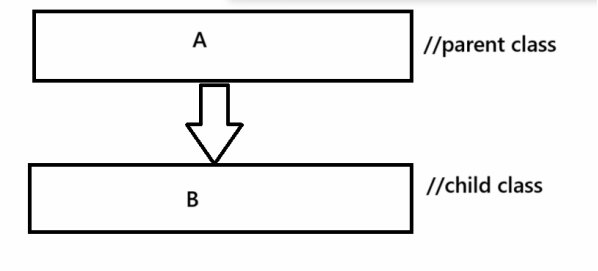
Inheritance

Q. What is inheritance?

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

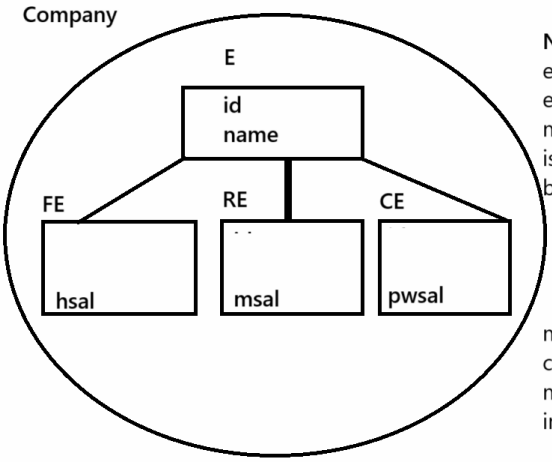


Q. Why we need to use inheritance or what is benefit of inheritance?

---

1. Reusability of code: means we can reuse class without creating its object called as reusability.
Means we can say we can reuse class with the help of inheritance

Suppose consider we have example

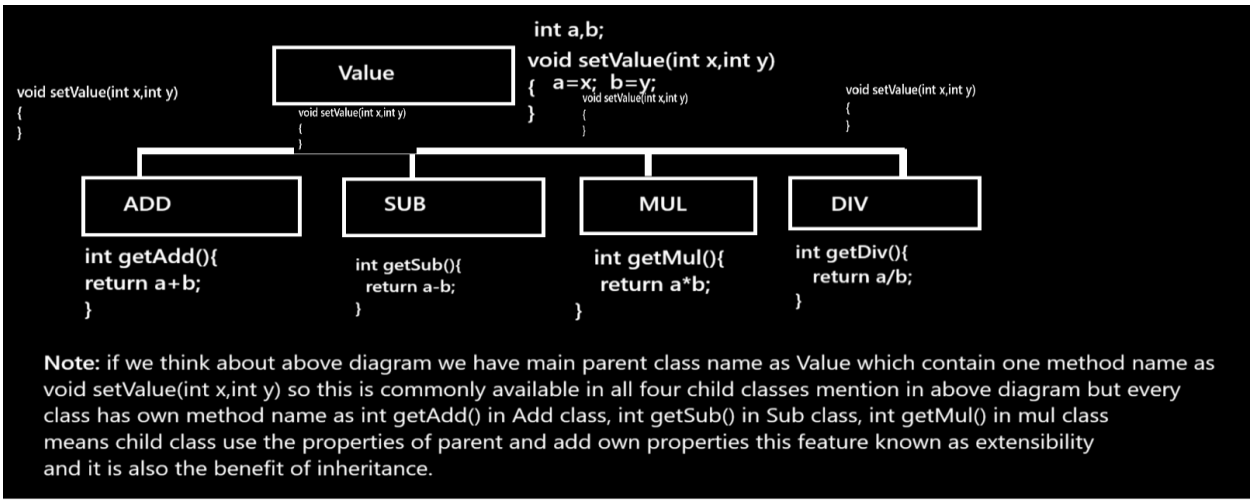


Note: if we think about this diagram we have every employee must have id and name means if your employee category is free lancer must have id and name may be salary structure is hourly if your employee category is Regular employee then it must have id and name may be salary structure is monthly salary etc means if we have some common logic required in multiple classes so creating object of class every time is not good approach so better way you can inherit the class where we use it called as reusability means id and name is common in all three classes which is child of E class mention in diagram means we not need to redeclare variables again in different child classes.

2. Extensibility of code

Extensibility means child class can acquire the properties from parent class and add own properties in it called as extensibility or child class can modify parent logic using overriding according to own requirement called as extensibility.

Example: Suppose consider we want to design calculator using inheritance and we want to perform four operations in Calculator like as Add, Mul, Div,Sub and if we think about these operation every operation required two values but every operation has different operation like as Add has different logic,Mul has different logic so your code structure is like as



Note: We cannot create any program in java without inheritance.
Because in java every class has default parent class known as Object class

Q. Why JAVA provide Object as parent to every class or why Object is parent of every class?

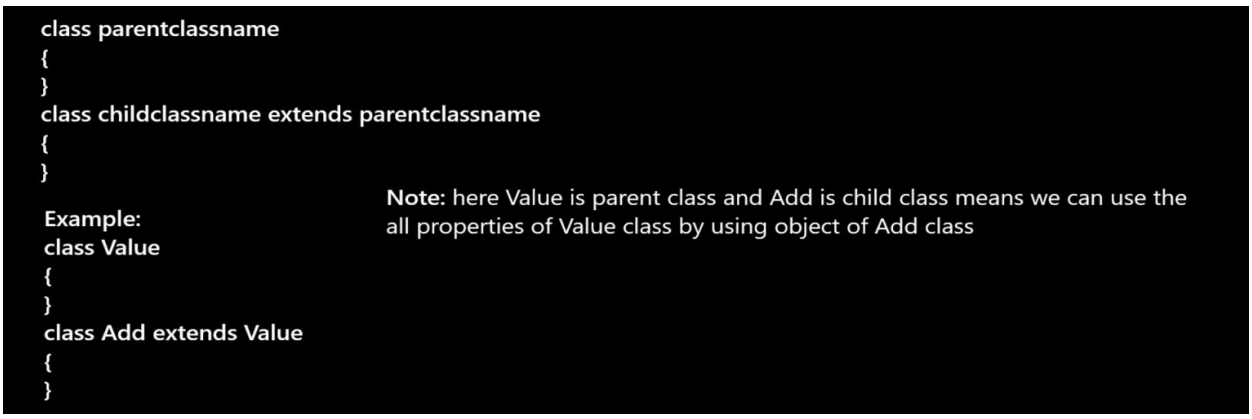
Because Object class contain some inbuilt methods in class required for daily operations

Methods of Object class

---

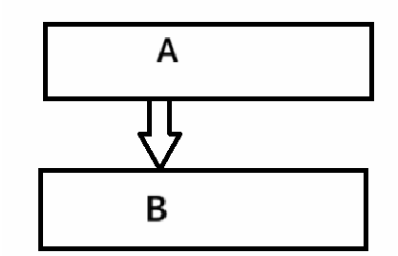
int hashCode(), boolean equals(), String toString(), Object clone(), Class getClass(), void notify(), void notifyAll (), void wait (int),void wait(),void finalize() ,static{ }

If you want to perform inheritance in java we have to use extends keyword

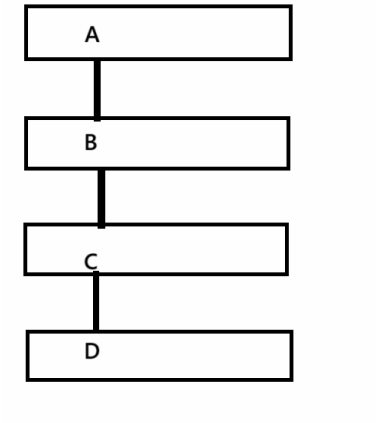


Types of inheritance

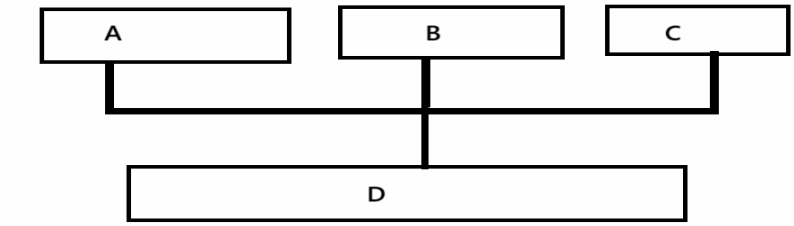
1. Single level inheritance: single level means there is single parent and single child class called as Single level inheritance.



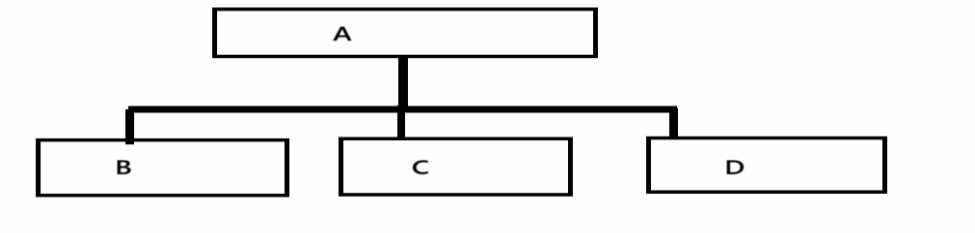
2. Multilevel inheritance: Multilevel inheritance means one class is parent of another and child of another called as multi level inheritance.



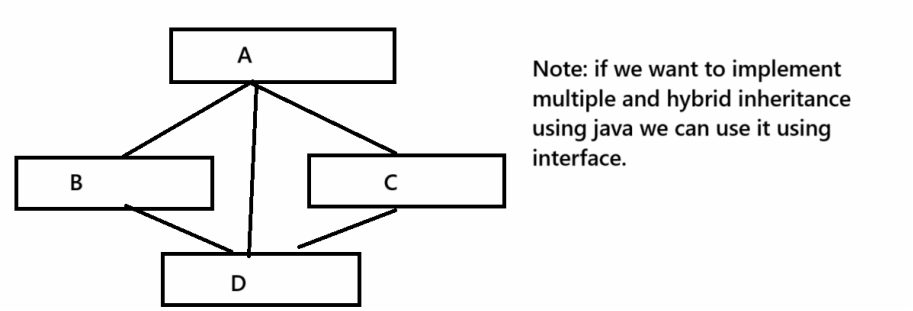
3. Multiple inheritances: Multiple inheritance means there is more than one parents and single child class called as multiple inheritances.



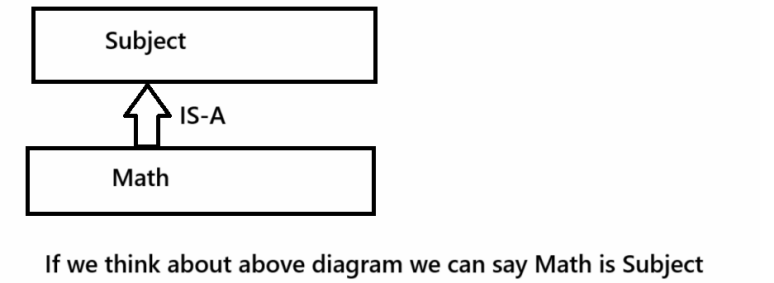
4. Hierarchical inheritance: Hierarchical inheritance means there is single parent and more than one child classes called as hierarchical inheritance.



5. Hybrid inheritance: Hybrid inheritance is combination of all types of inheritance



IS-A relation: When we implement the inheritance between two classes then classes bind in new relationship called as IS-A relationship.

