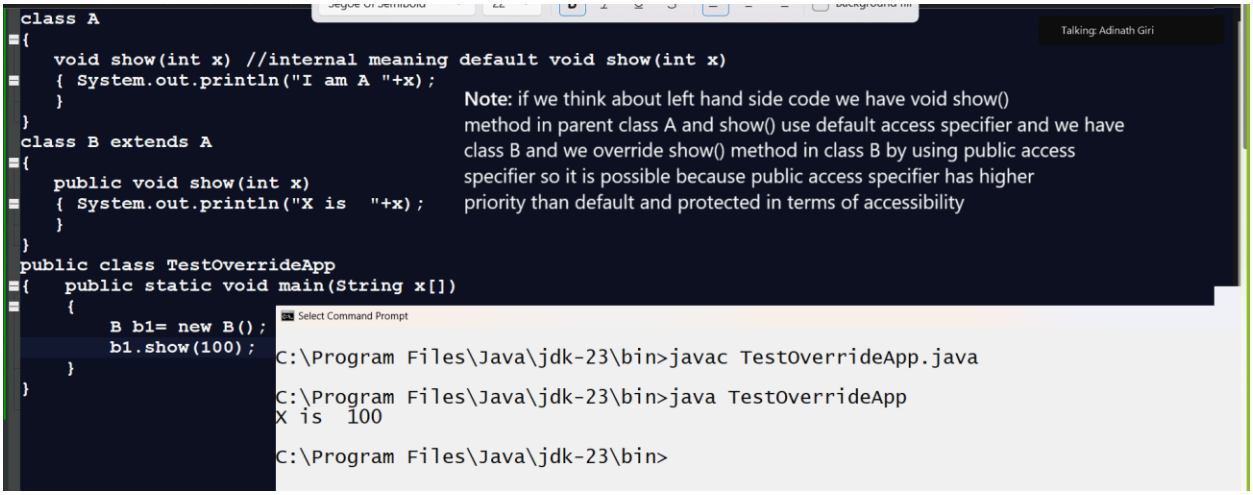
1. If we use same method name in parent and child class but pass different parameter then it is consider as overloading using inheritance.



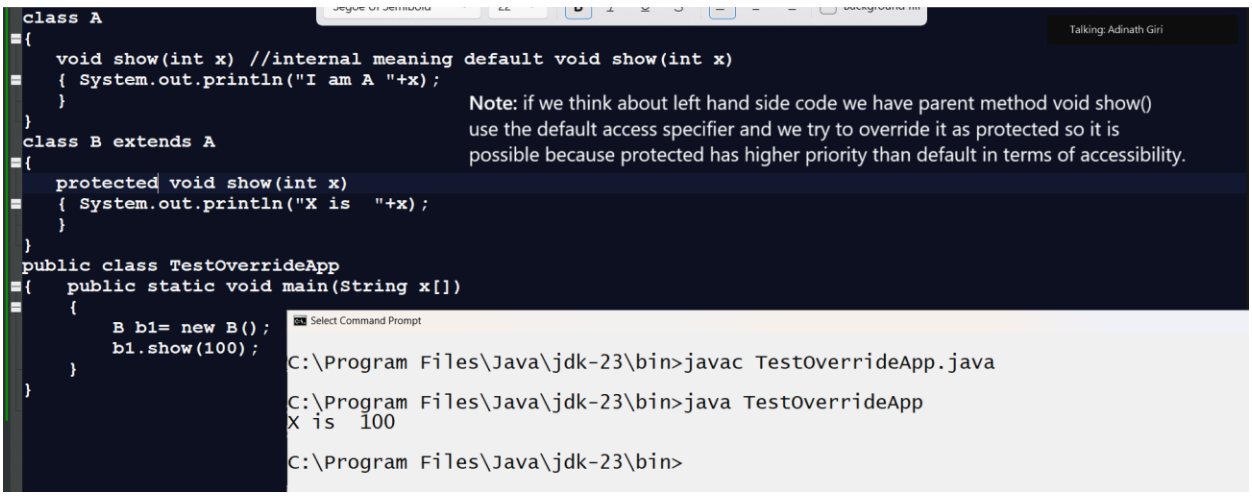
2. If we think about overriding return type is considered means return type of method in parent and child must be same



3. If your parent method is default then we can override it as public or protected

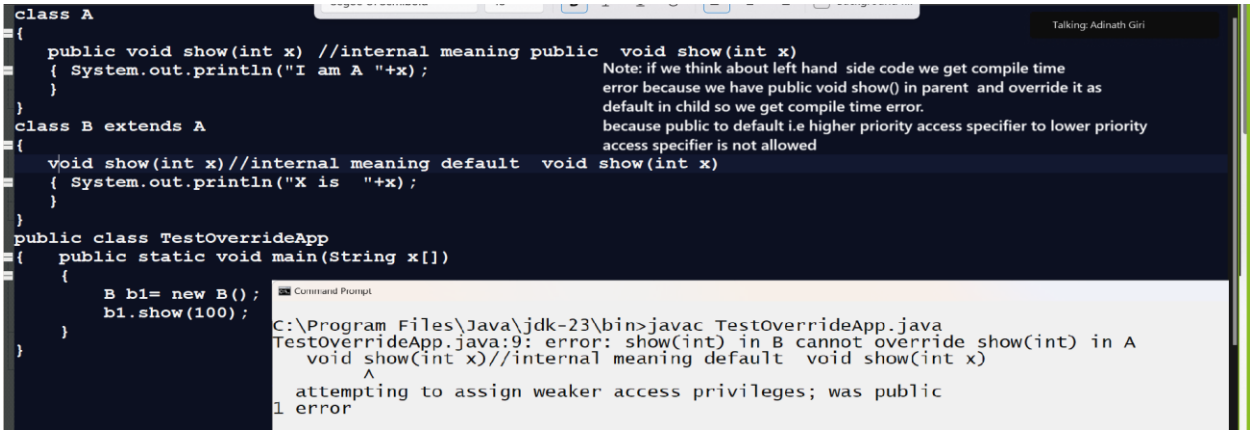


or

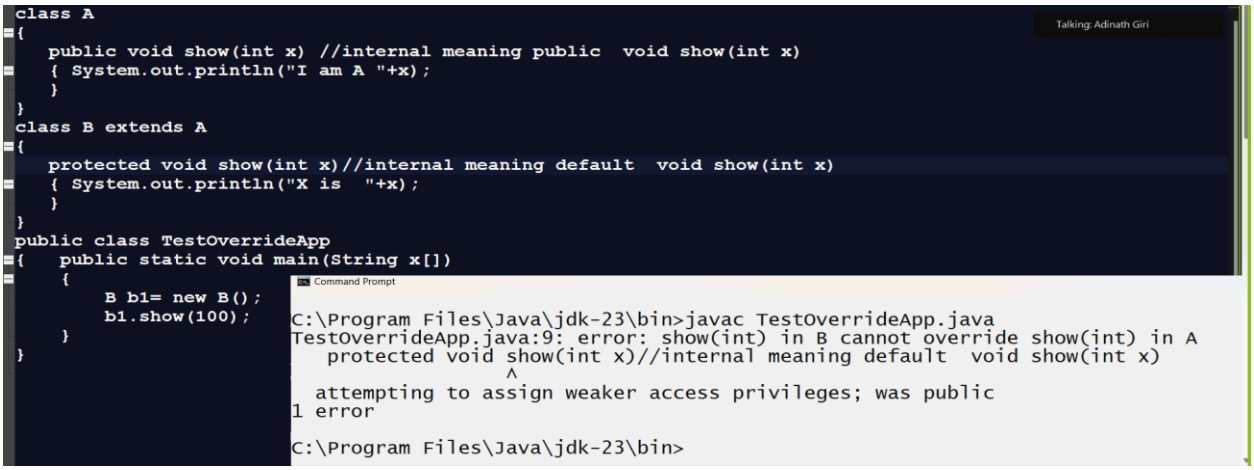


4. If your parent method is public then we cannot override it as protected or default or private

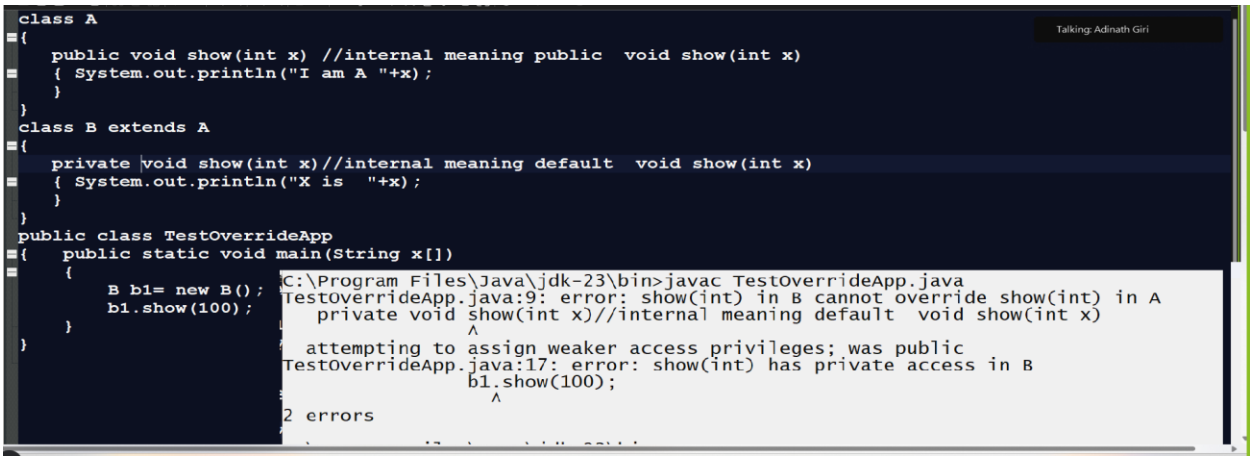
Note: when your parent method is public then child method must be public if we override it because higher priority access specifier to lower priority access specifier overriding is not possible.



or

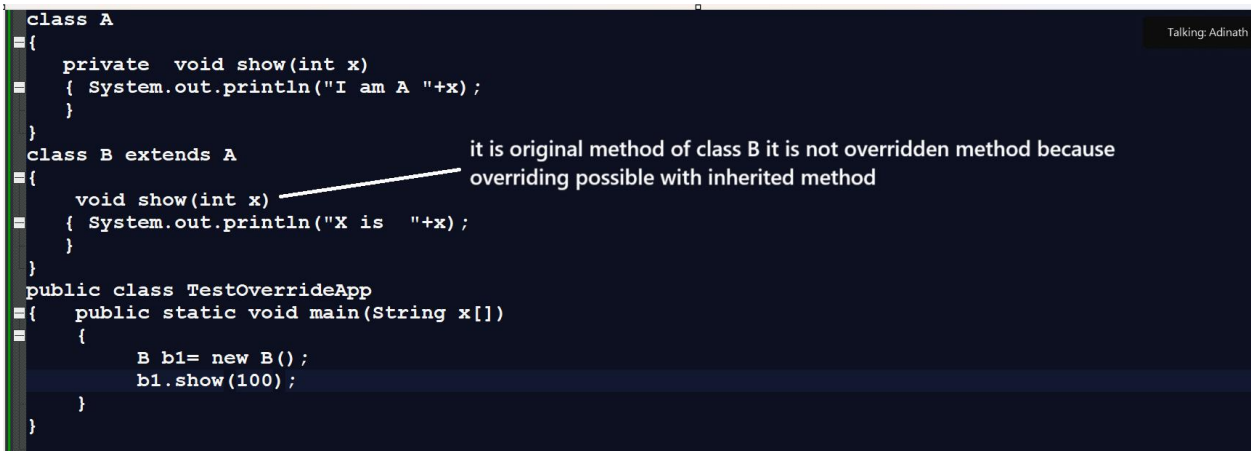


or



Q. Can we override private method?

No we cannot override private method because private access specifier not support to inheritance and overriding cannot work without inheritance so we cannot override private.



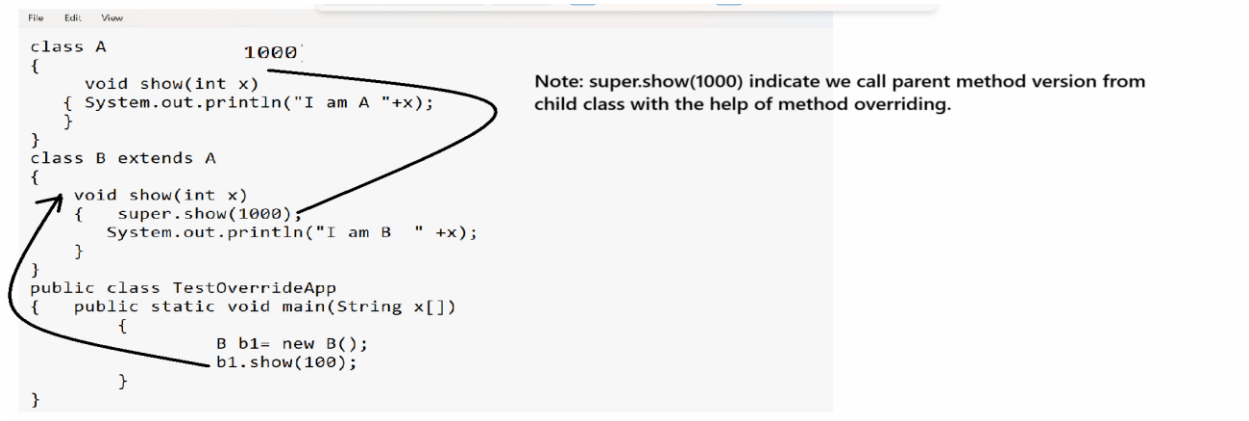
If we think about above code we have same method in class A and in class B name as void show() and we declare parent method as private so it look like as overriding but it is not overriding internally

How we can prove above code not implement the method overriding concept?

If we want to cross verify it is not method overriding we can use super keyword or reference

Q. What is super reference?

Super reference is internal reference present in child class which is used for call parent member from child class normally super recommend in method overriding



Note: if we I declare parent class method as private then we cannot call parent method with the help of super from child because super can call only those method which are inherited by parent in child or may

be override and private cannot support and override not work without inheritance so private cannot override shown in following screen shot

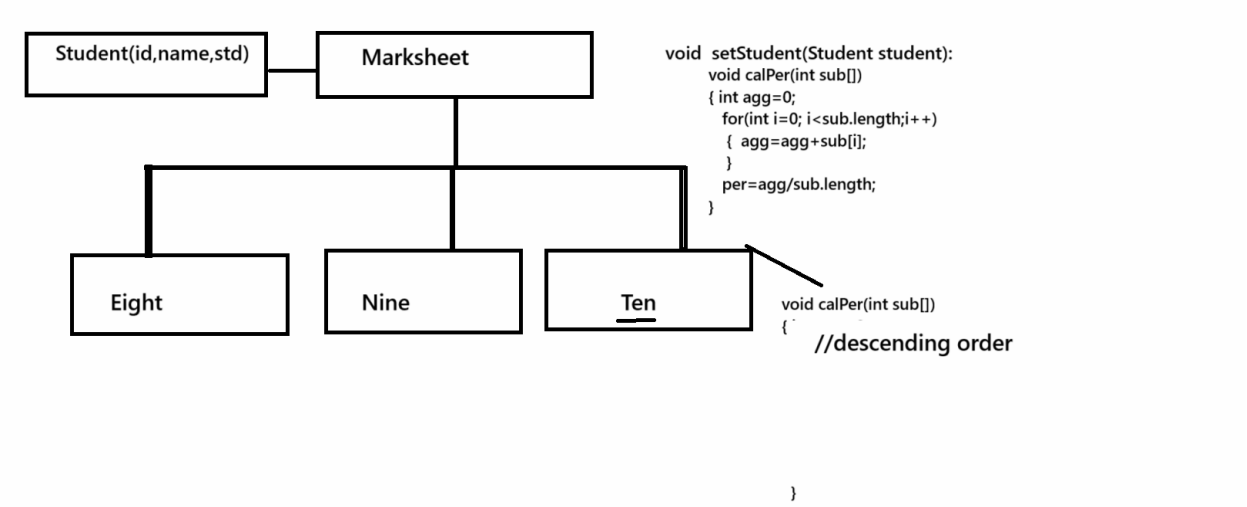
```
class A
{
   private void show(int x)
   { System.out.println("I am A "+x);
   }
}
class B extends A
{
    void show(int x)
    {   super.show(1000);
       System.out.println("I am B   " +x);
    }
}
public class TestOverrideApp
{   public static void main(String x[])
        {       B b1= new B();
                b1.show(100);
        }
}
```

```
Select Command Prompt

C:\Program Files\Java\jdk-23\bin>javac TestOverrideApp.java
TestOverrideApp.java:10: error: show(int) has private access in A
    {   super.show(1000);
              ^
1 error

C:\Program Files\Java\jdk-23\bin>
```

Q. Why super is very more important in the case of overriding?

By default in the case of overriding when we create object of child class and try to call overridden method then by default child logics get execute but if we want to execute the parent logic in override then we can super keyword

Scenario : Suppose consider we want to design application for Marksheet Generation and we have three classes name as Eight Nine and Ten and we have calPer() method which present in MarkSheet class which is parent of all classes and calPer() calculate the percentage by using best of six rules but we want to provide best five for tenth student but whose passing year is greater than 2015



Example with source code

Student.java

```
class Student
{  private int id;
   private String name;
   private String std;
   public Student(){
   }
   public Student(String name,int id,String std)
    { this.name=name;
      this.id=id;
      this.std=std;
   }
   public void setId(int id)
   { this.id=id;
   }
   public int getId()
   { return id;
   }
   public void setName(String name)
   { this.name=name;
   }
   public String getName()
   { return name;
   }
   public void setStd(String std)
    { this.std=std;
    }
```

```
  public String getStd()
   { return std;
   }
}
```

MarkSheet.java

```
class MarkSheet
{  protected Student student;
  protected int per;
  public void setStudent(Student student)
  {  this.student=student;
  }
  public void calPer(int subs[])
  {    int agg=0;
    for(int i=0;i<subs.length;i++)
     {  agg=agg+subs[i];
     }
    per=agg/subs.length;

  }
}
```

Eight.java

```
class Eight extends MarkSheet
{
   public void showEightResult()
   {
     System.out.println("Student Info");
      System.out.println(student.getId()+"\t"+student.getName()+"\t"+student.getStd());
    System.out.println("Marks Detail");
```

```
   System.out.println("Your Marks is  "+per);
  }
}
```

Nine.java

```
class Nine extends MarkSheet
{
  public void showNineResult()
  {
    System.out.println("Student Info");
     System.out.println(student.getId()+"\t"+student.getName()+"\t"+student.getStd());
   System.out.println("Marks Detail");
   System.out.println("Your Marks is  "+per);
  }
}
```

Ten.java

```
class Ten extends MarkSheet
{  private int year;
  public void setAdmissionYear(int year)
  {  this.year=year;
  }
   public void showTenResult()
  {
    System.out.println("Student Info");
     System.out.println(student.getId()+"\t"+student.getName()+"\t"+student.getStd());
      System.out.println("Marks Detail");
    System.out.println("Your Marks is  "+per);
  }
  public void calPer(int subs[])
```

```
 {
   if(year<=2015)
   { super.calPer(subs); //call parent logics
   }
else{
   int agg=0;
    for(int i=0; i<subs.length;i++)
    {
          for(int j=(i+1); j<subs.length; j++)
          {
                if(subs[i]<subs[j])
                {
                  int temp=subs[i];
                  subs[i]=subs[j];
                  subs[j]=temp;
                }
          }
    }
          System.out.println("Display array values");
          for(int i=0;i<(subs.length-1);i++)
          {  agg=agg+subs[i];
          }
     per=agg/(subs.length-1);
 }
 }

}
```

SuperTestApp.java

```
public class SuperTestApp
{
  public static void main(String x[])
  {
        Student s1 = new Student();
        s1.setId(1);
        s1.setName("Ram");
        s1.setStd("Eight");

        Student s2=new Student();
     s2.setId(2);
        s2.setName("Shyam");
        s2.setStd("Nine");

        Student s3 = new Student("Ganesh",3,"Ten");

     int a[]=new int[]{60,90,80,50,40,70};

    Eight e = new Eight();
        e.setStudent(s1);
        e.calPer(a);
        e.showEightResult();

        Nine n = new Nine();
        n.setStudent(s2);
        n.calPer(a);
        n.showNineResult();
```

```
        Ten t = new Ten();
        t.setStudent(s3);
        t.setAdmissionYear(2024);
        t.calPer(a);
        t.showTenResult();

  }
}
```

Q. How we can achieve dynamic polymorphism by using method overriding?

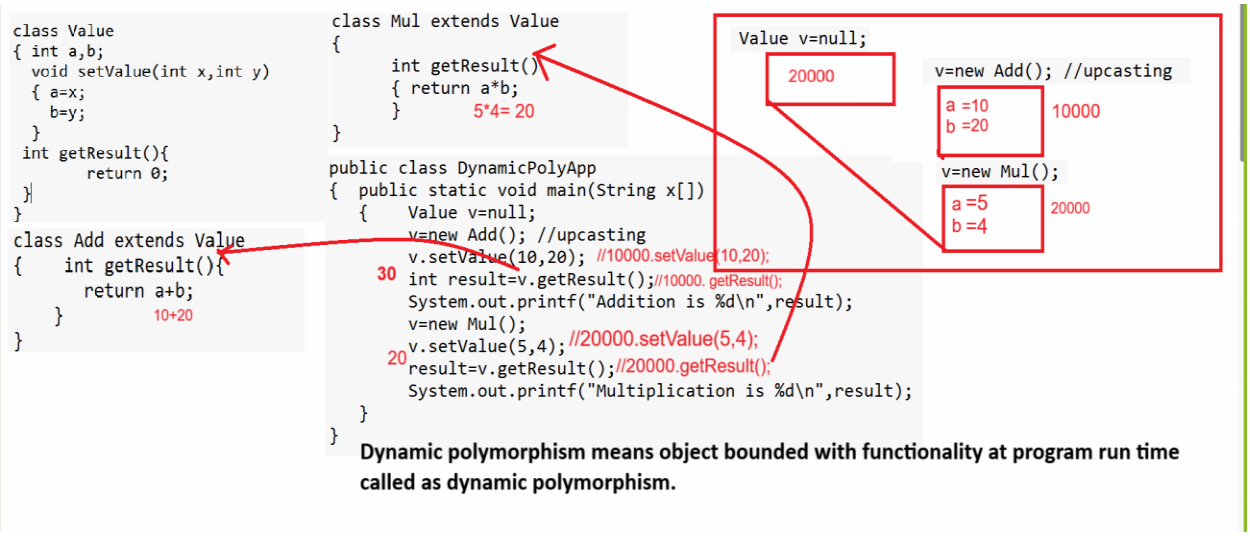
Note: before understand implementation of dynamic polymorphism we should have to

Q. What is dynamic polymorphism?

Dynamic polymorphism means object bounded with functionality at program run time called as dynamic polymorphism.

If we want to achieve dynamic polymorphism we need to create object of child class and reference of its parent class. When we create reference of parent and store child object address in it known up casting technique.

Following example shows the dynamic polymorphism?



Note: if we think about above code we have main method class name as DynamicPolyApp which contain reference Value v=null here we create reference of parent class mark it as null initially and we have statement v = new Add() means we create object of Add class whose address is 10000 and we store this address in reference variable v and we have statement v.setValue(10,20) means 10000.setValue(10,20) means we store 10 and 20 in child class object whose name is Add by parent referenced and we have statement v.getResult() means 10000.getResult() here we call getResult() method using parent reference but from Add class object because we have address of Add class object in parent reference means if we think about overriding when we use parent class reference and store child object in it and call overridden then child version get executed by parent reference and again we have v=new Mul() means here we create object of Mul class and store its address in reference variable which is we consider 20000 means our reference variable v release the address of Add class object and point new object in memory i.e to Mul object means we change object at program run time and after that we have statement v.setValue(5,4) means 20000.setValue(5,4) means we store 5 and 4 in Mul object whose address is 20000 and again we have statement result=v.getResult() means 20000.getResult() here we call version of getResult() from Mul class

Conclusion: here we have getResult() method and we write its two different logics one in Add class where we return addition means getResult() behave as addition in Add class and second in Mul class where we return multiplication logics means getResult() behave as multiplication in Mul class and here we use overriding technique means getResult() is same method with two different logics so this scenario indicate we implement polymorphism but if we think in main method we call getResult() two time with the help of reference v but v call getResult() by Add object as well as Mul object means v change object program run time for call getResult() method means here we have polymorphism but version should execute is dependent on run time object so we can say above code say implementation of dynamic polymorphism and we achieve it by using method overriding .

Q. What is the benefit of dynamic polymorphism?

The benefit of dynamic polymorphism is to achieve loose coupling.

Note: before understanding loose coupling we need to know

Q. What is coupling & how many ways to implement coupling?

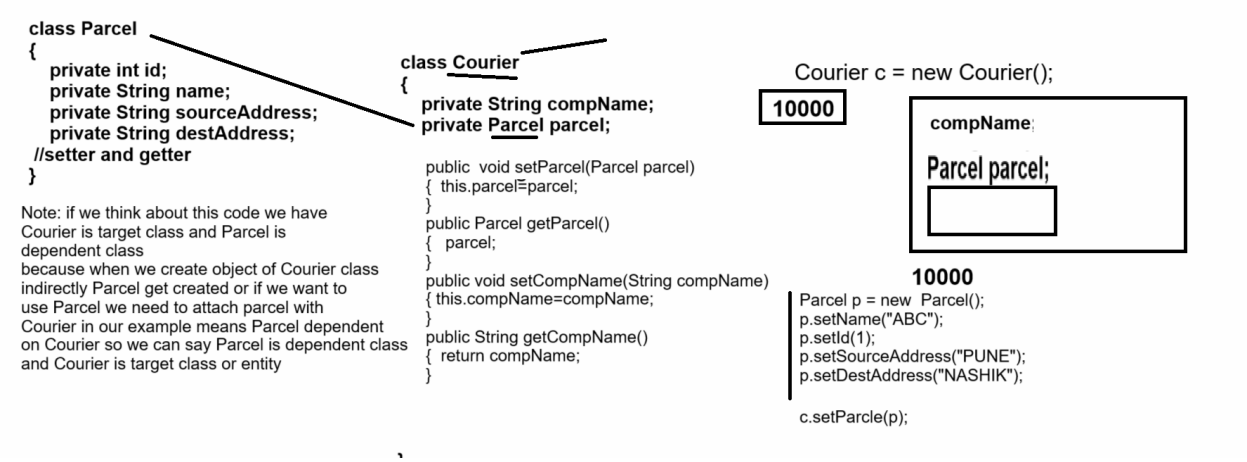
Coupling means when one object is dependent on another object called as coupling or dependency injection.