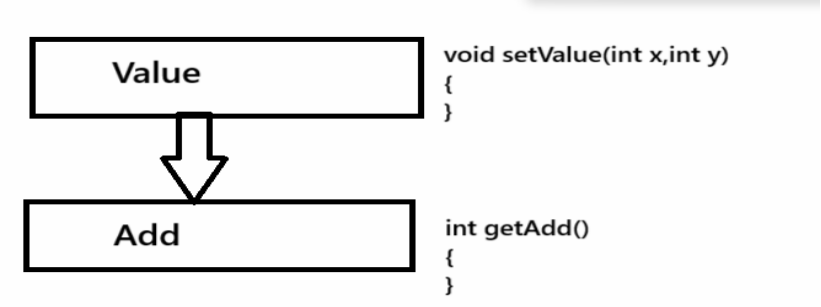


HAS-A relation: when we create object of one class in another class called as HAS-A relationship

```
class Chapter{
}
class Ratio extends Chapter
{
}
class Subject
{
}            //IS-A
class Math extends Subject
{
    Ratio r = new Ratio();

}
```

If we think about above code we can say Math is Subject & Math Has-A ration and Ratio is chapter
If we think about inheritance we not need to create object of parent class we can access all member of parent object using child class object.

Single level inheritance

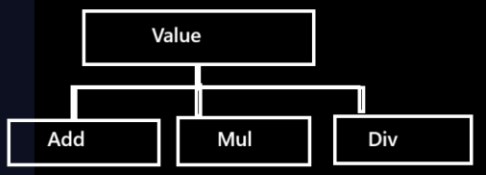


```
class Value
{
   int a,b;
   void setValue(int x,int y)
   {  a=x;
      b=y;
   }
}
class Add extends Value
{
   int getAdd()
   { return a+b;
   }
}
public class SingleLevelApp
{
   public static void main(String x[])
   {
       Add ad = new Add();
       ad.setValue(10,20);
       int result =ad.getAdd();
       System.out.printf("Addition is %d\n",result);
   }
}
```

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

C:\Program Files\Java\jdk-23\bin>java SingleLevelApp
Addition is 30
```

```
class Value
{
   int a,b;
   void setValue(int x,int y)
   {  a=x;
      b=y;
   }
}
class Add extends Value
{
   int getAdd()
   { return a+b;
   }
}
class Mul extends Value
{
       int getMul()
       { return a*b;
       }
}
class Div extends Value
{    int getDiv()
     {
          return a/b;
     }
}
```

```
public class HierarchicalApp
{
   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("Multiplication is %d\n",result);

       Div d = new Div();
       d.setValue(20,5);
       result=d.getDiv();
       System.out.printf("Division is %d\n",result);
   }
}
```



Multi level inheritance

```
class SportMarks
{    int sportMarks;
     void setSportMarks(int sportMarks)
     {  this.sportMarks=sportMarks;
     }
}
class AcadamicMarks extends SportMarks
{     int marks[];
      void setAcadamicMarks(int marks[])
      {     this.marks=marks;
      }
}
class Result extends AcadamicMarks
{   int agg=0;
      int getResult()
```

```
    {  for(int i=0;i<marks.length; i++)
       {  agg=agg+marks[i];
       }
      agg=agg+sportMarks;
   return agg/7;
    }
}

public class MultilevelApp
{
  public static void main(String x[])
  {
    Result r = new Result();
          int a[]=new int[]{60,60,60,60,60,60};
           r.setAcadamicMarks(a);
                r.setSportMarks(60);
          int result=r.getResult();
          System.out.println ("Result is “+result);
  }
}
```

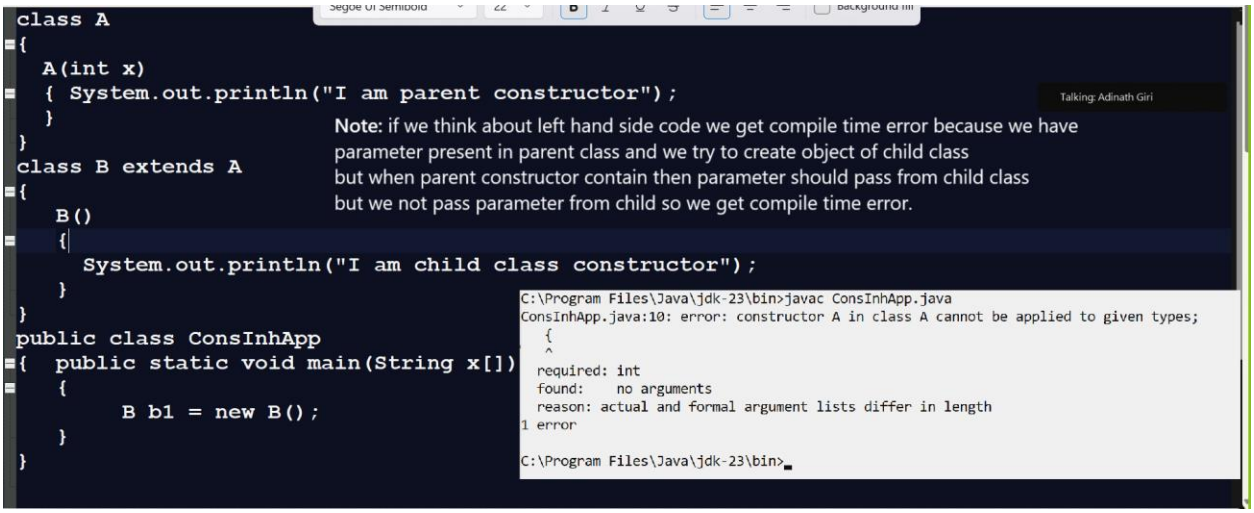
Constructor in inheritance

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

```
class A
{
  A()
  { System.out.println("I am parent constructor");
  }
}
class B extends A
{
   B()
   {
     System.out.println("I am child class constructor");
   }
}
public class ConsInhApp
{  public static void main(String x[])
   {
        B b1 = new B();
   }
}
```

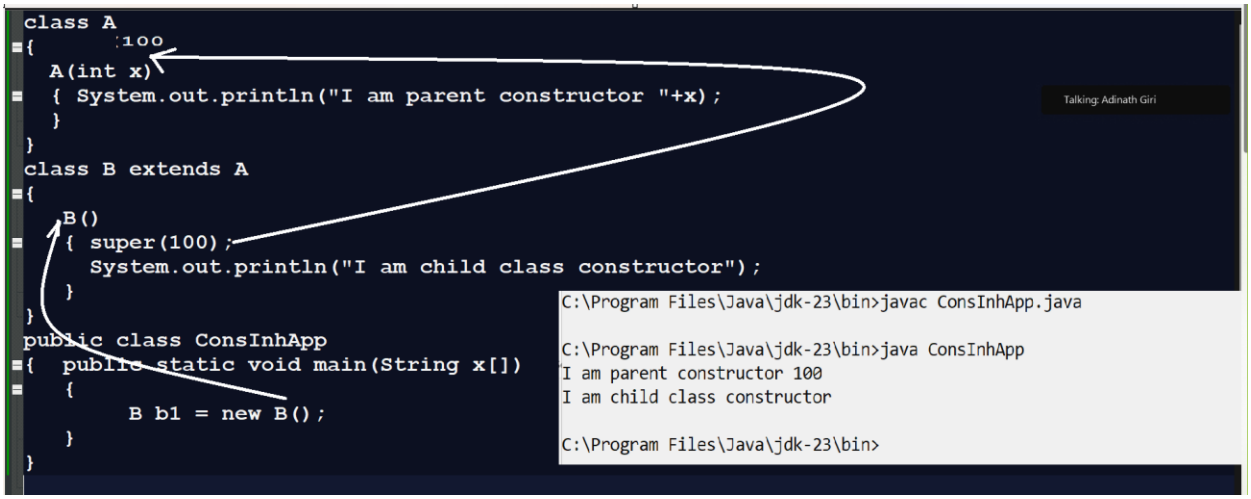
```
C:\Program Files\Java\jdk-23\bin>javac ConsInhApp.java

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

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

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



So if we want to avoid compile time error from above code we need to use super constructor.

Note: super() constructor must be first line of code in child class constructor



Q. What is super constructor?

---

super() is a constructor in java which is used for pass parameter from child class constructor to parent class and it must be first line of code in child class constructor shown in above code.

Q. What is constructor chaining and how we can handle it in inheritance?

Constructor chaining means if we call one constructor from another constructor called as constructor chaining

There are two ways to manage the constructor chaining

a) this() constructor : refer constructor topic

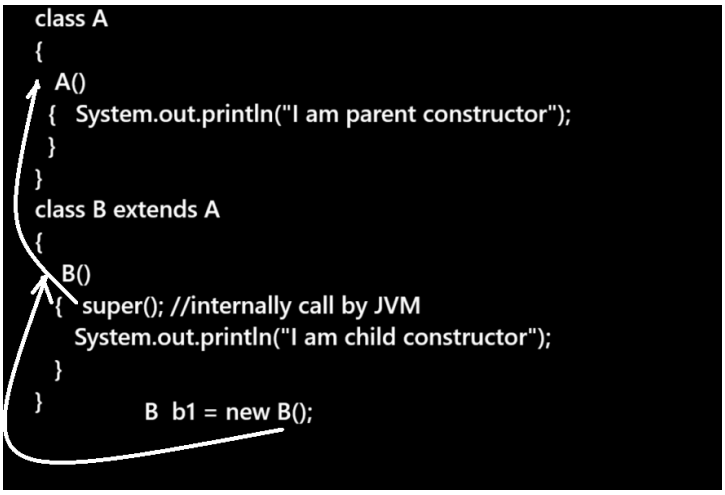
b) super() constructor : super constructor help us to manage or handle the constructor chaining using inheritance.

Q. is it by default super () constructor call by JVM from child class?

or

Q. Why we not need to call super() constructor manually when we have default constructor in parent?

When we have default constructor in parent then by default JVM call the super() constructor from child class internally so parent constructor get executed automatically shown in following code.

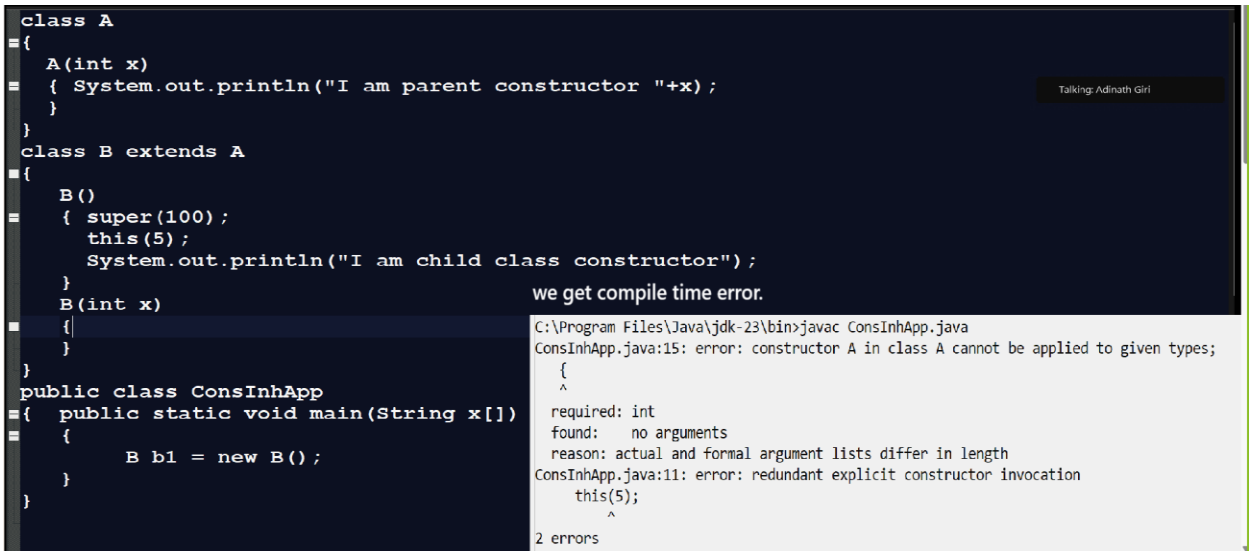


Q. Why we need to call super() constructor manually when we have parameterized constructor in parent?

When we have parameterized constructor in parent class then parameter value is not decided by JVM it is dependent on user so we need to pass parameter value as per the choice of user so we required calling super constructor and passing value to it.