If we want to work with coupling we should have two important terms

a) Target class: Target class means class which object created by developer

b) Dependent class: Dependent class means class whose object should use or should create is dependent on target class.

We want to understand the Target class and dependent class concept with the help of Example.



There are two types of coupling?

---

a) Tight coupling: tight coupling means if target class is 100% dependent on particular dependent class reference called as tight coupling.

b) Loose Coupling: Loose coupling means if target is partially dependent on dependent parent class reference called as loose coupling and it is achieve by using loose coupling as well as with the help of abstract class and interface also.

Now we want to implement the tight coupling scenario

Suppose consider we want to design calculator and we have following classes.

Value : this is the parent class of Add and Mul and it contain two methods

void setValue(int x,int y): this method is used for accept two parameter using Add and Mul class object and if we want to addition or multiplication we required two values it is common term between Add and Mul so we try to define it in parent so reuse it easily

int getResult(): this method initially return 0 in parent and we want to override this method in Add and Mul class

Add: Here Add is child class of Value and we want to override getResult() in Add and return addition logics

Mul : Here Mul is child class of Value class and we want to override getResult() in Mul and return multiplication logics

Calculator: this is the target class and which one method name as

void performOperation(Add ad): this method accept reference of Add class and generate result.

Example with source code

```
class Value
{ int a,b;
  void setValue(int x,int y)
  { a=x;
    b=y;
  }
 int getResult(){
            return 0;
 }
}
class Add extends Value
{    int getResult(){
       return a+b;
    }
}
class Mul extends Value
{
     int getResult()
     { return a*b;
     }
}
class Calculator
{
   void performOperation(Add ad)
   {    int result=ad.getResult();
            System.out.println("Result is  "+result);
   }
```

```
}

public class DynamicPolyApp

{  public static void main(String x[])

  {     Calculator c= new Calculator();

          Add ad1 = new Add();

          ad1.setValue(10,20);

          c.performOperation(ad1);

          Mul m1 = new Mul();

          m1.setValue(5,4);

          c.performOperation(m1);

  }

}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac DynamicPolyApp.java
DynamicPolyApp.java:38: error: incompatible types: Mul cannot be converted to Add
        c.performOperation(m1);
                           ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output
1 error
```

Note: if we think about above code we get compile time error because we have Calculator class which is our target class and we have one method in Calculator class name as performOperation(Add ad) and in main method we create object of Calculator class and try to pass Mul class reference in performOperation() which is not possible so we get compile time

Conclusion: the conclusion is if performOperation(Add ad) method accept parameter of Add class only then we cannot pass any other class reference in it means this method 100% dependent on Add class means your Target class which is Calculator class in our example is 100% dependent on Add class so this is tight coupling

If we want to solve this problem we have two solutions

a) Using compile time polymorphism: we can solve above problem with the help of method overloading concept means we overload the performOperation(Mul m) with Mul reference and in the case of overloading which method should execute dependent on parameter pass in it at the time of calling

```
class Value

{ int a,b;

  void setValue(int x,int y)

  { a=x;
```

```
    b=y;
  }
 int getResult(){
            return 0;
 }
}
class Add extends Value
{    int getResult(){
       return a+b;
    }
}
class Mul extends Value
{
     int getResult()
     { return a*b;
     }
}
class Calculator
{   void performOperation(Add ad)
   {    int result=ad.getResult();
          System.out.println("Result is  "+result);
   }
   void performOperation(Mul m)
   {    int result=m.getResult();
            System.out.println("Result is  "+result);
   }
}
public class DynamicPolyApp
```

```
{  public static void main(String x[])
  {       Calculator c= new Calculator();
          Add ad1 = new Add();
          ad1.setValue(10,20);
          c.performOperation(ad1); //call Add version
          Mul m1 = new Mul();
          m1.setValue(5,4);
          c.performOperation(m1); //call Mul version
  }
```

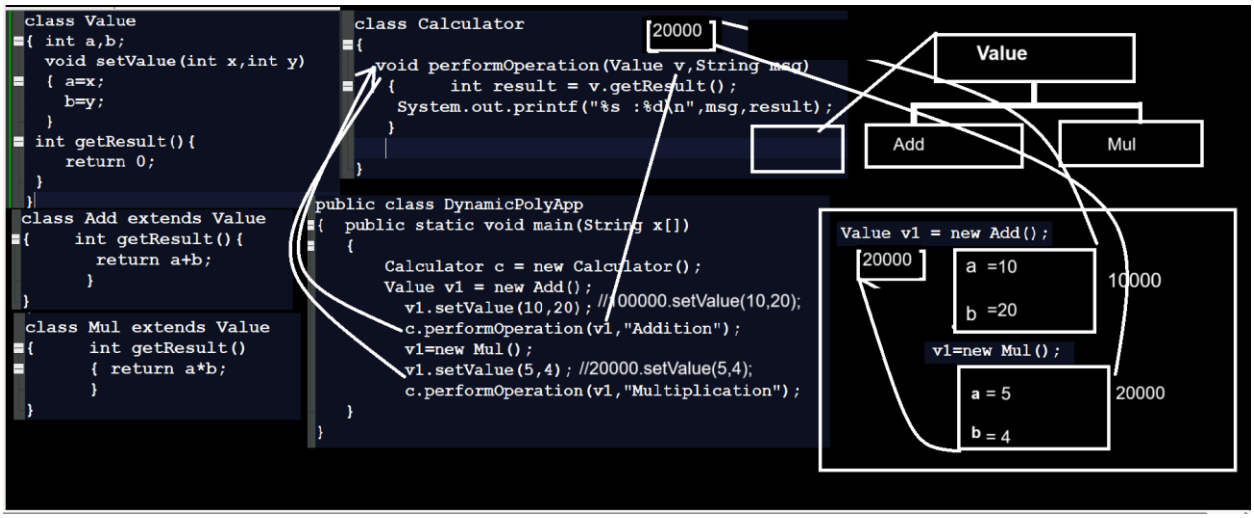
Note: if we think about above code we have overload performOperation() two times first with Add and second time with Mul so we resolve above problem but the limitation when we have 100 operations in Calculator class we need to overload performOperation() 100 times and it is not feasible in real time senario. So we want to define only one performOperation() method in Calculator class and method should able to work with 100 different objects at run time then loose coupling comes in picture or dynamic polymorphism comes in picture.

How to implement loose coupling by using dynamic polymorphism

---

If we want to implement loose coupling using dynamic polymorphism we need to pass parent class as reference in target class method as dependent class and when call method from target then pass reference of parent and object of child which we required

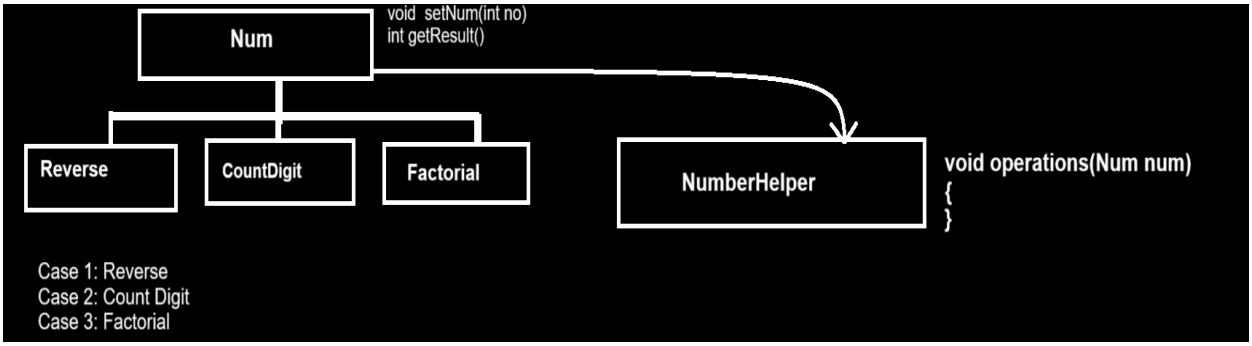
Note: when we pass parent class reference as parameter in method then we can pass any child class object in that method with the help of up casting technique



Example using dynamic polymorphism and loose coupling

---

we want to design application for Numeric operation for single value



Example with source code

```
import java.util.*;
class Num
{ protected int no;
  void setValue(int no)
  {this.no=no;
  }
   int getResult(){
     return 0;
   }
}
class Reverse extends Num
{  int rev=0,rem;
   int getResult()
   {  while(no!=0)
      {
                rem = no % 10;
                        no  = no /10;
```

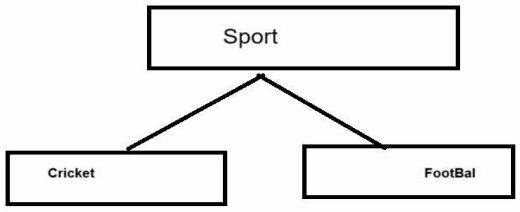
```
                    rev=rev*10+rem;
            }
            return rev;
  }
}
class CountDigit extends Num
{   int count=0;
    int getResult()
            {  while(no!=0)
              { no = no/10;
                    ++count;
                  }
                  return count;
            }
}
class Factorial extends Num
{ int f=1;
  int getResult()
  {
   for(int i=1;i<=no;i++)
            { f=f*i;
            }
            return f;
  }
}
class NumberHelper
{
  void operation(Num num)
```

```
  {  int result=num.getResult();
    System.out.printf("Result is %d\n",result);
  }
}
public class NumDynamicPolyApp
{
  public static void main(String x[])
  {  Scanner xyz  = new Scanner(System.in);
    System.out.println("Enter your choice");
            int choice=xyz.nextInt();
            System.out.println("Enter number\n");
            int no=xyz.nextInt();
            NumberHelper h=new NumberHelper();
    Num n=null;
    switch(choice)
    {
              case 1:
                    n=new Reverse();
                    n.setValue(no);
                    h.operation(n);
                   break;
                   case 2:
                   n=new CountDigit();
                   n.setValue(no);
                   h.operation(n);
                   break;
                   case 3:
                   n=new Factorial();
```

```
                    n.setValue(no);
                    h.operation(n);
                    break;
                    default:
                    System.out.println("Wrong choice");
    }
  }
}
```

Q. What is upcasting?

Upcasting means we convert child object in to parent object called as up casting means in the case of up casting normally we create reference of parent and store child object address in it or reference in it.



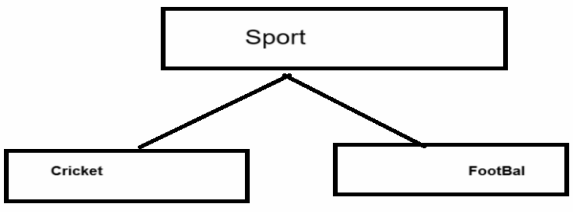

```
class  Gathering
{
   void sportActivity(Sport sport)
   {

   }
}
```

```
Sport s = new Circket();
```

here we have Cricket is our child object and Sport is our parent object means we convert child object in to parent object called as upcasting

Q. What is downcasting?

Down Casting means convert parent object in to the child object called as down casting



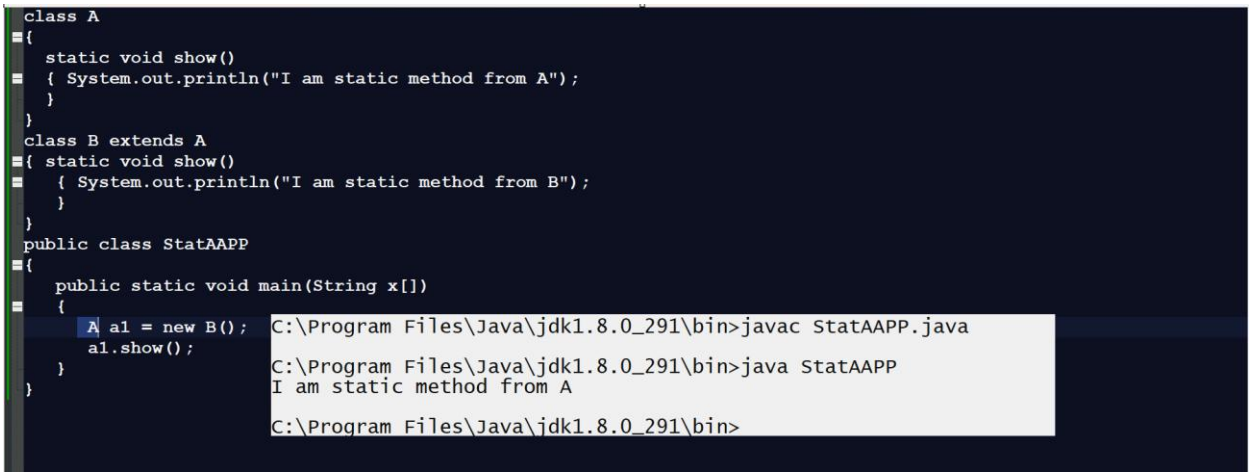

```
class  Gathering
{
   void sportActivity(Sport sport)
   {

   }
}
```

```
Sport   s = new Cricket();
Cricket c=(Cricket)s; //downcasting
```

Q. Can we override static method?

No we cannot override static method and when we try to override static method then it is not method overriding it is method hiding.

Method hiding means we hide the parent method override by child class and if we want to check method hiding then create reference of parent and object child and try to call static method then we get logic from parent class even static method override in child class called as method hiding .



```
class A
{
  static void show()
  { System.out.println("I am static method from A");
  }
}
class B extends A
{ static void show()
   { System.out.println("I am static method from B");
   }
}
public class StatAAPP
{
   public static void main(String x[])
   {
      A a1 = new B();
      a1.show();
   }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac StatAAPP.java

C:\Program Files\Java\jdk1.8.0_291\bin>java StatAAPP
I am static method from A

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Q. What is difference between method overriding and method hiding?

Overriding	**Hiding**
Overriding can happen with non static methods	Method hiding can perform with static method
Overriding always modify the logic of parent by child	Hiding not allow to customize logic even not provide method copy to child for modification
If we use the upcasting and try to call overridden method then by default child version get executed	If use the upcasting and try to call overridden method then by default parent logic executed

Q. Why we cannot override static method?

---

static method always support to compile time polymorphism and method overriding support to dynamic polymorphism means purpose static method is completely opposite of overriding so we cannot override static method.

Q. What is difference between compile time polymorphism and run time polymorphism?

Compile Time Polymorphism	**Runtime polymorphism**
Compile time polymorphism means functionally can bound with object at program compile time	Runtime polymorphism means functionality can bound with object at program run time
Compile time polymorphism can implement by using function overloading	Run time polymorphism can implement using method overriding
Compile time polymorphism can implement using inheritance or without inheritance	Runtime polymorphism cannot implement without inheritance
Compile time polymorphism can support to tight coupling implementation	Run time polymorphism can support to implement loose coupling
Compile time polymorphism is not efficient or not code maintainable if we think about large application	Runtime polymorphism is efficient and maintainable code large application as well as run time polymorphism is more efficient than compile time polymorphism
Compile time polymorphism can work with static method	Runtime polymorphism cannot work with static method

Q. What is difference between overloading and overriding?

Overloading	**Overriding**
Overloading is static binding or compile time polymorphism	Overriding is run time polymorphism
Overloading can develop using single class or may be using inheritance	Overriding cannot develop without inheritance it must be develop using inheritance.
In method overloading return type is not consider means we can give different return type to every method definition	In the case of overriding return type and signature of parent and child method must be same
Method overloading can work with final	Method overriding cannot work with final
Method overloading can support to static methods or can work with static method	Method overriding cannot work with static methods
Which method should execute is dependent on parameter list ,its data type and using its sequence in method overloading	Which method should execute is dependent on child class object and according to that version of method get executed.

Q. What is abstract class and abstract method in JAVA?

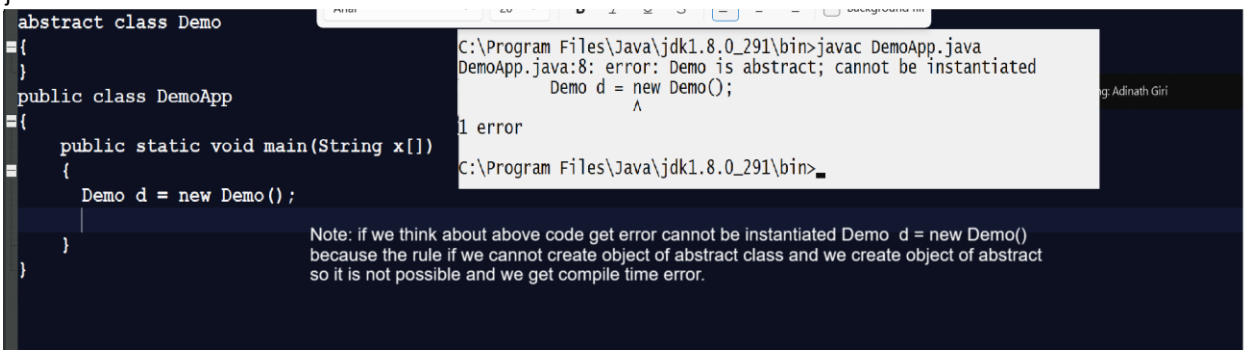
Abstract class means a class cannot create its object and abstract method means method cannot have logics and if want to declare abstract class and abstract method in java we have abstract keywords

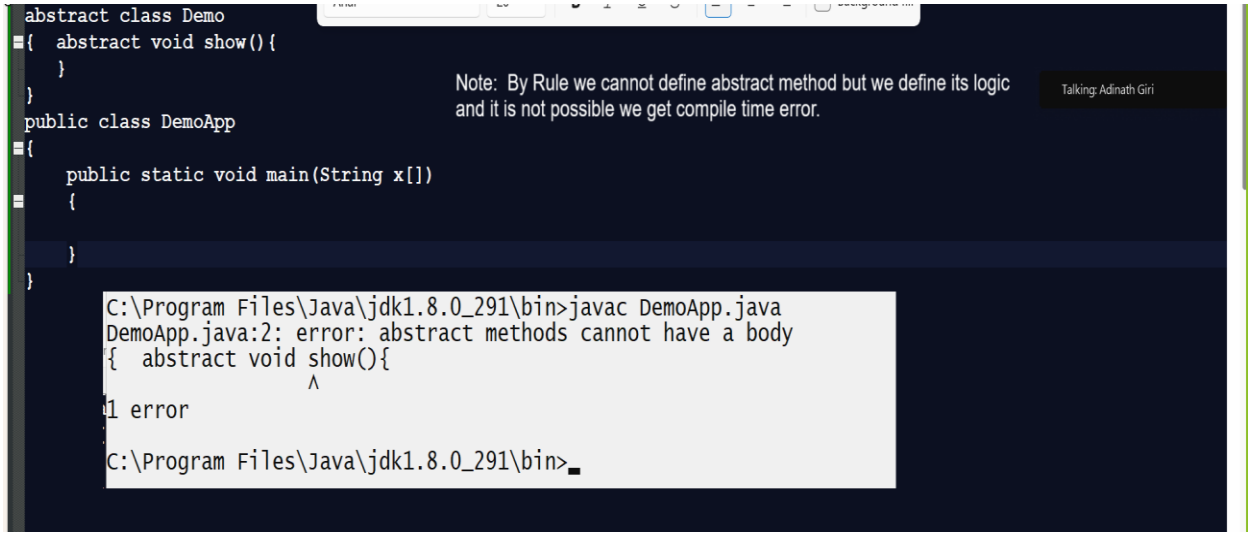
Syntax:

abstract class classname {

  abstract return type functionname(datatype variablename);

}





Q. Why we need to use abstract class and abstract methods?

---

1. To achieve abstraction

2. To achieve dynamic polymorphism and loose coupling.

Q. What is abstraction?

Abstraction means to hide the implementation detail from end user at design level called as abstraction means if we think about abstract we not provide method logic just we provide method template and we write its logic in future according to user requirement in future means abstract we know what to do but we don't how to do it. It is dependent on scenario and requirement.

Example: Suppose consider we want to design application VehicleManuFact company they want to design engine() for Vehicle but they want to manufacture different kinds of Vehicle and every Vehicle has engine() then we cannot provide single implementation of engine() to them so just we need to prototype of engine() to company we can customize that engine as per the vehicle category called as abstraction in short we can say every vehicle has engine but design of engine vary from vehicle to vehicle.

```
abstract class  Vehicle
{
  abstract void engine();
}
  Note: abstract method cannot have logic but we if we want to write logic of abstract method we can
  override abstract method in different child classes and can write logic of abstract method as per child requirement.
```

```
abstract class  Vehicle
{
  abstract void engine();
}
  Note: abstract method cannot have logic but we if we want to write logic of abstract method we can
  override abstract method in different child classes and can write logic of abstract method as per child requirement.

  class Bike extends Vehicle
  {
      void engine()
      { System.out.println("200 CC");
      }
  }
  class Car extends Vehicle
  {
     void engine()
     { System.out.println("1000 CC");
     }
  }
```

Example: Suppose consider we are working for ManPowerHiring Agency and the main role agency to recruit manpower or employee and every employee must have skills but skills are not fix it is vary from Designation to Designation

Here abstract is every employee need skill but vary according to position of employee so your abstraction like as

```
abstract class Employee
{  abstract void skillSet();
}
class Developer extends Employee
{
    void skillSet(){
      System.out.println("Should good in coding and communication");
     }
}
class HR extends Employee
{
   void skillSet(){
    System.out.println("Should good in communication and management skill");
   }
}
```

Means if we want to use abstract class we normally create object of child class and use the abstract members using child class object shown in following code.

```
abstract class Vehicle

{  abstract void engine();

}

class Bike extends Vehicle

{

            void engine()

            { System.out.println("100 CC");

            }

}

class Car extends Vehicle

{   void engine()

           { System.out.println("1000 CC");

           }

}

public class DemoApp

{

   public static void main(String x[])

           {
```

```
        Car c = new Car();
                    c.engine();
            Bike b =new Bike();
            b.engine();
        }
}
```

Important points related abstract class or Interview Question on abstract class and abstract method?

Q. What is abstract class and abstract method?

Abstract class means a class cannot create its object and abstract method means method cannot have logics and if want to declare abstract class and abstract method in java we have abstract keywords

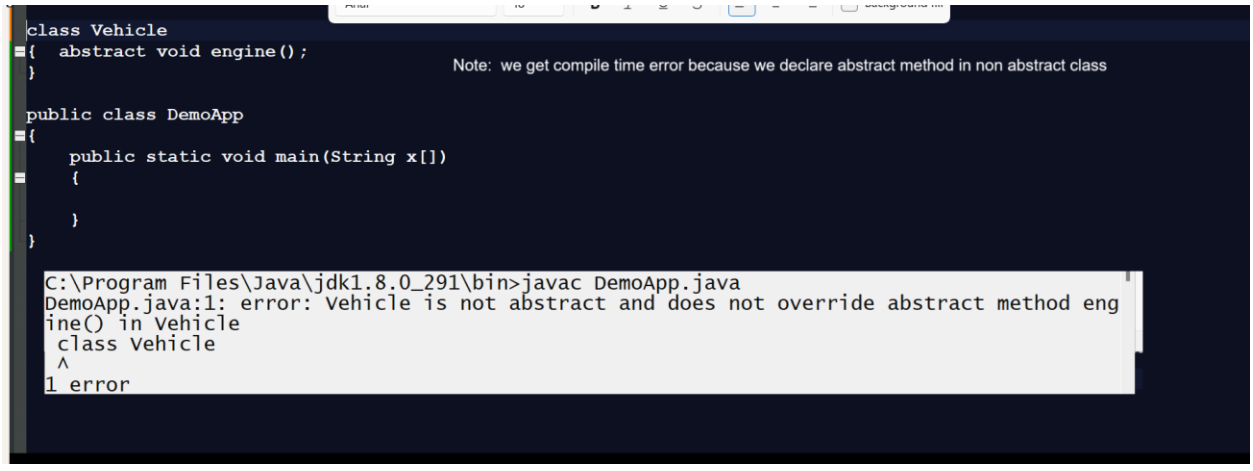
Q2. Why use abstract class and abstract method?

1. To achieve abstraction

2. To achieve dynamic polymorphism and loose coupling.

Q3. Can we declare abstract method in non abstract class?

No we cannot declare abstract method in non abstract class and if we try to declare it we get compile time error Means the conclusion is for abstract method class must be abstract



Q4. Why we cannot declare abstract method in non abstract class?

Suppose consider if java allow to declare abstract method in non abstract class then there is possibility we can create object of that class if class is non abstract and using that object abstract method may be call but if we want to call any method it must have definition and abstract method cannot have

definition for avoiding this problem java restrict us to declare abstract method in abstract class so this is major reason we cannot declare abstract method in non abstract class.

Q5. Can we define non abstract method in abstract class?

Yes we can define non abstract method in abstract class and when we define non abstract method in abstract class then class known as concrete class.

```
abstract class Vehicle
{
  abstract void engine();
  void defaultColor(){
    System.out.println("Black");
  }
}
```

Note: here Vehicle is concrete class because it contain one abstract method and one non abstract method.

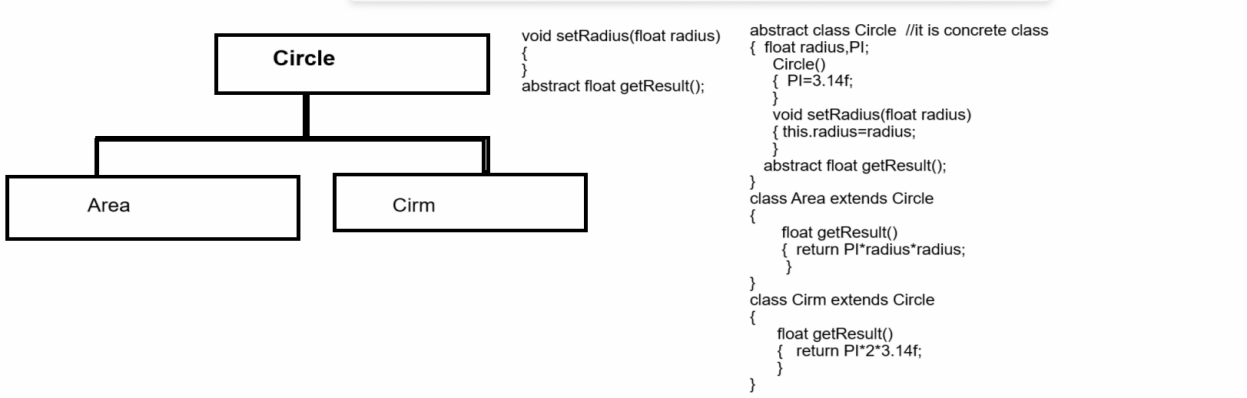
Q6. Can we declare constructor in abstract class?

Yes we can declare constructor in abstract class

```
abstract class Vehicle
{
    public Vehicle(){
    }
  abstract void engine();
}
```

Q7. Why we declare constructor in abstract class if abstract class not creates its object?

Abstract class constructor help us to initialize the abstract class member when we create object of child class because if we create object of child class then before child class constructor parent class constructor get executed so that constructor we can initialize the parent member



Note: when we want to initialize abstract class then we can define constructor within abstract class.