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

No we cannot use this() constructor and super() constructor at same time because this() constructor first line of code and super() constructor also required first line of code so we cannot use this() and super() at same time and if we try to use it then we get compile time error shown in following code.



Q. What is difference between this() and super() constructor?

this() constructor	super() constructor
this() constructor can perform constructor chaining using a same class	super() constructor can perform constructor chaining using inheritance.
this() constructor can perform constructor when we have constructor overloading within same class	Super () constructor can perform constructor chaining when we have constructor present in parent class.

Q. What is similarity between this() and super() constructor?

1. Both are use for constructor chaining
2. this() and super() must be on same line or must be use on same line

final keyword

Q. what is final?

final is non access specifier in java we can use with variable, function and class

Q. What is final variable?

final variable means a variable cannot modify its value once we assign it means we can say final variable is use for declare the constant vale in JAVA.

```
public class FinalVarApp
{
    public static void main(String x[])
    {
        final int a=10;
        ++a;
        System.out.printf("A is %d\n",a);
    }
}
```

Note: if we think 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 to us

```
C:\Program Files\Java\jdk-23\bin>javac FinalVarApp.java
FinalVarApp.java:6: error: cannot assign a value to final variable a
                ++a;
                  ^
1 error

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

Q. What is final class?

Final class means a class cannot inherit in any another class and if we try to inherit the final class in any another class we get compile time error.

Final class normally used for create immutable classes in JAVA.

```
final class A
{
}
class B extends A
{
}
public class FinalApp
{
    public static void main(String x[])
    {
    }
}
```

Note: if we think about left hand side code we get compile time error because we have final class A and we try to inherit it in class B and we cannot inherit final class in any another class so get compile time error

```
C:\Program Files\Java\jdk-23\bin>javac FinalApp.java
FinalApp.java:5: error: cannot inherit from final A
class B extends A
                ^
1 error

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

Note: we can create object of final class but we cannot inherit it.

Q. What is final method?

---

final method means a method cannot override in child class.

Note: before understanding final method we should have to know

Q. What is method overriding?

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

If we think about method overriding return type of method, function name, data type , argument list and sequence of argument must be same.

```
class A
{
  void show()
  {  System.out.println("I am parent method");
  }
}
class B extends A
{
  void show()
  {
    System.out.println("I am child method");
  }
}
```

Note: we override show() method from class A to class B

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

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

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

C:\Program Files\Java\jdk-23\bin>java OverrideApp
I am child show method

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

Note: if we think about above output we can say override always execute the logic of child means we lost the logic of parent using overriding.

Q. What is benefit of overriding or why use overriding?

---

1. To customize parent logic according to child class: means child class can modify the parent logic according to his requirement

2. To achieve dynamic polymorphism:

Note: we will discuss dynamic polymorphism later in this chapter

Q. is method overriding is beneficial?

It is dependent on scenario or situation some time it is beneficial but some time not.

Q. How we can avoid method overriding or what is goal of final method?

---

if we want to avoid method overriding then we can define parent method as final and when we define parent method as final then we cannot override it in child class means using final we can avoid method overriding or we cannot restrict child class to modify parent logic by using method overriding technique it is goal of final method or it is technique of avoid method overriding.

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

```
Command Prompt

C:\Program Files\Java\jdk-23\bin>javac AvoidOverrideApp.java
AvoidOverrideApp.java:9: error: show() in B cannot override show() in A
   void show()
        ^
  overridden method is final
1 error

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

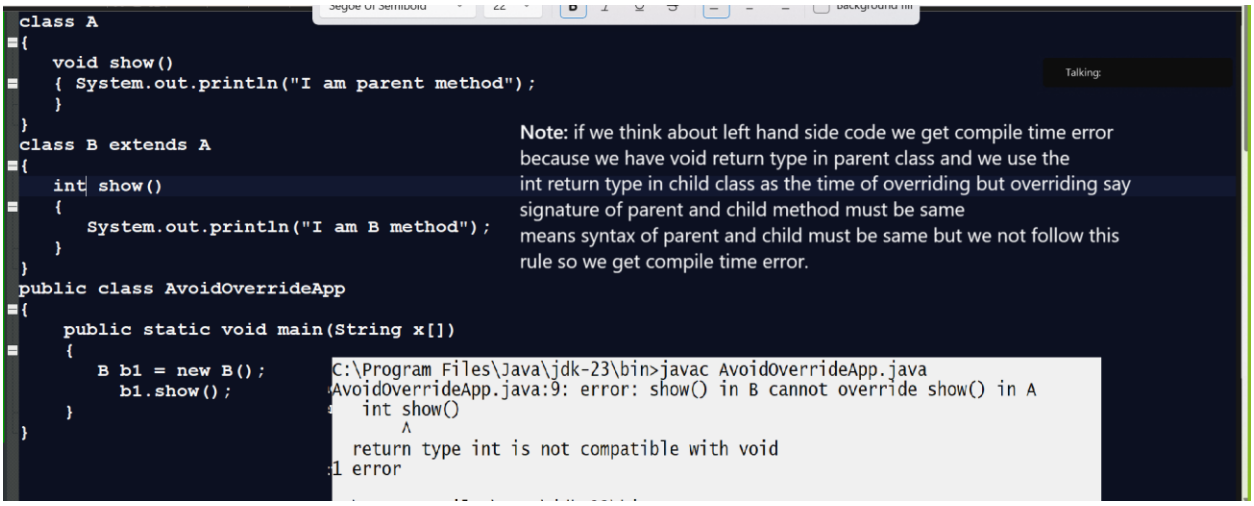
Note: above code generate error to us because we try to declare parent method as final.

If we want to work with method overriding in java we should have to know some

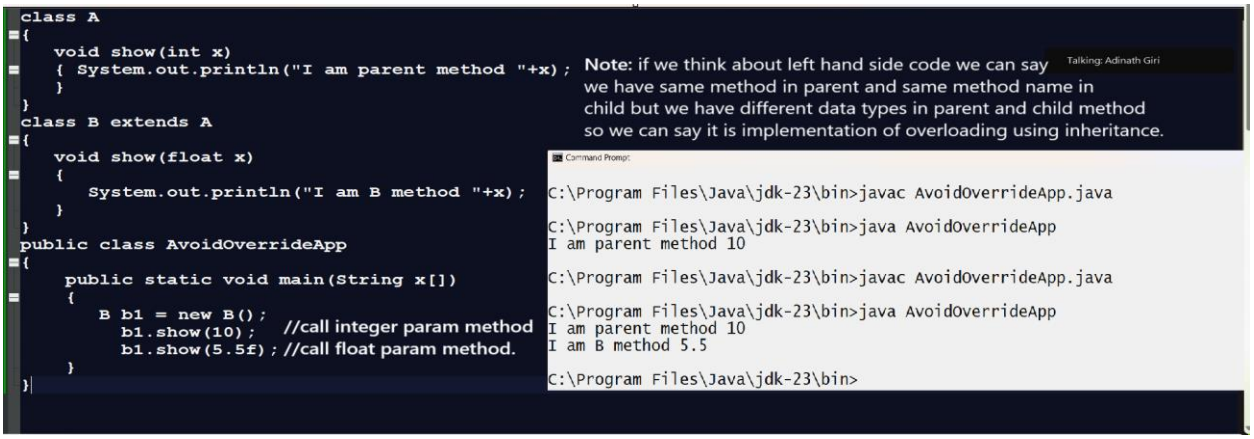
Important points related with overriding

---

1. Return type must be same in overriding as well name, data type and sequence of data type must be same.



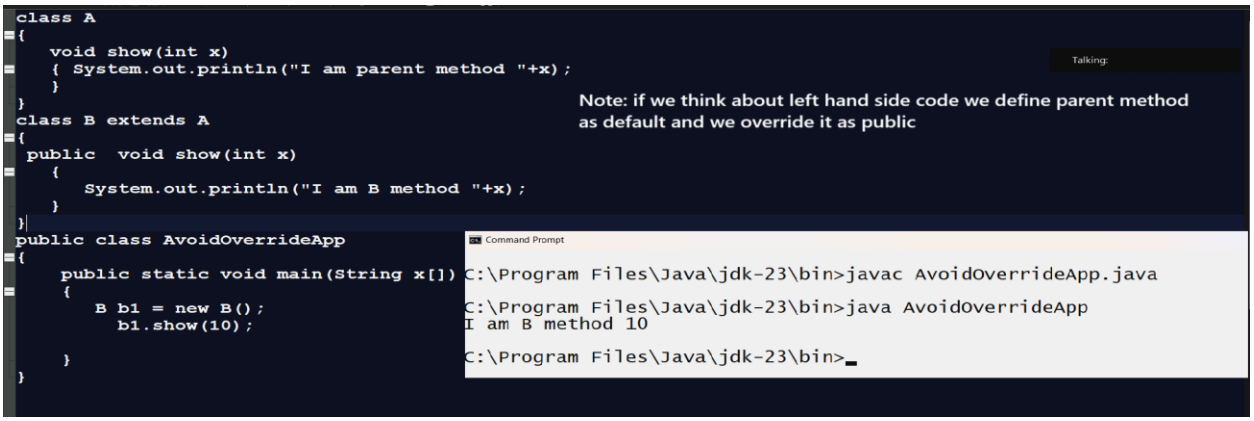
2. If we give return type and method name same and pass different parameter list or different parameter type in function then it is consider as overloading in inheritance.



3. If we have default access specifier or protected access specifier with method in parent class then we can override it as public in child class

---

Because public has higher accessing priority than default or protected



Example protected to public overriding

```
class A
{
  protected void show(int x)
   { System.out.println("I am parent method "+x);
   }
}
class B extends A
{
 public  void show(int x)
   {
      System.out.println("I am B method "+x);
   }
}
public class AvoidOverrideApp
{
    public static void main(String x[])
    {
       B b1 = new B();
         b1.show(10);

    }
}
```

Note: if we think about left hand side we define parent method as protected and we override it as public so it is possible

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

C:\Program Files\Java\jdk-23\bin>java AvoidOverrideApp
I am B method 10

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

4. If we have default method in parent class then we can override it as protected in child

```
class A
{
   void show(int x)
   { System.out.println("I am parent method "+x);
   }
}
class B extends A
{
  protected void show(int x)
   {
      System.out.println("I am B method "+x);
   }
}
public class AvoidOverrideApp
{
    public static void main(String x[])
    {
       B b1 = new B();
         b1.show(10);