```
abstract class Circle
{ protected float radius,PI;


  public Circle()
  {   PI=3.14f;
  }
  void setRadius(float radius)
  {  this.radius=radius;
  }
  abstract float getResult();
}
class Area extends Circle
{
  float getResult()
  {  return PI*radius*radius;
  }
}
class Cirm extends Circle
{
 float getResult()
 {  return PI*2*radius;
 }
}
public class AbsConsApp
{
  public static void main(String x[])
  {
```

```
    Area a= new Area();
            a.setRadius(3.0f);
            float result=a.getResult();
            System.out.printf("Area of circle is %f\n",result);
            Cirm cm = new Cirm();
            cm.setRadius(3.0f);
            result=cm.getResult();
            System.out.printf("Cirmference of Circle is %f\n",result);
  }
}
```

Q7. Can we use abstract keyword with variable if yes then why if no then why?

---

we cannot use abstract keyword with variable the goal of abstract keyword is achieve abstraction and it is possible with the help of method overriding only because in abstract we can modify the logic of method in future but the goal of variable store only data not write logic so this is major we cannot use abstract keyword with variable

```
abstract class Circle
{  abstract float radius,PI;

   public Circle()
   {   PI=3.14f;
   }
   void setRadius(float radius)
   {  this.radius=radius;
   }
   abstract float getResult();
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac AbsConsApp.java
AbsConsApp.java:2: error: modifier abstract not allowed here
{  abstract  float radius,PI;
                 ^
AbsConsApp.java:2: error: modifier abstract not allowed here
{  abstract  float radius,PI;
                        ^
2 errors
```

Q8. is it true if abstract class contain more than one abstract methods then all method must be override where abstract class inherit?

---

yes it is true because if abstract class contain more than one abstract method then all method must be override where abstract class get inherit and need to override all method as blank if we not required.

Example with error

```
abstract class T
{
  abstract void s1();
  abstract void s2();
  abstract void s3();

}
class T1 extends T
{
   void s1()
  { System.out.println("I required s1 method");
   }
}
class T2 extends T
{
   void s2()
  { System.out.println("I required s2 method");
   }
}
class T3 extends T
{
   void s3()
  { System.out.println("I required s3 method");
   }
}
public class TestAbsMethodRuleApp
{   public static void main(String x[])
```

```
  {


        }
}
```

Output

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac TestAbsMethodRuleApp.java
TestAbsMethodRuleApp.java:8: error: T1 is not abstract and does not override abstract method s3() in T

class T1 extends T
^
TestAbsMethodRuleApp.java:14: error: T2 is not abstract and does not override abstract method s3() in
T
class T2 extends T
^
TestAbsMethodRuleApp.java:20: error: T3 is not abstract and does not override abstract method s2() in
T
class T3 extends T
^
3 errors
```

if we think about code we get compile time error because we have three non abstract class T1,T2 and T3 and abstract class name as T and in abstract class we have three abstract method name as s1(),s2(),s3() and we override s1() in T1, s2() in T2 and s3() in T3 but the rule is we required to override all abstract method if we inherit any abstract class in non abstract class so we required to override remaining two methods as blank in every child class

Example with source code

```
abstract class T

{

  abstract void s1();

  abstract void s2();

  abstract void s3();



}

class T1 extends T

{

   void s1()

   { System.out.println("I required s1 method");

   }

   void s2()
```

```
  {
  }
  void s3(){
  }
}
class T2 extends T
{
  void s2()
  { System.out.println("I required s2 method");
  }
  void s1()
  {
  }
  void s3()
  {
  }
}
class T3 extends T
{
  void s3()
  { System.out.println("I required s3 method");
  }
  void s1(){
  }
  void s2(){
  }
}
public class TestAbsMethodRuleApp
```

```
{   public static void main(String x[])
   {
          T1 t1 = new T1();
                t1.s1();


                T2 t2 = new T2();
                t2.s2();


                T3 t3 = new T3();
                t3.s3();
        }
}
```

Output

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac TestAbsMethodRuleApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java TestAbsMethodRuleApp
I required s1 method
I required s2 method
I required s3 method

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Note: the limitation of above approach is if we have abstract class with 100 methods and we have 10 child classes for that abstract class and we required to override single method from abstract class in every child class then we required to override 1000 methods but we need to only 10 and 999 need to define blank

If we want to solve this problem we have solution name as adapter class

Q9. What is adapter class and why use it?

Adapter class is intermediate class which contain all blank method of abstract class or override all blank method of abstract class or interface and which is able to provide specific method to abstract child class called as adapter class.

```
abstract class T

{  abstract void s1();

  abstract void s2();
```

```
  abstract void s3();


}
class ADP extends T
{   void s1()
    {
    }
    void s2()
            {
            }
            void s3()
            {
            }
}
class T1 extends ADP
{
   void s1()
   { System.out.println("I required s1 method");
   }


}
class T2 extends ADP
{
   void s2()
   { System.out.println("I required s2 method");
   }


}
```

```
class T3 extends ADP
{
  void s3()
  { System.out.println("I required s3 method");
  }
}
public class TestAbsMethodRuleApp
{   public static void main(String x[])
   {                T1 t1 = new T1();
                    t1.s1();
                    T2 t2 = new T2();
                    t2.s2();
                    T3 t3 = new T3();
                    t3.s3();
          }
}
```

Q10. Can we create reference of abstract class and why use it?

We cannot create object of abstract class but can create its reference and if we want to create reference of abstract class we need to use up casting technique or need to create object of its child class and the goal of abstract class reference is achieve loose coupling & dynamic polymorphism

```
abstract class Arr
{   int arr[];
  void setArr(int arr[])
  {  this.arr=arr;
  }
  abstract int [] getResult();
```

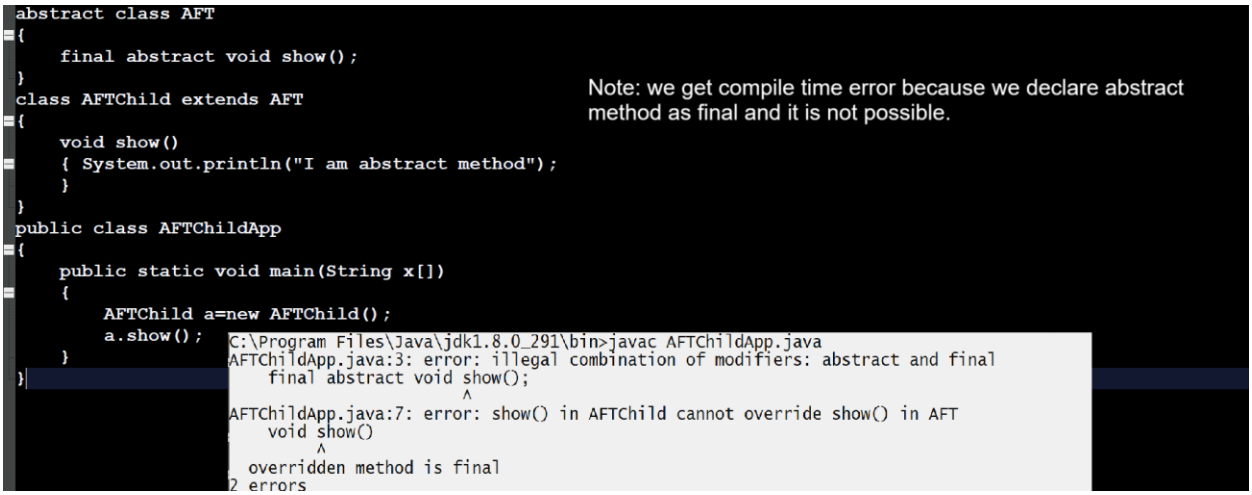
```
}
class Rev extends Arr
{
  int [] getResult()
  {  int mid=arr.length/2;
     int end=arr.length-1;
            int start=0;
            while(start<mid)
            {  int temp=arr[start];
               arr[start]=arr[end];
                     arr[end]=temp;
                     start++;
                     end--;
            }
            return arr;
  }
}
class Sort extends Arr
{
  int [] getResult()
  {
     for(int i=0; i<arr.length;i++)
            {
               for(int j=(i+1); j<arr.length;j++)
                     {
                        if(arr[i]>arr[j])
                              {
                                 int temp=arr[i];
```

```
                    arr[i]=arr[j];
                    arr[j]=temp;
                  }
              }
          }
          return arr;
  }
}
class ArrDS
{
  void applyAlgo(Arr a)
  {
    int []result=a.getResult();
          System.out.println("\nDisplay Array\n");
          for(int i=0;i<result.length;i++)
          {
           System.out.printf("%d\t",result[i]);
          }
  }
}
public class DynamicPolyUsingAbs
{
  public static void main(String x[])
  {     ArrDS ads=new ArrDS();
              int a[]=new int[]{5,3,4,2,1};
              System.out.println("Original Array");
              for(int i=0;i<a.length;i++)
              {
```

```
                System.out.printf("%d\t",a[i]);
            }
            System.out.println();
            Arr ar =new Rev();
              ar.setArr(a);
               ads.applyAlgo(ar); //work with reverse

                    ar=new Sort();
                    ar.setArr(a);
                    ads.applyAlgo(ar); //work with sort

  }
}
```

Q11. Can we declare abstract method as final?

No we cannot declare abstract method as final because abstract method cannot work without inheritance and overriding and final method not support to overriding so we cannot declare abstract method as final and if we try to declare it as final then we get compile time error.

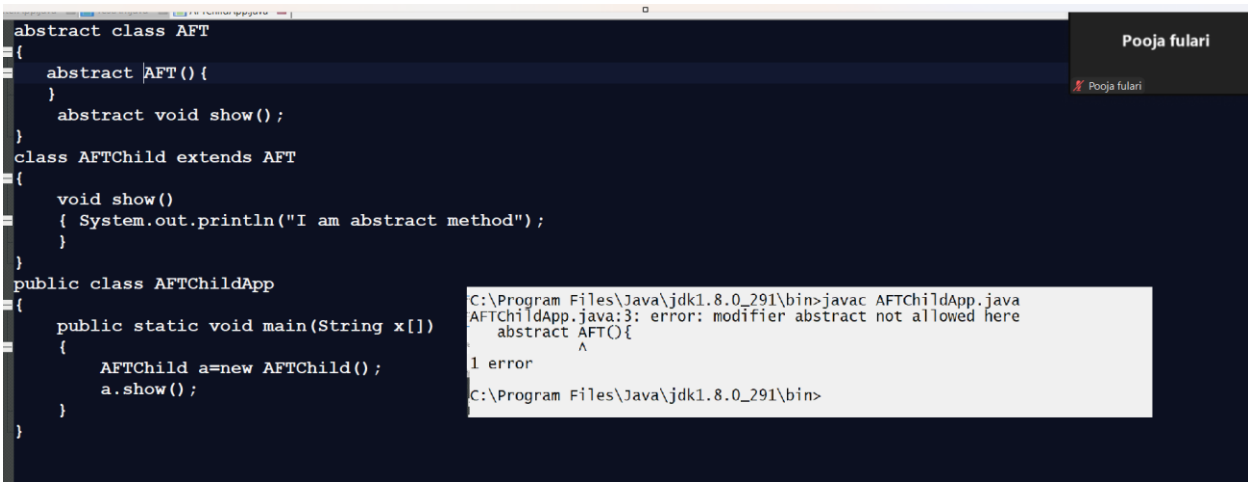


Q12. Can we declare abstract method as static?

No we cannot declare abstract method as static because when we declare abstract method as static then we get compile time error because static method not support to run time polymorphism or dynamic polymorphism as well as overriding and abstract method specially design achieve dynamic polymorphism with the help of method overriding means here we can static and abstract has opposite working principal so we cannot declare abstract method static.

Q13. Can we declare constructor as abstract?

No we cannot declare constructor as abstract because if we try to declare constructor as abstract then we get compile time error because abstract method specially design for method overriding purpose but constructor cannot override so declaring constructor as abstract is useless so java not allow to declare constructor as abstract.

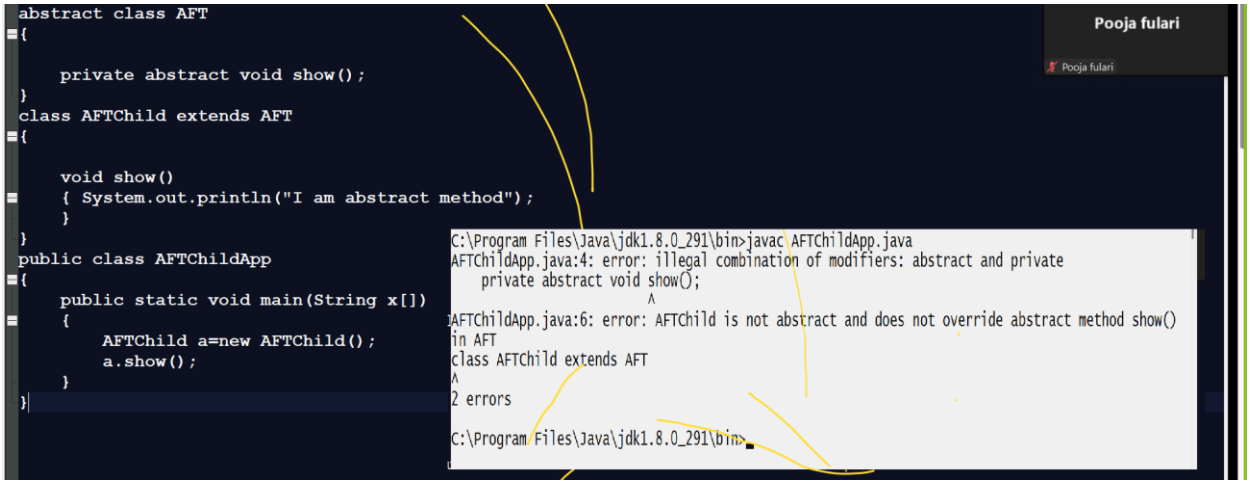


Q14. Can we override constructor?

No we cannot override constructor because when we override method then child method name or signature must be match parent method signature but if we try to override constructor then there is no chance match parent constructor and child class name same because constructor name and class name must be same so parent constructor name and child class name cannot be same so overriding is constructor is meaningless this is major reason we cannot override constructor as well as if constructor not inherit in child class then cannot override also because overriding cannot work without inheritance

Q15. Can we declare abstract method as private?

Not we cannot declare abstract method as private because private method not support to inheritance and overriding not work without inheritance so we cannot declare abstract method as private.



Q16. is it true abstract class cannot achieve 100% abstraction?

---

yes we cannot achieve 100% abstraction by using abstract class because it contain some non abstract methods as well as by default every method is not abstract so this is major reason we cannot achieve 100% abstraction using abstract class.

Q17. What is difference between abstraction and encapsulation?

Abstraction	**Encapsulation**
Abstraction means hide implementation detail at design level	Encapsulation means hide implementation detail at implementation level
Purpose of abstraction is achieve dynamic polymorphism	Purpose of encapsulation is achieve security or protected to user data
Abstraction can achieve with the help of abstract keyword in java	Encapsulation can achieve with the help of private access specified means declaring variable as private access via public function
Abstraction means know what to do but don't know how to do it and implementation dependent in future requirement	Encapsulation means but know what to do as well as how to do it by using security

Q18. How we can achieve dynamic polymorphism using abstract class?

You can achieve dynamic polymorphism using abstract class with the help of up casting technique.

Q20. What is co-variant return type?

Covariant return type means return type of an overriding method. It allow to narrow down return type of overriding method without need to cast the type or check the return type.

Note: covariant return type normally work with non-primitive data type means work with referential data type.

```
abstract class Value

{  int a,b;

  void setValue(int x,int y)

  {  a=x;
```

```
    b=y;
  }
  abstract int getResult();
  Value getValue(){
    return this;
  }
}
class Add extends Value
{
   Add getValue(){
      return this;
    }
  int getResult(){
            return a+b;
  }
}
class Mul extends Value
{
   Mul getValue(){
      return this;
    }
   int getResult(){
            return a*b;
  }
}
public class CoVarientReturnTypeApp
{  static  Mul m = new Mul();
  static Add ad = new Add();
```

```
    public static void main(String x[])
        {    System.out.println("Hashcode of Mul  "+System.identityHashCode(m));


                Value v = m.getValue(); //v= new Mul();
                System.out.println("Hashcode of v "+System.identityHashCode(v));
                v.setValue(10,20);
                int result=v.getResult();
                System.out.println("Mul is  "+result);
                 System.out.println("Hashcode of Add  "+System.identityHashCode(ad));
                v=ad.getValue();
                 System.out.println("Hashcode of V "+System.identityHashCode(v));
                v.setValue(10,20);
                result=v.getResult();
                System.out.println("Addition is  "+result);
        }
}
```

Q19. What is Factory pattern or what is Factory method in Java?

Factory method is design pattern or creational design pattern used in software development. It provides an interface for creating object in super class while allowing sub classes to specify the types of object they state

1. Factory pattern or factory method simplifies the object creation process from dedicated method by using loose coupling technique

2. This method provides flexibility, extensibility and maintainability to sub classes to implement their own factory methods for creating specific object types

Q. When to use factory pattern or factory methods?

---

a) If your object creation technique is complex or varies under the different conditions using a factory method can make your client code simpler

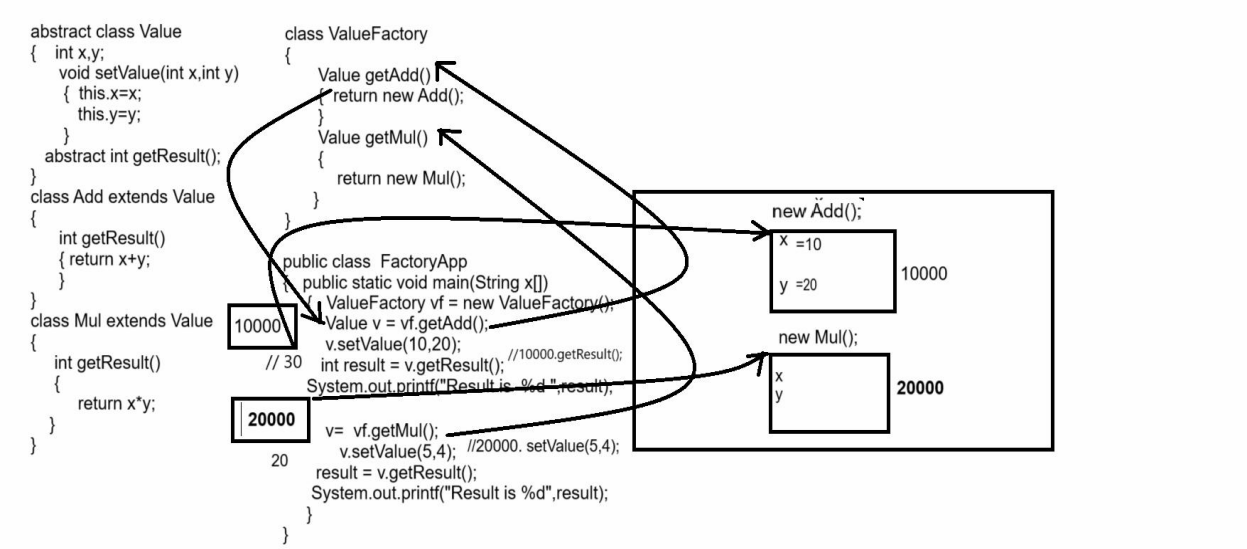
b) Factory pattern allow to create objects using abstract or interfaces using upcasting technique

c) If you need to different kinds of objects in different situation then we can use factory pattern.

Syntax: classname methodname();

Note: if we think about factory method then its return type should be interface name or class name

Example of factory pattern



Example with source code

---

abstract class Value

{ int x,y;

void setValue(int x,int y)

{ this.x=x;

this.y=y;

}

abstract int getResult();

}

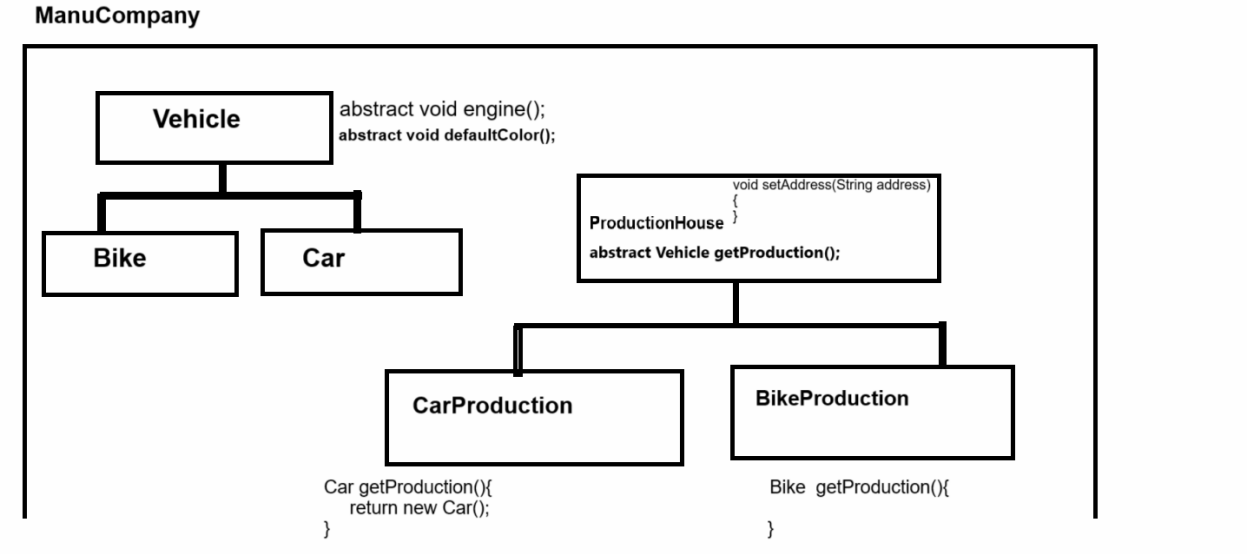
class Add extends Value

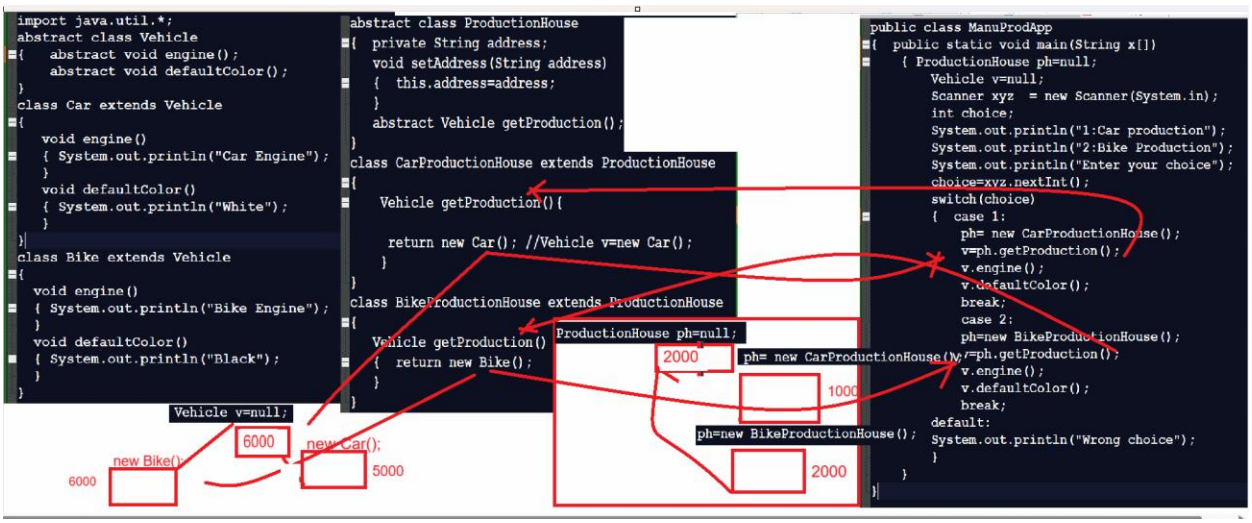
```
{
  int getResult(){
    return x+y;
  }
}
class Mul extends Value
{
  int getResult()
  {
    return x*y;
  }
}
class ValueFactory
{
    Value getAdd(){
            return new Add();
           }
          Value getMul(){
            return new Mul();
          }
}
public class FactoryApp
{
   public static void main(String x[])
         {
                ValueFactory vf = new ValueFactory();
                Value v =vf.getAdd();
                v.setValue(10,20);
```

```
            int result=v.getResult();
            System.out.printf("Addition is %d\n",result);
            v=vf.getMul();
            v.setValue(5,4);
            result=v.getResult();
            System.out.printf("Multiplication is %d\n",result);
        }
}
```

Example:

Suppose consider we are working for Manufacturing company and we think company produce vehicles and company produces two types of Vehicle i.e Car and Bike but here we want to achieve abstraction so we think Car is vehicle and Bike is also Vehicle and both vehicle required engine() and some default color we design one Vehicle which is abstract and declare engine() and defaultColor() as abstract and mark Vehicle is parent to Car and Bike and we consider Manu Company has two productions name as CarProduction and BikeProduction means production houses produces Vehicle but dependent on production house here we declare on abstract class name as ProductionHouse which contain non-abstract method name as void setAddress(String address) because this method logic is common both production houses but we design one method as abstract name as Vehicle getProduction() and this method return reference of Vehicle so we want to override this method in CarProduction and BikeProduction class and return object of Vehicle according to production type so here we can specify return type of method as Vehicle in child classes or we can specify return as specific child class according to co-variant return type shown in following diagram.