    }
}
```

Note: we declare parent method as default and we override it as protected in child class.

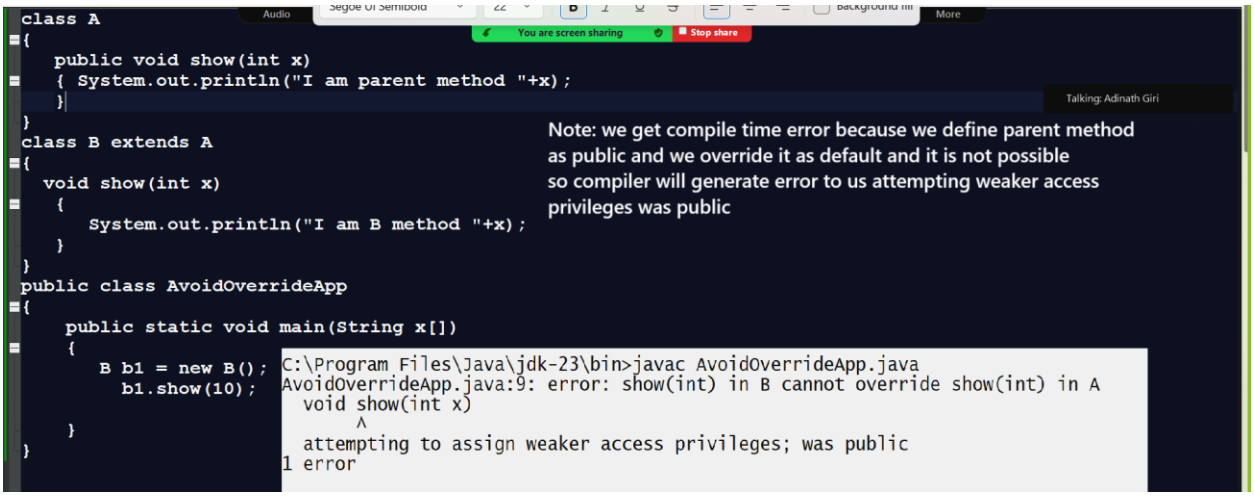
```
C:\Program Files\Java\jdk-23\bin>javac AvoidOverrideApp.java

C:\Program Files\Java\jdk-23\bin>java AvoidOverrideApp
I am B method 10
```

5. If we declare parent method as protected then we cannot override it as default because default has lower priority than protected

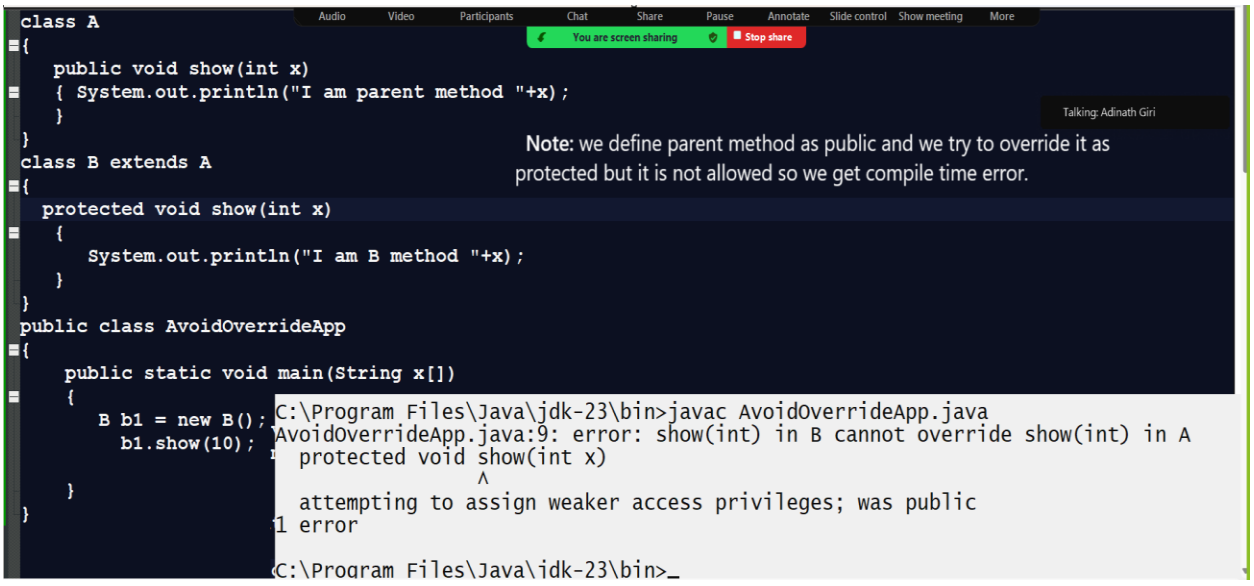
```
class A
{
   protected void show(int x)
   { System.out.println("I am parent method "+x);
   }
}
class B extends A
{
  void show(int x)
   {
      System.out.println("I am B method "+x);
   }
}
public class AvoidOverrideApp
{
    public static void main(String x[])
    {
       B b1 = new B();
         b1.show(10);

    }
}
```

```
C:\Program Files\Java\jdk-23\bin>javac AvoidOverrideApp.java
AvoidOverrideApp.java:9: error: show(int) in B cannot override show(int) in A
  void show(int x)
       ^
  attempting to assign weaker access privileges; was protected
1 error

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

6. If we define parent method using public access specifier then we cannot override it as protected or default and if we try to define as protected or default then we get compile time error.



or

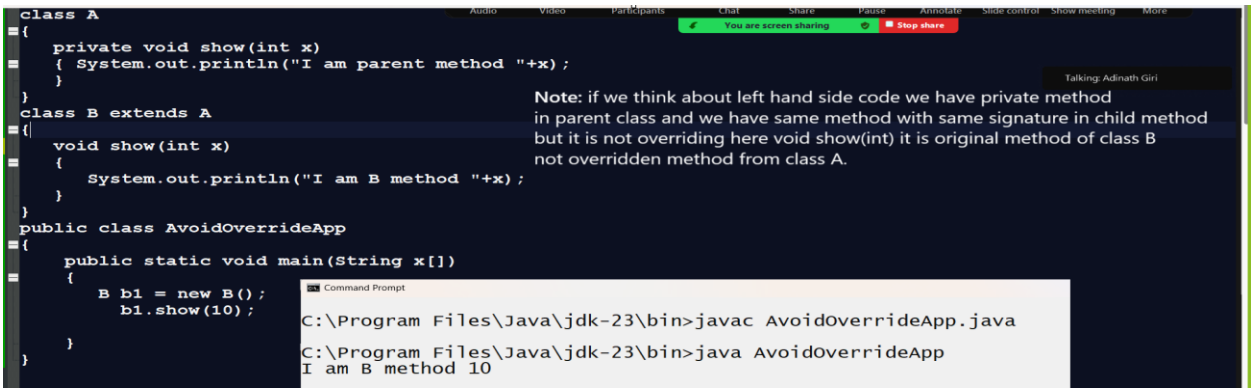


Q. Can we override final method?

No we cannot override final method because final method is used for avoid method overriding in java.

Q. Can we override private method?

No we cannot override private method because private method not support to inheritance and overriding cannot work without inheritance so this is major reason we cannot override private method.



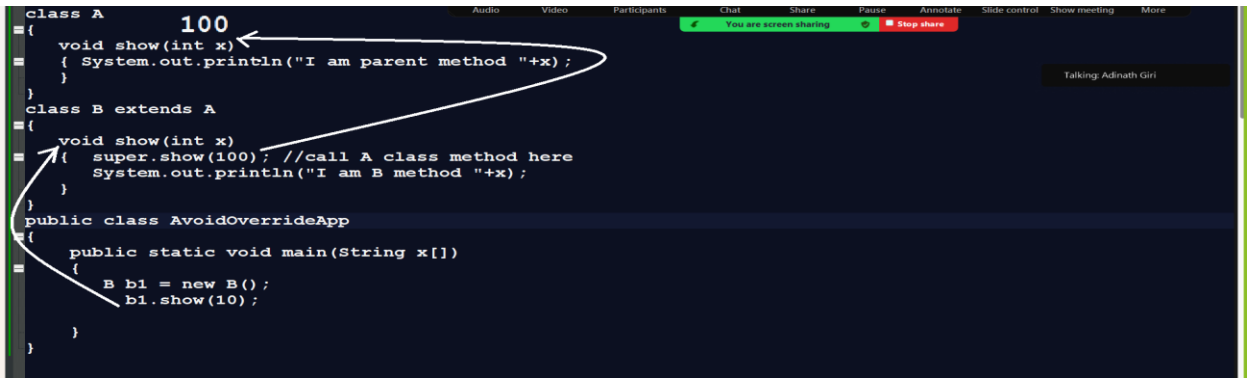
Q. How we can prove show() is not overridden method?

If we want to check method is overridden or not we can check it by using super keyword or super reference.

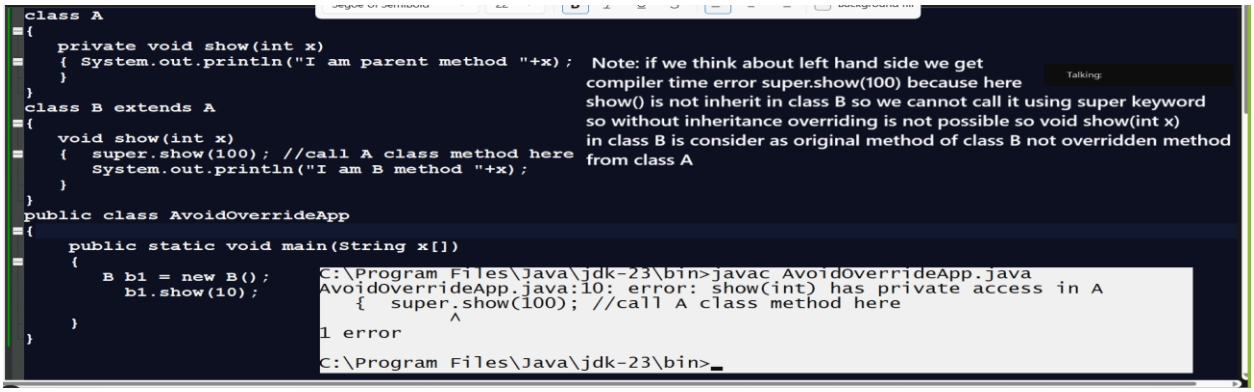
Q. What is super reference?

Super reference or super keyword is internally present in child class which is used for point the parent member from child class object

Normally super refer when we override method or variable from parent class to child class then by default using child object we get overridden method or child version of method so if we want to call parent version in the case of overriding then we can use super reference.