Example with source code

```
import java.util.*;
abstract class Vehicle
{   abstract void engine();
            abstract void defaultColor();
}
class Car extends Vehicle
{  void engine()
   { System.out.println("Car Engine");
   }
   void defaultColor()
   { System.out.println("White");
   }
}
class Bike extends Vehicle
{ void engine()
  { System.out.println("Bike Engine");
```

```
  }
  void defaultColor()
  { System.out.println("Black");
  }
}
abstract class ProductionHouse
{  private String address;
   void setAddress(String address)
   {  this.address=address;
   }
   abstract Vehicle getProduction();
}
class CarProductionHouse extends ProductionHouse
{   Car getProduction(){
      return new Car(); //Vehicle v=new Car();
            }
}
class BikeProductionHouse extends ProductionHouse
{  Bike getProduction()
   {  return new Bike();
   }
}
public class ManuProdApp
{  public static void main(String x[])
            { ProductionHouse ph=null;
                        Vehicle v=null;
                        Scanner xyz  = new Scanner(System.in);
                        int choice;
```

```
                System.out.println("1:Car production");
                System.out.println("2:Bike Production");
                System.out.println("Enter your choice");
                choice=xyz.nextInt();
                switch(choice)
                {  case 1:
                   ph= new CarProductionHouse();
                        v=ph.getProduction();
                        v.engine();
                        v.defaultColor();
                        break;
                        case 2:
                        ph=new BikeProductionHouse();
                        v=ph.getProduction();
                        v.engine();
                        v.defaultColor();
                        break;
                default:
                System.out.println("Wrong choice");
                }
        }
}
```

Interface

Q. What is interface?

Interface is a same like as abstract class in java or it is internally abstract class

Q. Why use interface if we have abstract class already in java?

Reasons to use interface

---

1. Achieve 100% abstraction

2. To Achieve dynamic polymorphism

3. To achieve multiple inheritance.

How to declare interface in JAVA

---

if we want to declare interface in java we have to use interface keyword

Syntax:

interface interfacename{

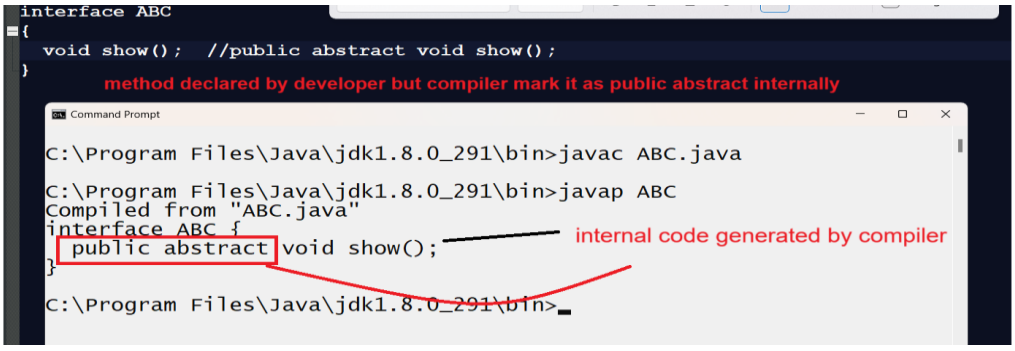
returntype functionname(arguments);

}

```
interface ABC
{
  void show();
}
```

How we can say interface achieve 100% abstraction?

Because when we declare any method within interface then method is by default abstract and using abstract method we can achieve abstraction so this behavior of interface shows we can achieve 100% abstraction using interface.



If we want to write logic of abstract method or interface method we need to implement in another class and override its method and write method logic in implementer class

Syntax to implement interface in another class

---

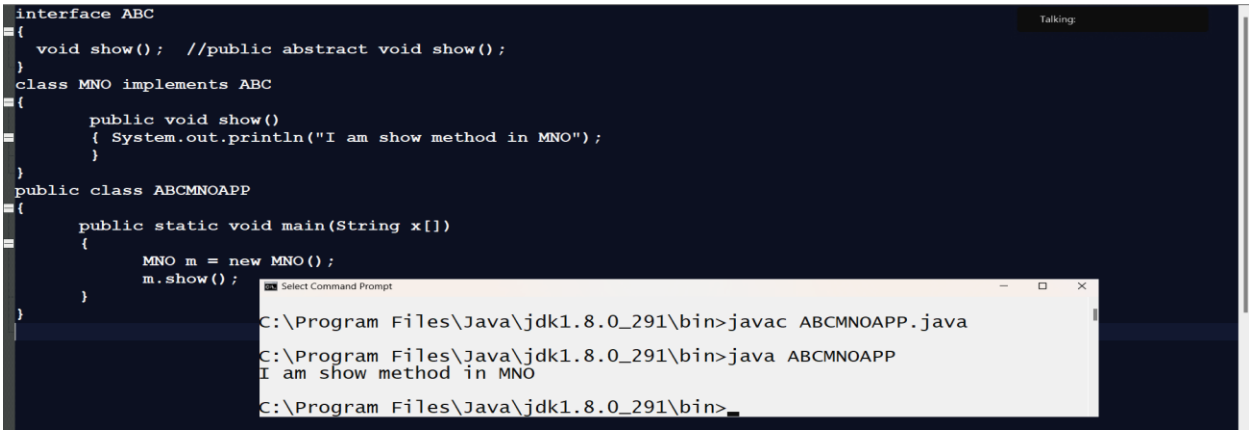
class classname implements interfacename{

public void methodname(datatype variables){

}

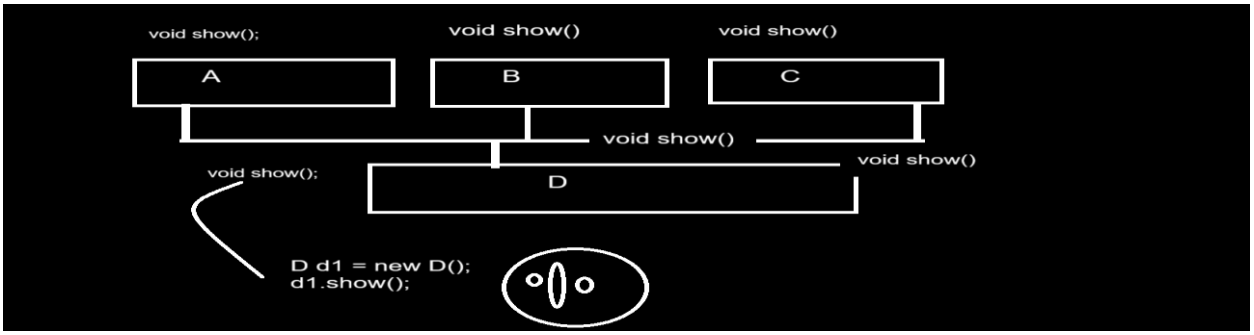
}

Note: when we implement or override interface method it must be override public because internally it is public and we cannot override public to default so we must be write public at the time of overriding.



Q. Why JAVA not implement multiple inheritances by using classes?

Because of diamond problem here we can say diamond problem means if we have more than one parent classes and single child class and if there is possibility if multiple parent classes contain same name method then in child class method may be duplicated from different parent classes and if we create object of child class and try to call method whose name same in different parent then your compiler may be get confused called as diamond problem and if we want to solve this problem we have to use interface.



Note: if we think about above diagram we have three parent classes name as A , B and C and every parent class contain one common method name as void show() and means show() method three version present in class D one is from A, one is From B and one is from C and if we try to create object of class D and try to call show() method then compiler may be get confused because compiler cannot predicate which parent version should executed and if we want to solve this problem we have to use interface

If we want to work with interface we have some important points.

---

1. Interface is by default internally abstract class means we cannot create object of interface.

2. Interface methods are internally by default public abstract

3. If we want to use interface we must be implement in another implementer class.

4. If we declare any variable within interface it is by default public static final

Important related interface variables

---

a) When we declare variable within interface it must have some initialize otherwise compiler generate error to us

```
interface ABC
{
  void show();  //public abstract void show();
  float PI;
}
```

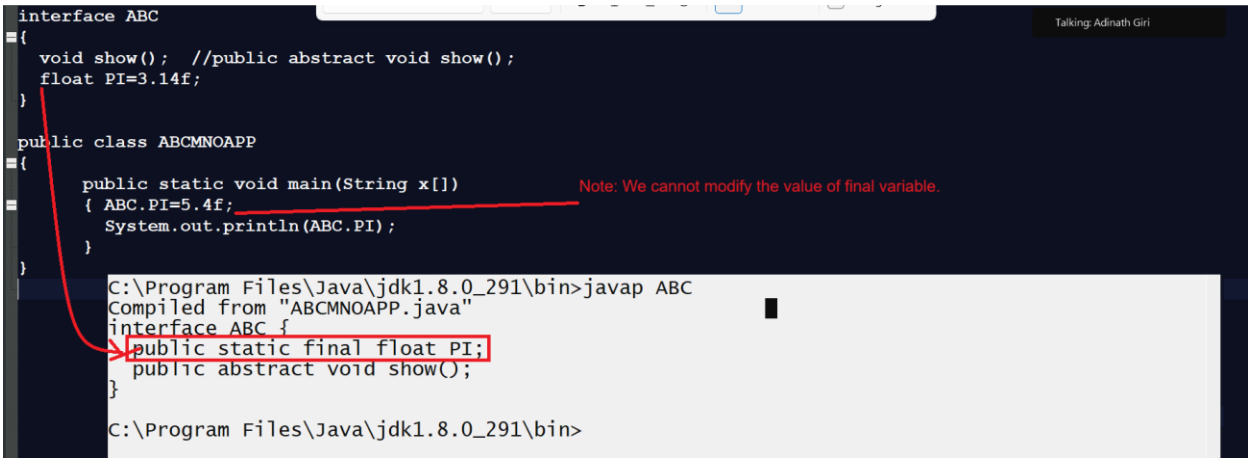
```
C:\Program Files\Java\jdk1.8.0_291\bin>javac ABCMNOAPP.java
ABCMNOAPP.java:4: error: = expected
  float PI;
          ^
1 error
```

b) We can access interface variable by using interface because they are static and static can using class name and interface is same as class so it is possible.

```
interface ABC
{
  void show();  //public abstract void show();
  float PI=3.14f;
}

public class ABCMNOAPP
{
      public static void main(String x[])
      {
        System.out.println(ABC.PI);
      }
}
```

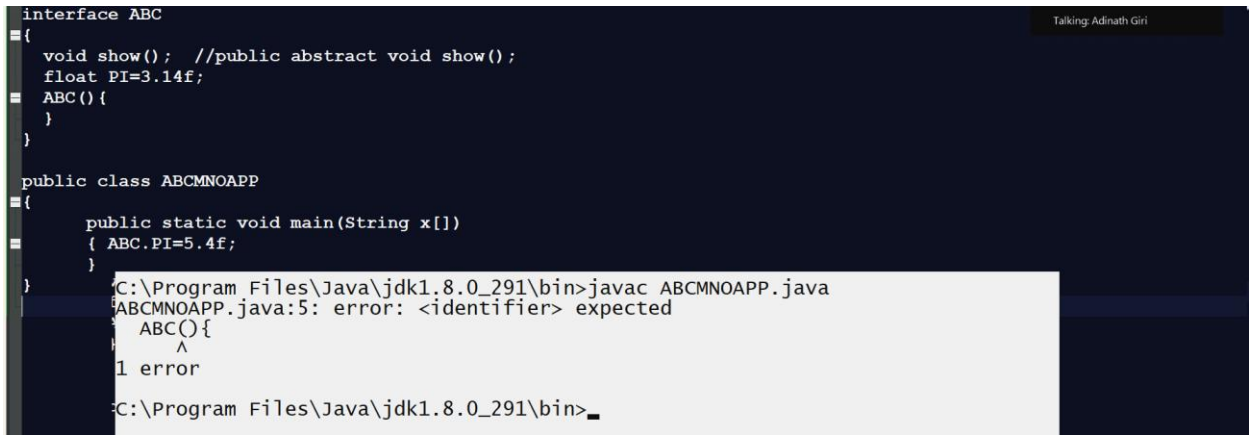
```
Select Command Prompt

C:\Program Files\Java\jdk1.8.0_291\bin>javac ABCMNOAPP.java

C:\Program Files\Java\jdk1.8.0_291\bin>java ABCMNOAPP
3.14

C:\Program Files\Java\jdk1.8.0_291\bin>
```

C) We cannot modify the value of interface variable because they public static and final



5. Can we declare constructor within interface?

No we cannot declare constructor within interface because when we try to declare constructor within interface then we get compile time error.

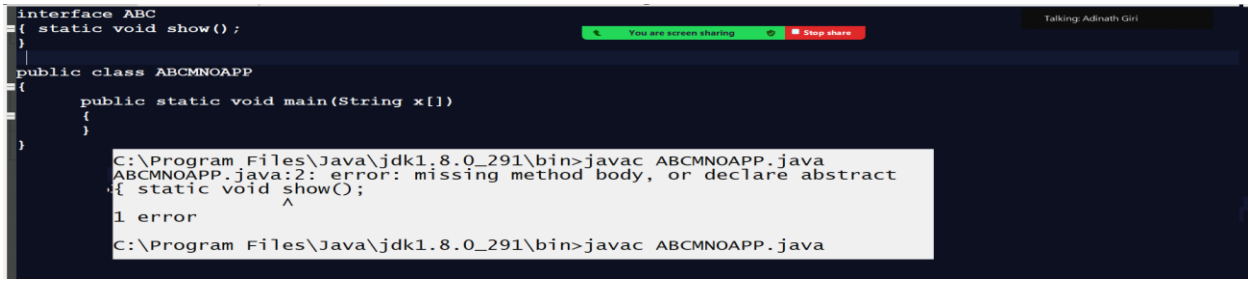


Q. If we can declare constructor within abstract class and interface is internally like as abstract class so why we cannot declare constructor within interface?