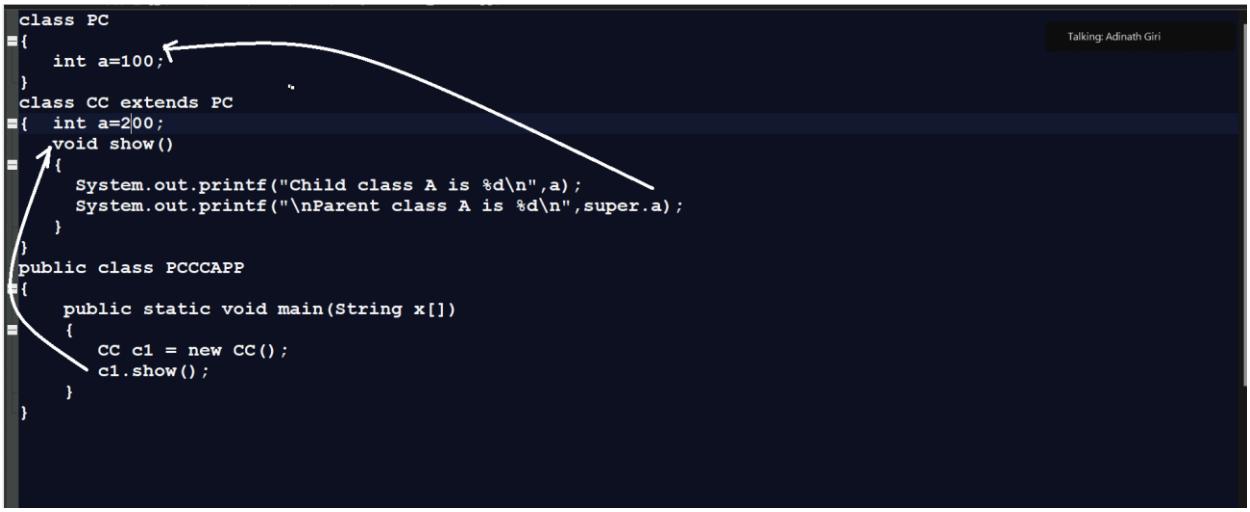


Here super.show(100) call parent version from class B



Q. Can we use parent class variable in child class by using super keyword?

Yes we can access parent class member in child class by using super keyword
Some time if we have same name variable present in parent class and present in child class then we can access parent member using super keyword shown in following code.




```
class PC
{
   int a=100;
}
class CC extends PC
{  int a=200;
   void show()
   {
     System.out.printf("Child class A is %d\n",a);
     System.out.printf("\nParent class A is %d\n",super.a);
   }
}
public class PCCCAPP
{
    public static void main(String x[])
    {
       CC c1 = new CC();
       c1.show();
    }
}
```

Q. What is difference between super() constructor and super keyword?

---

1. Super constructor helps us to call parent class constructor from child class and pass parameter and super keyword can access the parent member in child class.
2. Super constructor is used for perform constructor chaining and super keyword can perform method chaining using overriding technique.

Q. is it true super keyword can access only member of immediate parent?

Its true super keyword access only immediate parent members.

```
class TC
{ int a=400;
}
class PC extends TC
{
   int a=100;
   void show()
   {  System.out.println("TC A is  "+super.a);
   }
}
class CC extends PC
{  int a=200;
   void show()
   {
     System.out.printf("Child class A is %d\n",a);
     System.out.printf("\nParent class A is %d\n",super.a);
     super.show();
   }
}
public class PCCCAPP
{
    public static void main(String x[])
    {
       CC c1 = new CC();
       c1.show();
    }
}
```

Q. can we use static member of parent class using super keyword?

```
class PC
{
   static int a=100;          Note: Yes we can use static member of parent class using super keyword.

}
class CC extends PC
{  int a=200;
   void show()
   {
     System.out.printf("Child class A is %d\n",a);
     System.out.printf("\nParent class A is %d\n",super.a);

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

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

C:\Program Files\Java\jdk-23\bin>java PCCCAPP
Child class A is 200

Parent class A is 100

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

Q. What is difference between static and this reference?

---

1. static reference points to parent class object and these points to current running object in memory.
2. static reference normally use for access parent class member from child class using super keyword and this reference normally used for access the same class member or normally recommend when we have same name instance variable and same name local variable.

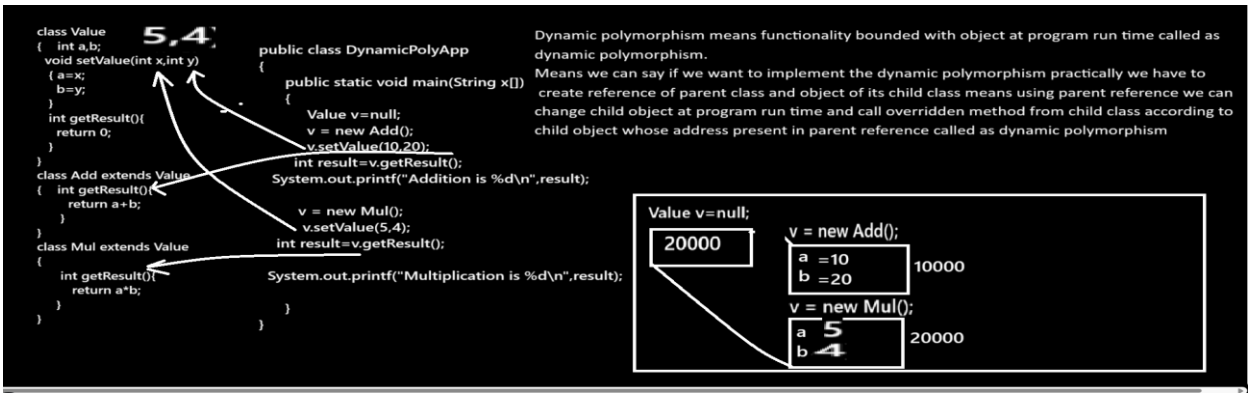
How we can achieve dynamic polymorphism by using method overriding?

Q. What is dynamic polymorphism?

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

Means we can say if we want to implement the dynamic polymorphism practically we have to create reference of parent class and object of its child class means using parent reference we can change child object at program run time and call overridden method from child class according to child object whose address present in parent reference called as dynamic polymorphism

Example:



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;
  }
}
public class DynamicPolyAppAug2024
{
   public static void main(String x[])
          {
            Value v=null;
             v=new Add();
                    v.setValue(10,20);
                    int result=v.getResult();
                    System.out.printf("\nAddition is %d\n",result);
                    v=new Mul();
                    v.setValue(5,4);
                    result=v.getResult();
                    System.out.printf("Multiplication is %d\n",result);
          }
}
```

Output

```
C:\Program Files\Java\jdk-23\bin>javac DynamicPolyAppAug2024.java

C:\Program Files\Java\jdk-23\bin>java DynamicPolyAppAug2024

Addition is 30
Multiplication is 20

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

Example of dynamic polymorphism

```
import java.util.*;
class Bill
{
  protected String prodName;
  protected int qty,rate;
  void setProductDetail(String prodName,int qty,int rate)
  {  this.prodName=prodName;
    this.qty=qty;
          this.rate=rate;
  }
  int getBill(){
    return 0;
  }
}
class WithGST extends Bill
{  int getBill()
  {
   int gstAmt=(qty*rate)*18/100;
          int total=gstAmt+(qty*rate);
          return total;
  }
}
class WithoutGST extends Bill
{int getBill()
  {
   int total=(qty*rate);
          return total;
  }
}
public class GSTWithDynamicPolyApp
{
  public static void main(String x[])
  {
    Scanner xyz  = new Scanner(System.in);
          System.out.println("Enter product name");
          String prodName=xyz.nextLine();
          int qty=xyz.nextInt();
          int rate=xyz.nextInt();
          Bill b=null;
          b=new WithGST();
          b.setProductDetail(prodName,qty,rate);
```

```
        int result=b.getBill();
        System.out.printf("Bill With GST is %d\n",result);
        b=new WithoutGST();
        b.setProductDetail(prodName,qty,rate);
        result=b.getBill();
        System.out.printf("Bill Without GST is %d\n",result);
  }
}
```

Output

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

C:\Program Files\Java\jdk-23\bin>java GSTWithDynamicPolyApp
Enter product name
ABC
10
100
Bill With GST is 1180
Bill Without GST is 1000

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

Q. Can we override static method?

No we cannot override static method when we override static method then method hiding happen not overriding.

Q. What is method hiding?

Method hiding means if we override static method in child class and create reference of parent and object of child class and call overridden method then parent logics get executed because of static keyword so we can say we save method from overriding so we can say it is called as method hiding.

```
class T
{
  static void show()
  { System.out.println("Hey i am parent static method");
  }
}
class C extends T
{
 static void show()
  { System.out.println("Hey i am child static method");
  }
}
public class TestHidingConceptApp
{
```

```
  public static void main(String x[])
  {
     T t1 = new C();
           t1.show();
  }
}
```

Output

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

C:\Program Files\Java\jdk-23\bin>java TestHidingConceptApp
Hey i am parent static method

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

Q. Can we override main method?

No we cannot override main method because main method is static and static method cannot override and when override method then internally it is method hiding so main method can hide but not override

Q. Can we execute logic before main method?

Yes we can execute logic before main method with the help of static block means if static block in class where main method is present then JVM executes the static block before main shown in following code.

Abstract method and Abstract class in JAVA

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

Abstract class means class cannot create its object and abstract method means a method cannot have logics

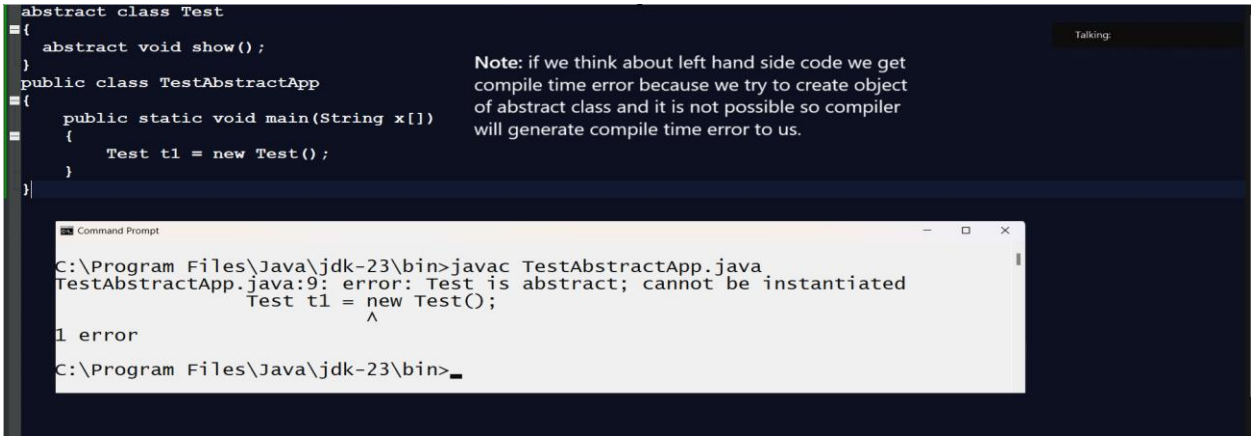
How to declare abstract class and abstract method

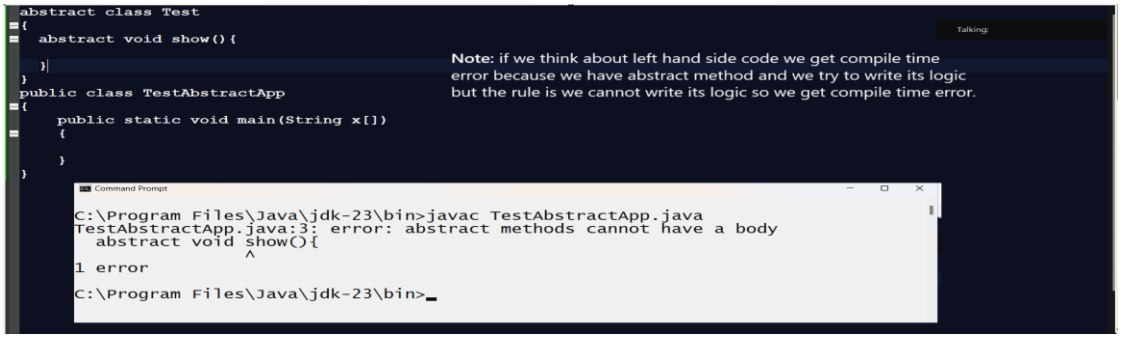
If we want to declare abstract class and abstract method we have to use abstract keyword

Syntax: abstract class classname{

abstract return type functionname(datatype variablename);

}





Q. Why use abstract class and abstract methods?

---

1. To achieve abstraction
2. To achieve dynamic polymorphism.

Q. What is abstraction?

Abstraction means to hide the implementation detail from end user at design level called as abstraction. Means if we want to implement abstraction practically then we provide or create template for method and write its implementation according to future requirement in child classes called as abstraction Means if we think about abstraction we can say we know what to do but don't know how to do it and it is dependent on future requirements.

Example with source 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");
  }
}
```

Example: Suppose consider we are working consultancy or man power hiring agency

---

```
abstract class Employee
{   abstract void skillSet();
}
class Cook extends Employee
{
  void skillSet(){
    System.out.println("Need knowledge related cooking");
  }
}
class Developer extends Employee
{  void skillSet()
   { System.out.println("Need good in coding and communication");
   }
}
```

Note: if we want to write logic of abstract method we required to override abstract method in child classes and write its logic so we can access or call abstract method by using child class of abstract class.

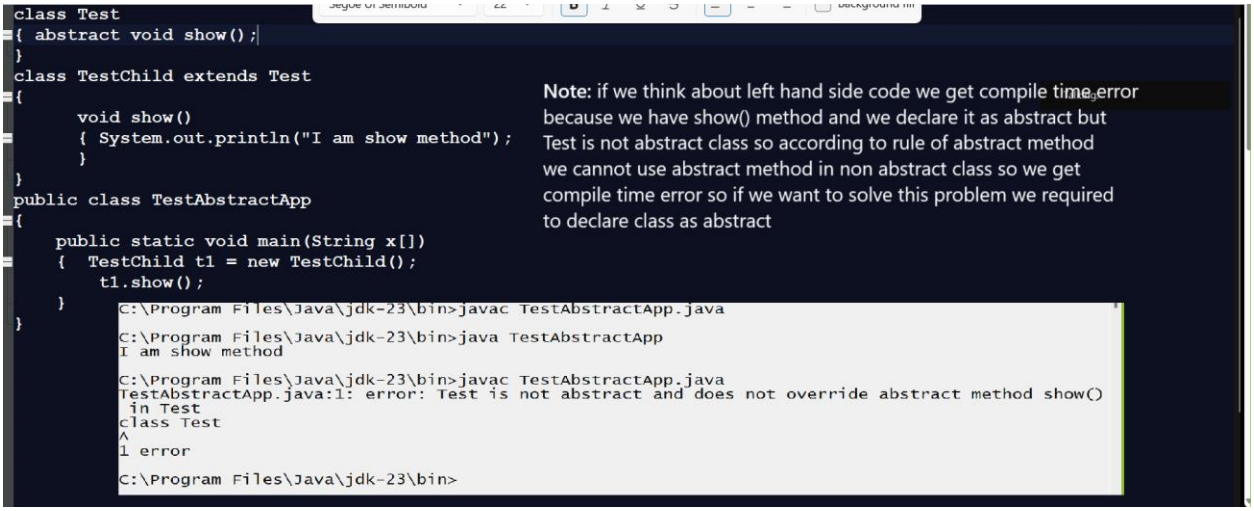
```
abstract class Test
{ abstract void show();
}
class TestChild extends Test
{
      void show()
      { System.out.println("I am show method");
      }
}
public class TestAbstractApp
{
    public static void main(String x[])
    {  TestChild t1 = new TestChild();
       t1.show();
    }
}
```

```
Command Prompt

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

C:\Program Files\Java\jdk-23\bin>java TestAbstractApp
I am show method

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

If you want to work with abstract class and abstract methods we have know the some important points.

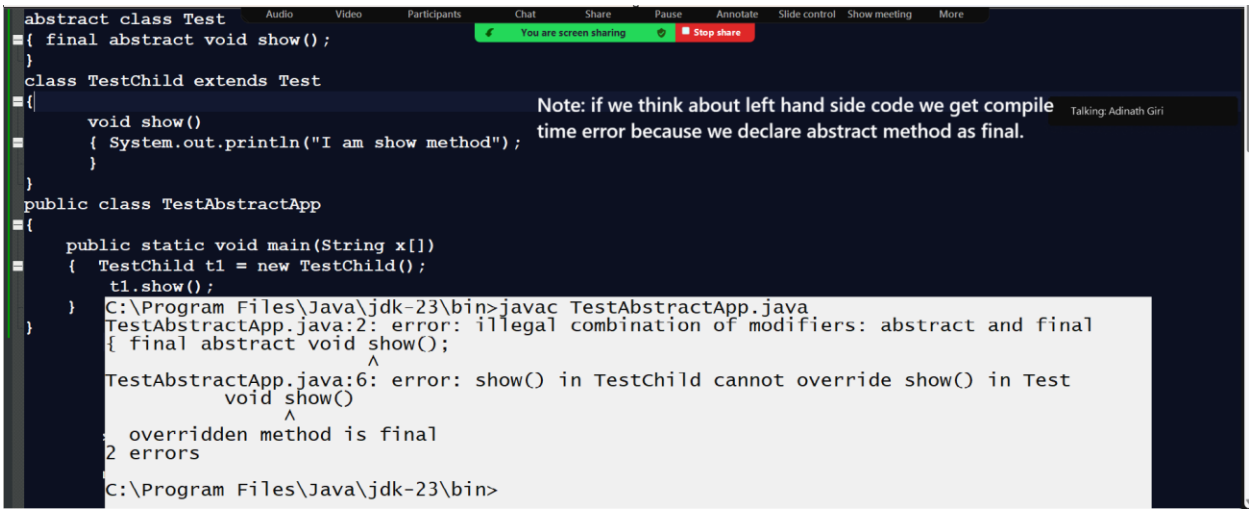
---

1. Abstract class cannot create its object.
2. Abstract method cannot have logics
3. If method is abstract then class must be abstract means abstract method cannot declare in non abstract class.



Q. Can we declare abstract method as final?

No we cannot declare abstract method as final because final method is used for avoid method overriding and abstract method must be override so we can say final and abstract has opposite behavior so we cannot declare abstract method as final and if we declare abstract method as final we get compile time error.



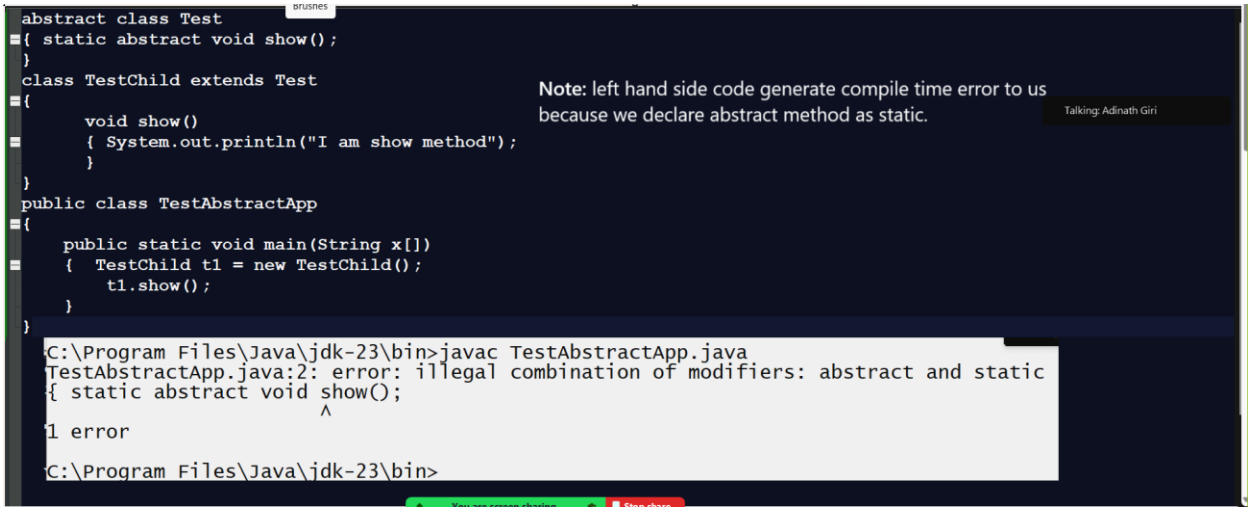
Q. Can we declare abstract method as static?

No we cannot declare abstract method as static because there are three reasons.

a) Static method support to compile time polymorphism and abstract method specially design for achieve dynamic polymorphism and loose coupling.

b) static method support to method hiding purpose and abstract just work with method overriding

c) static method must have definition or logical block but abstract method cannot have logical block.



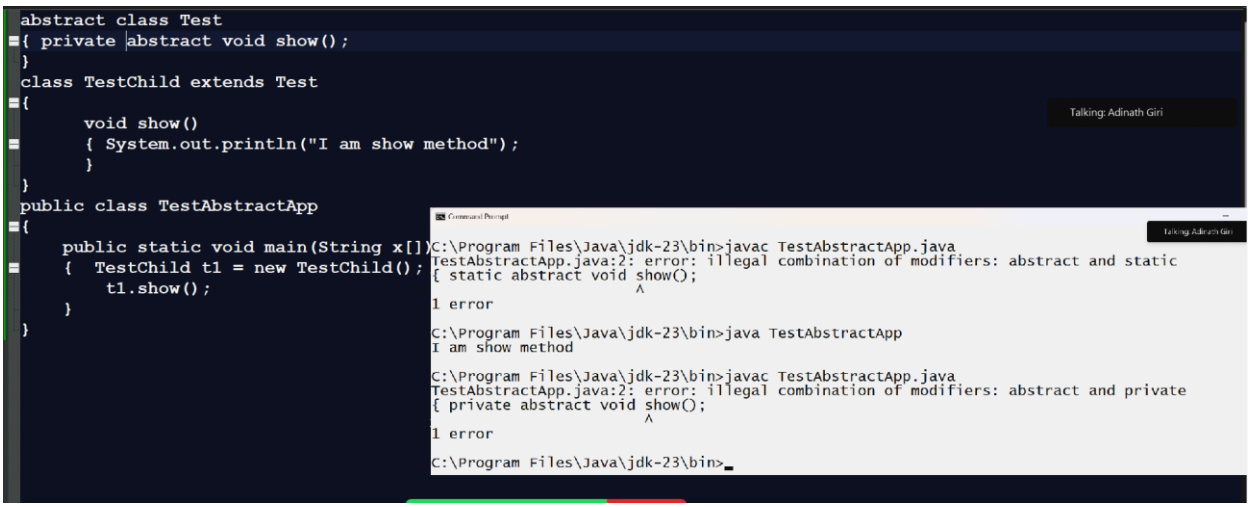
Q. Can we declare abstract method as private?

We cannot declare abstract method as private because

1) Private method not support to inheritance and abstract cannot work without inheritance

2) Private cannot support to overriding because overriding is not possible without inheritance and abstract specially design for achieve dynamic polymorphism using overriding.

Means we can say abstract and private has different behavior or working principal so we cannot declare abstract method as private and if we declare it as private then we get compile time error.



Q. Can we declare abstract method as protected?

Yes we can declare abstract method as protected because protected can support to inheritance and overriding so this is major reason we can declare abstract method as protected

```
abstract class Test
{ protected abstract void show();
}
class TestChild extends Test
{
     protected void show()
      { System.out.println("I am show method");
      }
}
public class TestAbstractApp
{
    public static void main(String x[])
    {  TestChild t1 = new TestChild();
       t1.show();
    }
}
```

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

C:\Program Files\Java\jdk-23\bin>java TestAbstractApp
I am show method

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

Note: if abstract class contains more than one abstract method then all methods must be override where abstract class get inherit if we not required then define as blank or override as blank if we not override any one method then we get compile time error.