Because the goal of constructor is initialize the value to class member but if we think about interface when we declare variable within interface it must be initialize some value and interface variables are by default final so we cannot modify it later means if we try to initialize them using constructor so it is meaningless so java not allowed constructor in interface even interface is same like as abstract class

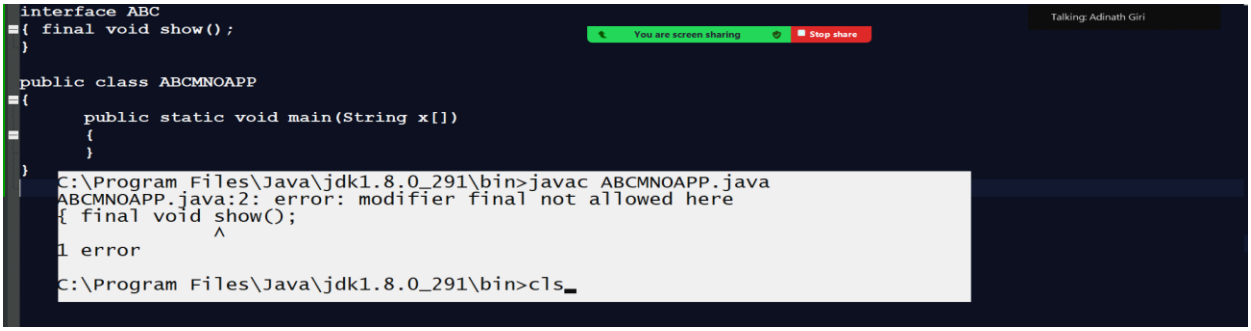
Q. Can we declare interface method as static?

No we cannot declare interface method as static because interface methods are internally public abstract and abstract method cannot declare as static so interface method cannot declare as static.



Q. Can we declare interface method as final?

No we cannot declare interface method as final because final method cannot override and interface method is internally abstract and it must be override in another class so we cannot override it.



Q. Can we declare interface method as private?

No we cannot declare interface method as private because private method cannot support to inheritance and overriding and interface method must be override so we cannot declare interface method as private.

Q. Can we declare interface method as protected?

No we cannot declare interface method as protected and if we try to declare it as protected then we get compile time error.

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac ABCMNOAPP.java
ABCMNOAPP.java:2: error: modifier protected not allowed here
{ protected  void show();
             ^
1 error

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Q. Can we declare abstract method in abstract as protected?

Yes we can declare abstract method in abstract class as protected and can override it in child class shown in following code.

Q. If abstract method in abstract class can declare as protected and if interface method is internally abstract so why we cannot declare it as protected?

---

because when we declare method within interface it is internally public abstract and when we declare it as protected then internally it is consider as protected public abstract in java we cannot use two access specifier at same time or there is not combination of protected public allow at same so we cannot declare it as protected and protected has less accessing priority so protected cannot override on public so it is a consider as protected public with method and when we declare abstract method in abstract class then it works with default access specifier and when we use protected with abstract method in abstract class then protected get override on default so it is possible shown in following diagram.

illegal combination of access specifier with method

```
interface ABC
{
   protected void show();  //protected  public  abstract  void show();
}

abstract class ABC
{
  protected  abstract void show(); //protected  abstract void show()
}
```

Q. Can we inherit interface to interface?

Yes we can inherit interface to interface and if we want to inherit interface to interface we have to use extends keyword.

```
interface A
{ void show();
}
interface B extends A
{
  void display();
}
class C implements B
{
   public void show()
   { System.out.println("I am method of A interface");
   }
   public void display()
   {
   }
}
public class InfApp
{
    public static void main(String x[])
    {
        C c1  = new C();
        c1.show();
    }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InfApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InfApp
I am method of A interface

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Q. Is it mandatory to override all methods of interface where interface implements?

Yes it is mandatory to override all methods of interface where interface get implemented because all methods of interface are internally public and abstract and interface is internally consider as abstract class so when we have more than one abstract method in parent then all method must be override where interfaces implements and if we not required then need to define blank otherwise you can use adapter class for solve above problem.

Example: Solution without adapter class

```
interface B
{
  void display();
  void show();
}
class C implements B
{
   public void show()
   { System.out.println("I am method of B interface");
   }
   public void display()
   {
   }
}
public class InfApp
{
    public static void main(String x[])
    {
        C c1  = new C();
        c1.show();
    }
}
```

Talking: Adinath Giri

Note: Left hand side code indicate solution without adapter class.

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InfApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InfApp
I am method of B interface

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Example: Solution with adapter class

```
interface B
{
  void display();
  void show();
}
class ADP implements B{
      public void show(){
      }
      public void display(){
      }
}
class C extends ADP
{  public void show()
   { System.out.println("I am method of B interface");
   }
}
public class InfApp
{   public static void main(String x[])
    {  C c1  = new C();
        c1.show();
    }
}
```

Talking:

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InfApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InfApp
I am method of B interface

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Q. Can we create reference of interface?

Yes we can create reference of interface using up casting technique means we have to create object of its implementer class and store its address in interface reference.

Syntax:

interfacename ref = new implementerclassname();

Example: Value v = new Add();

Here we consider Value v is interface and new Add() is implementer class of Value.

The goal of interface is achieve loose coupling and dynamic polymorphism show in following code

```
interface Value
{ void setValue(int x,int y);
  int getResult();
}
class Add implements Value
{  int a,b;
   public void setValue(int x,int y)
   { a=x;
     b=y;
   }
   public int getResult(){
            return a+b;
   }
}
class Mul implements Value
{  int a,b;
   public void setValue(int x,int y)
   { a=x;
     b=y;
   }
   public int getResult(){
            return a*b;
   }
}
class Calc
{
  void performOperation(Value v)
```

```
 {
    int result = v.getResult();
            System.out.println("Result is  "+result);
 }
}
public class LooseCouplingInfApp
{   public static void main(String x[])
   {   Calc c = new Calc();
                       Value v = new Add();
                        v.setValue(10,20);
                        c.performOperation(v);
                        v=new Mul();
                        v.setValue(5,90);
                        c.performOperation(v);

           }
}
```

Q. Can we use the interface without implements in another class?

Yes we can use interface without implement in another class by using anonymous inner class concept.

Q. What is anonymous inner class?

Anonymous inner class is one type of local inner class where we create object of abstract class or interface and override its abstract method can use it without implementation or inheritance

Syntax: interfacename ref = new interfacename(){

```
  //override methods
};
```

or

```
abstractclassname ref = new abstractclassname(){
  // override here all methods of abstract class
};
```

Example anonymous inner class using interface.

```
interface Value
{ void setValue(int x,int y);
 int getResult();
}
public class LooseCouplingInfApp
{   public static void main(String x[])
   {   Value v = new Value(){
                              int a,b;
                        public void setValue(int x,int y)
        {  //overriding
                                 a=x;
                                       b=y;
        }
        public int getResult(){
                    return a+b;
                              }
                    };
                     v.setValue(10,20);
                     int result=v.getResult();
                     System.out.println("Addition is  "+result);
          }
}
```

Example anonymous inner class using abstract class

```
abstract class Value
{ abstract void setValue(int x,int y);
  abstract  int getResult();
}
public class LooseCouplingInfApp
{   public static void main(String x[])
    {   Value v = new Value(){
                                int a,b;
                        public void setValue(int x,int y)
          {  //overriding
                                    a=x;
                                          b=y;
          }
          public int getResult(){
                      return a+b;
                                }
                      };
                       v.setValue(10,20);
                       int result=v.getResult();
                       System.out.println("Addition is  "+result);
           }
}
```

Q. Can we declare class within interface?

Yes we can declare within interface or nested class in interface and you can use interface using a following ways

```
interface A
{  void show();
   class B implements A
   {
       public void show(){
         System.out.println("I am show method of A");
       }
   }
}
class C extends A.B
{

}
public class InfClassApp
{
    public static void main(String x[])
    {  C c1 = new C();
        c1.show();
    }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InfClassApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InfClassApp
I am show method of A

C:\Program Files\Java\jdk1.8.0_291\bin>
```

or

```
interface A
{  void show();
   class B
   {

   }
}
class C extends A.B implements A
{
      public void show(){
           System.out.println("I am show method");
      }

}
public class InfClassApp
{
    public static void main(String x[])
    {  C c1 = new C();
        c1.show();
    }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InfClassApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InfClassApp
I am show method

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Q. Can we declare interface within class?

Yes we can declare interface within class

```
class A
{
   interface B
   {
        void show();
   }
}
class C implements A.B
{
      public void show(){
           System.out.println("I am show method");
      }

}
public class InfClassApp
{
    public static void main(String x[])
    {  C c1 = new C();
        c1.show();
    }
}
```

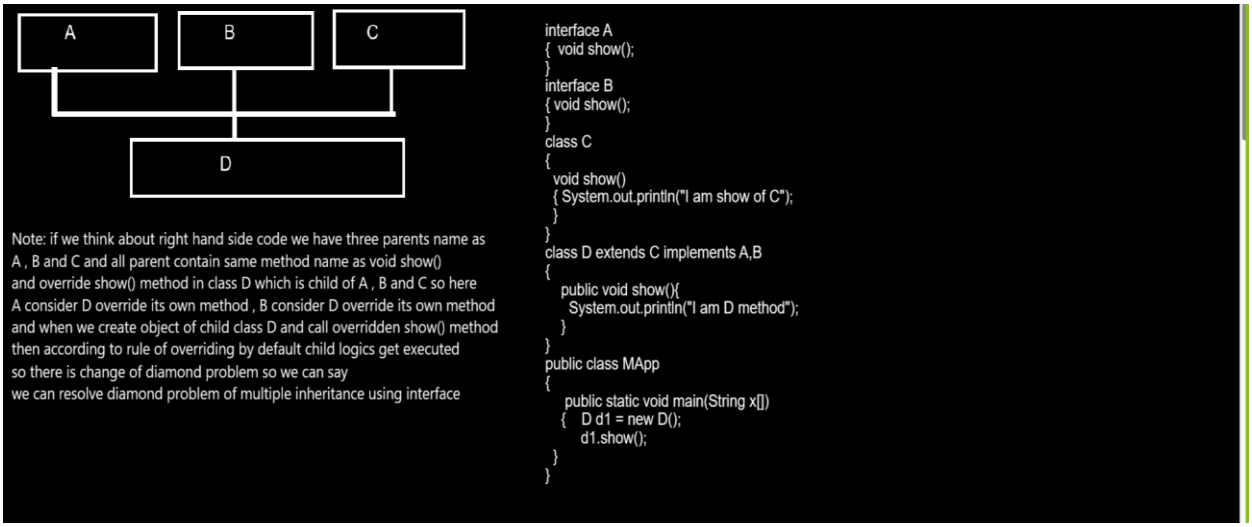
```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InfClassApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InfClassApp
I am show method

C:\Program Files\Java\jdk1.8.0_291\bin>
```

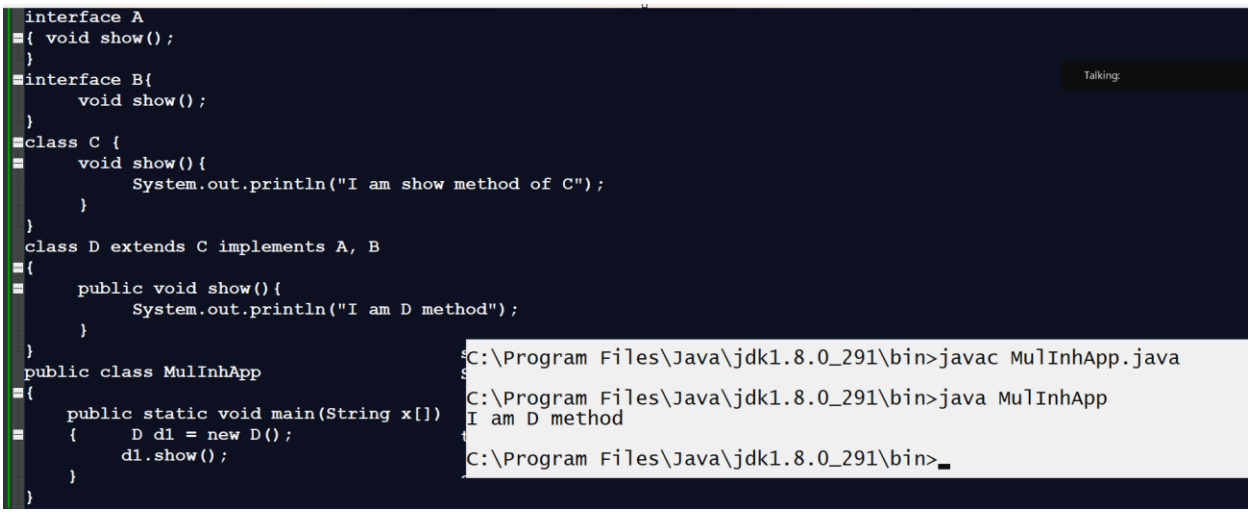
How to achieve multiple inheritances using interface and how avoid diamond problem

---

Q. What is multiple inheritances?

Multiple inheritances mean more than one parents and single child class called as multiple inheritances and multiple parents there may be only one parent class and remaining all are interfaces or all parents are interfaces.





Q. What is difference between interface and abstract class?

Abstract class	**Interface**
Abstract class need to use abstract keyword for mark abstract	Interface not required to use abstract keyword because internally it is abstract by default
Abstract class contain non final variables and may be final variable	Interface variables are by default final
Abstract class can have constructor	Interface cannot have constructor
Abstract class contain non abstract method	Interface cannot contain non abstract method and if we want to define then need to use default or static
Abstract class need to extends in child class	Interface can implements in child class
Abstract class methods are by default use default access specifier	Interface methods by default use the public access specifier
Abstract class can declare abstract method as protected	Interface cannot declare abstract method as protected
Abstract class cannot support to multiple inheritance	Interface can support to multiple inheritance
Abstract class not achieve 100% abstraction	Interface can achieve 100% abstraction
Abstract class not support to lambda expression	Interface can support to lambda expression