```
abstract class Test
{
   abstract void s1();
   abstract void s2();
   abstract void s3();
}
class TestChildFirst extends Test
{
    void s1(){
        System.out.println("I Required s1 method");
    }
}
class TestChildSecond extends Test
{
    void s2()
    { System.out.println("I Required s2 method");
    }
}
public class AbsWithMoreMethodApp
{
    public static void main(String x[])
    {
        TestChildFirst tf = new TestChildFirst();
        tf.s1();
        TestChildSecond ts= new TestChildSecond();
        ts.s2();
    }
}
```

Note: if we think about left hand side code we get compile time error because we have abstract class which contain three abstract methods and we inherit this class in Two different classes but we override single method from 3 abstract methods but need to override all so we get compile time error

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac AbsWithMoreMethodApp.java
AbsWithMoreMethodApp.java:7: error: TestChildFirst is not abstract and does not override abstract metho
d s3() in Test
class TestChildFirst extends Test
^
AbsWithMoreMethodApp.java:14: error: TestChildSecond is not abstract and does not override abstract met
hod s3() in Test
class TestChildSecond extends Test
^
2 errors

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

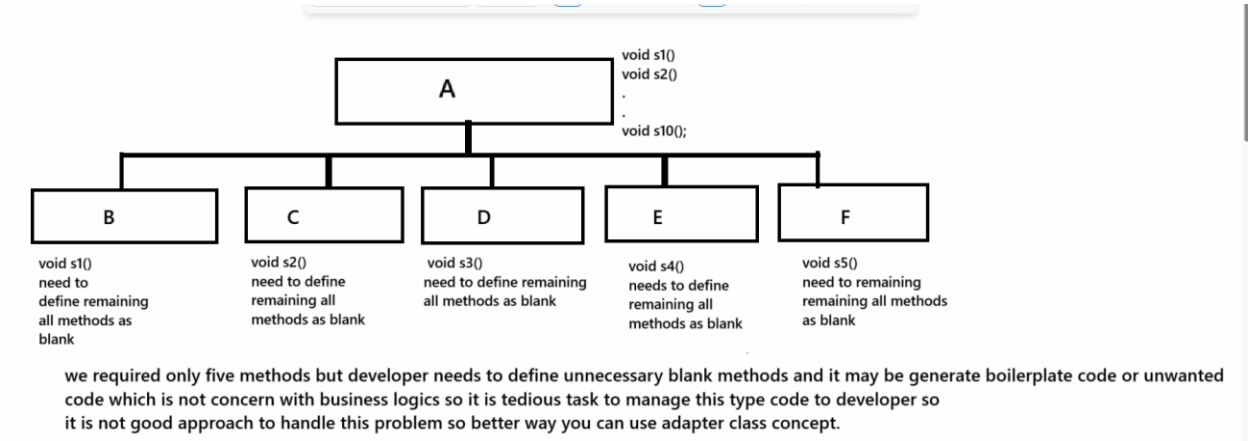
So if we want to solve above problem we have two solutions.

1) Define all methods or override all method as blank

```
abstract class Test
{
  abstract void s1();
  abstract void s2();
  abstract void s3();
}
class TestChildFirst extends Test
{  void s1(){
```

```
        System.out.println("I Required s1 method");
    }
    void s2(){
    }
    void s3(){
    }
}
class TestChildSecond extends Test
{
    void s2()
    { System.out.println("I Required s2 method");
    }
    void s1(){
    }
    void s3(){
    }
}
public class AbsWithMoreMethodApp
{
    public static void main(String x[])
    {
        TestChildFirst tf = new TestChildFirst();
        tf.s1();
        TestChildSecond ts= new TestChildSecond();
        ts.s2();
    }
}
```

Note: if we think about above code or above approach there is some limitations if we have abstract class which contain 10 abstract methods and we have five child classes of abstract class and every child class required single abstract method In every child class then we required to override 50 methods but 45 blank method and 5 with logic methods



2) Use Adapter class

Q. What is adapter class?

Adapter class is intermediate class which contain all blank methods of abstract class or interface and which is able to provide specific method to child of abstract class or interface called as adapter class Means we can say adapter class can work as bridge or intermediate communicator between abstract class and its child classes or interface and its implementer classes

Example with source code

```
abstract class Test
{
  abstract void s1();
  abstract void s2();
  abstract void s3();
}
class TestADP extends Test
{  void s1(){
   }
   void s2(){
   }
   void s3(){
   }
}
class TestChildFirst extends TestADP
{  void s1(){
        System.out.println("I Required s1 method");
   }

}
class TestChildSecond extends TestADP
{  void s2()
   { System.out.println("I Required s2 method");
   }
}
public class AbsWithMoreMethodApp
{
   public static void main(String x[])
   {
        TestChildFirst tf = new TestChildFirst();
        tf.s1();
        TestChildSecond ts= new TestChildSecond();
```

```
        ts.s2();
    }
}
```

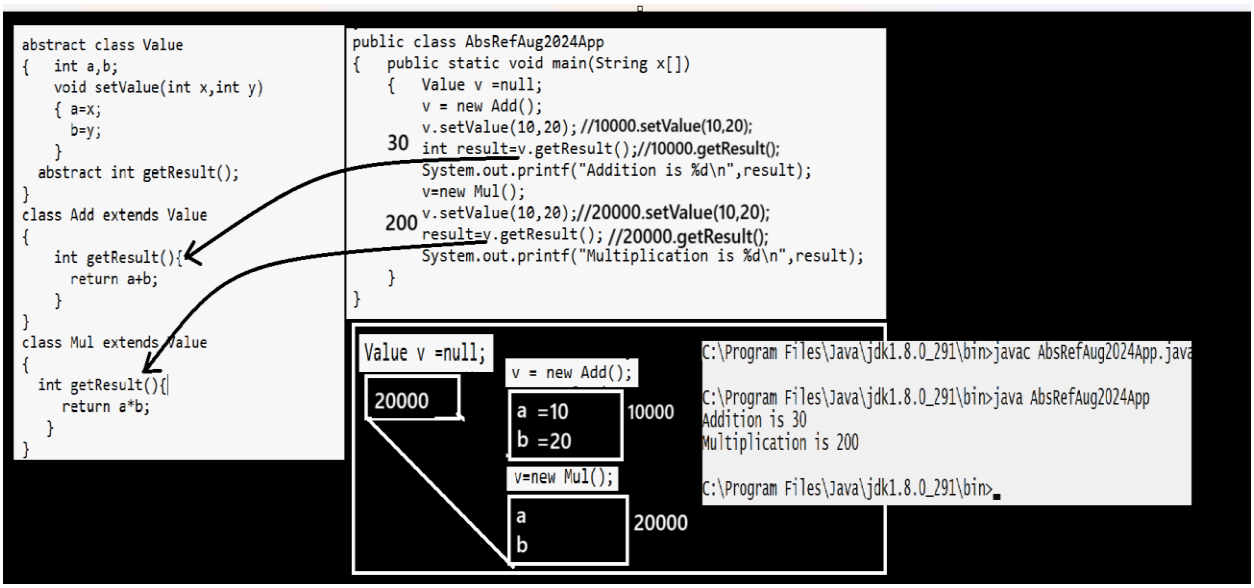
Q. What is adapter design pattern or adapter class?

Adapter design pattern is used for covert the limitation of abstract class and abstract methods.
if abstract class contains more than one abstract method then all methods must be override where abstract class get inherit if we not required then define as blank or override as blank then we can solve this problem using adapter class or design pattern and we can implement shown in above example

Q. Can we create reference of abstract class?

Yes we can create reference of abstract class but we cannot create its object and if we want to create reference of abstract class we required to create object of its child class and the major purpose abstract class reference is achieve dynamic polymorphism and loose coupling.

Following code is implementation of dynamic polymorphism using abstract class reference.



Another purpose of abstract class reference is achieve loose coupling

Q. What is coupling?

Coupling means if one object is dependent on another object called as coupling or if particular function calling is also dependent on another object called as coupling or dependency injection

There are two types of coupling?

---

1. Tight coupling: Tight coupling means if particular method is 100% dependent on particular class reference called as tight coupling.

2. Loose Coupling: loose coupling means if particular method of class is partially dependent on particular class object called as loose coupling and loose coupling can achieve by using dynamic polymorphism concept.

Example of tight Coupling

```
class Add
{ int a,b;
  void setValue(int x,int y)
   { a=x;
     b=y;
   }
  int getResult(){
   return a+b;
  }
}
class Mul
{  int a,b;
  void setValue(int x,int y)
   { a=x;
     b=y;
   }
  int getResult(){
    return a*b;
  }
}
```

```
class Calculator
{
     void performOperation(Add ad)
     {
       int result= ad.getResult();

       System.out.printf("Addition is %d\n",result);
     }
}

public class TightCouplingApp
{  public static void main(String x[])
   {   Calculator c=new Calculator();
       Add ad = new Add();
         ad.setValue(10,20);
       c.performOperation(ad);
       Mul  m = new Mul();
         m.setValue(5,4);
       c.performOperation(m);
   }
```

Note: if we think about left hand side code we get compile time error because we have function name as performOperation() in Calculator class and in this method we have parameter name as Add class object and we try to pass Mul class object in performOperation() method so it is not possible so we get compile time error
so we can say it is called as tight coupling.

If we want to resolve the above problem we have two solutions

1. Solve it using compile time polymorphism

means if we want to solve above compile time error we can overload the performOperation() method using two different parameter shown in below

void performOperation(Add ad): this function can accept Add reference from calling point
void performOperation(Mul m): this function can accept Mul reference from calling point**.**

Example with source code

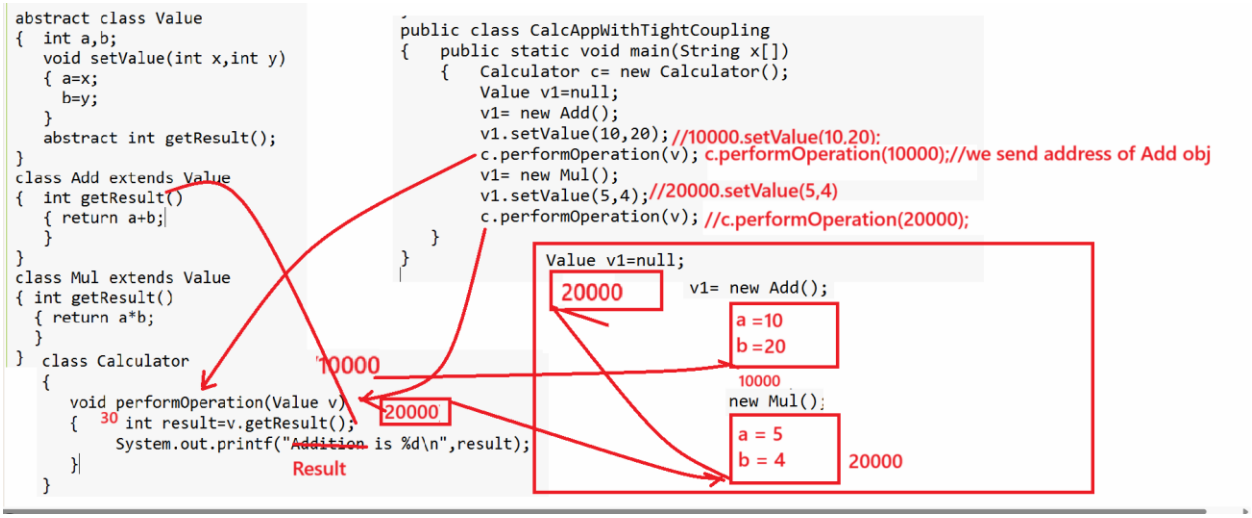
```
class Add
{
  int a,b;
  void setValue(int x,int y)
  { a=x;
    b=y;
  }
  int getResult()
  { return a+b;
  }
}
class Mul
```

```
{ int a,b;
  void setValue(int x,int y)
  { a=x;
    b=y;
  }
  int getResult()
  { return a*b;
  }
}
class Calculator
{
   void performOperation(Add ad)
   {
      int result=ad.getResult();
            System.out.printf("Addition is %d\n",result);
   }

  void performOperation(Mul m)
   {
      int result=m.getResult();
            System.out.printf("Multiplication is %d\n",result);
   }
}
public class CalcAppWithTightCoupling
{   public static void main(String x[])
    {   Calculator c= new Calculator();
            Add ad = new Add();
            ad.setValue(10,20);
            c.performOperation(ad); //call Add class version
            Mul  m = new Mul();
            m.setValue(5,4);
            c.performOperation(m); //call Mul class version
   }
}
```

The limitation of compile time polymorphism is suppose consider we have 100 different operation in Calculator class and declare 100 different classes for manage the 100 operations so we required to overload 100 different performOperation() methods in Calculator class so it is not possible in real time scenario

So we want to design only one method which should able access the object of any class then loose coupling or dynamic polymorphism comes in picture

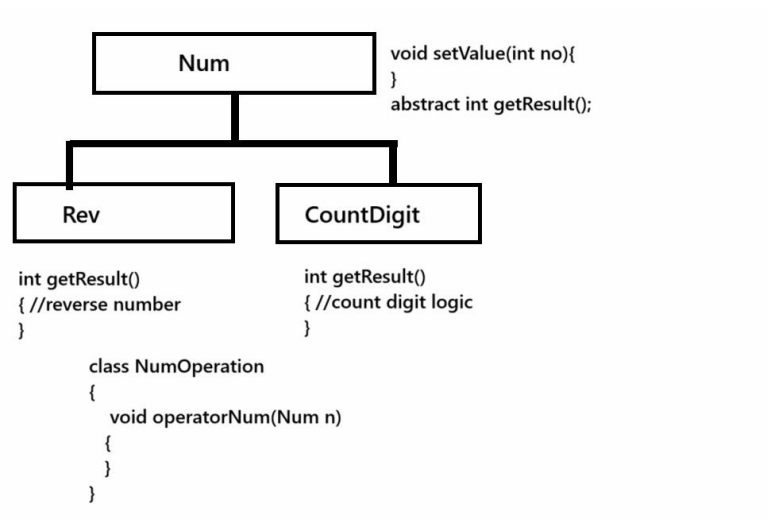
2. Solve it using run time polymorphism



Note: If we think about above code it is implementation of loose coupling.

```
abstract class Value
{ int a,b;
  void setValue(int x,int y)
  { a=x;
    b=y;
  }
  abstract int getResult();
}
class Add extends Value
{ int getResult()
  { return a+b;
  }
}
class Mul extends Value
{ int getResult()
 { return a*b;
 }
}
class Calculator
{
  void performOperation(Value v,String type)
  {    int result=v.getResult();
          System.out.printf("%s is %d\n",type,result);
  }
}
public class CalcAppWithLooseCoupling
```

```
{   public static void main(String x[])
   {   Calculator c= new Calculator();
            Value v1=null;
            v1= new Add();
            v1.setValue(10,20);
            c.performOperation(v1,"Addition"); //call Add class version
            v1= new Mul();
            v1.setValue(5,4);
            c.performOperation(v1,"Multiplication"); //call Mul class version
  }
}
```



Example with source code

```
abstract class Num
{   int no,r=0;
   void setValue(int x)
   {  no=x;
   }
  abstract int getResult();
}
class Rev extends Num
{

   int getResult()
   { while(no!=0)
     {  int rem=no%10;
            no=no/10;
```

```
            r=r*10+rem;
    }
     return r;
  }
}
class CountDigit extends Num
{
  int getResult()
  { while(no!=0)
    {  int rem=no%10;
            no=no/10;
             ++r;
    }
     return r;
  }
}
class NumOperation
{
   public void operateNum(Num n,String operationType)
   {  int result=n.getResult();
            System.out.printf("%s\t%d\n",operationType,result);
   }
}
public class TestLooseCoupledApp
{  public static void main(String x[])
  {
            NumOperation nop=new NumOperation();
            Num n1 = new Rev();
            n1.setValue(1234);
            nop.operateNum(n1,"Reverse Number");
       n1=new CountDigit();
       n1.setValue(12345);
            nop.operateNum(n1,"Number of Digit");
  }
}
```

Q. Can we write non abstract methods in abstract class?

Yes we can write non abstract method in Abstract class and when we write non abstract methods in abstract class then class known as concrete class shown in following code.

```
abstract class Num
```

```
{  int no,r=0;
  void setValue(int x)
  {  no=x;
  }
 abstract int getResult();
}
```

If we think about above code we can say we have Num class and it mark as concrete class because it contain setValue() method which is non abstract method and getResult() method which is abstract method so it is combination of abstract and non abstract so it is concrete

Q. Can we use abstract keyword with variable?

No we cannot use the abstract keyword with variable because variable is used for just store data or manage state of object using variable we cannot achieve abstraction and abstract keyword specially design for manage the abstraction so we this is major reason we cannot use the abstract keyword with variable.

Q. Can we declare constructor as abstraction?

No we cannot declare constructor as abstract and if we try to declare it as abstract then we get compile time error

Declaring a constructor as abstract is meaningless because when we declare any method as abstract then we must be override and constructor cannot support to overriding so this is major we cannot declare constructor as abstract

```
abstract class Num
{   int no,r=0;
   abstract Num();
   void setValue(int x)
   {  no=x;
   }
  abstract int getResult();
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac TestLooseCoupledApp.java
TestLooseCoupledApp.java:3: error: modifier abstract not allowed here
   abstract Num();
            ^
1 error
```

Q. Can we declare constructor as final?

No we cannot declare constructor as final because final keyword is used for avoid method overriding and constructor not support to overriding so declaring it as final is meaningless so we cannot declare constructor as final.

```
abstract class Num
{   int no,r=0;
   final Num(){

   }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac TestLooseCoupledApp.java
TestLooseCoupledApp.java:3: error: modifier final not allowed here
   final Num(){
         ^
1 error

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

Q. What is difference between final and abstract keyword?

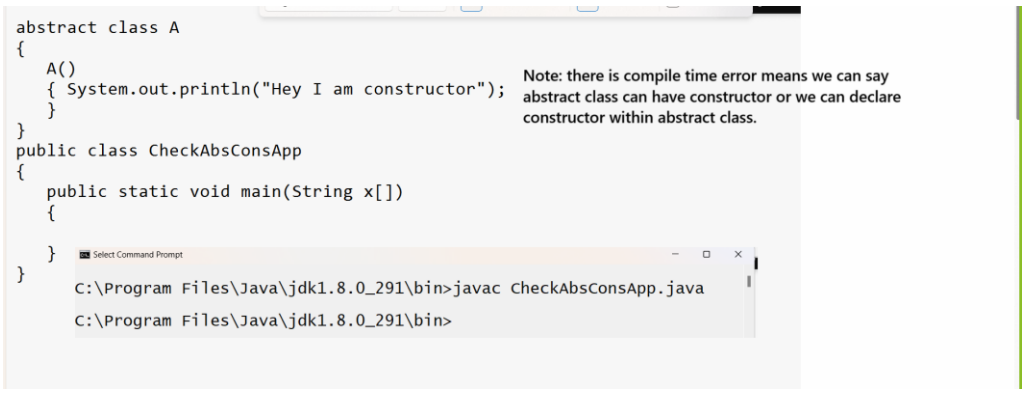
Final keyword	Abstract keyword
We can use with variable, function and with class	We can use with function and class only
Is used for avoid method overriding	Must be override method when we use abstract keyword
Can write logic of final method	Cannot write logic of abstract method
When we use final with class cannot perform inheritance	When we use the abstract with class then must be perform inheritance
Final class is used for create immutable classes	Abstract class is used for achieve abstraction
Final method not support to dynamic polymorphism because we cannot override it	Abstract method can support to dynamic polymorphism because we must be override

Q. What is difference between static and abstract?

Static	Abstract
Static keyword can use with variable or with function or with nested class	Abstract keyword can use with class and function only
When we use static keyword with method then method must have definition	When we use abstract with method then method cannot have definition
When we use static keyword with method and try to override then method hiding happen	When we use the abstract keyword with method then actual method overriding happen
Static method support to compile time polymorphism	Abstract method support to dynamic polymorphism or loose coupling or run time polymorphism or late binding
Static member can allocate before creating object of class	Abstract member can allocate memory after creating child class object

Q. Can we declare constructor within abstract class?

Yes we can declare constructor in abstract class shown in following code.



Q. If abstract class not creates its object then why we need to declare constructor within abstract class?

If we have some instance variable or member in abstract class and if we want to initialize them when we create object of the class then we can initialize them using abstract class constructor means abstract class constructor call when we create object of child class

```
abstract class A
{ int no;
  A()
  { System.out.println("Hey I am parent constructor");
        no=100;
  }
}
class AC extends A
{
  AC()
  { System.out.println("I am child class constructor");
  }
  void show()
  { System.out.println("I am child class method  "+no);
  }
}
public class CheckAbsConsApp
{
  public static void main(String x[])
  {
        AC a1 = new AC();
        a1.show();
  }
}
```

Output

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

C:\Program Files\Java\jdk1.8.0_291\bin>java CheckAbsConsApp
Hey I am parent constructor
I am child class constructor
I am child class method  100

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

Q. If abstract class not creates its object when abstract constructor gets executed?

When we create object of abstract child then abstract class constructor get executed.

Q. What is difference between encapsulation and abstraction?

Encapsulation	Abstraction
Encapsulation means hide the implementation detail from end user at implementation level	Abstraction means hide the implementation detail from end user at design level
We can implement encapsulation by declaration variable of class as private	We can implement abstraction by declaring method as abstract
Goal of encapsulation is to provide data security	Goal of abstraction is to achieve dynamic polymorphism
If we think about encapsulation we know what to and how to do it	If we think about abstraction we know what to do but don't how to do it
Encapsulation can achieve by using single class or using inheritance also	Abstraction can only achieve by using inheritance

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

_____

Method Overloading	Method Overriding
Method overloading is compile time polymorphism	Method overriding is a run time polymorphism
It can occurs within class or may be inheritance	But overriding always occurs in inheritance.
Method overloading method must have same name but different signatures or syntaxes	In method overriding parent method and child method signature must be same
Method overloading which method should call is dependent on data type and number of parameter in it and parameter list with sequence	In method overriding which method should execute is dependent on child class object at run time
Method overloading is used for avoid to remember the multiple name of method when we have multiple logic under the same domain	Method overriding also help us to achieve abstraction using abstract keyword

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

Compile time polymorphism	Dynamic polymorphism
Compile time polymorphism can achieve by using overloading	Dynamic polymorphism can achieve by using method overriding
In Compile time polymorphism method or functionality bounded with bounded with object at program compile time means which function should call decide at compile time by using same object	In Run time polymorphism method or functionality bounded with objet at program run time means same function can call by using more than one object at program run time
Compiler time polymorphism can achieve using single class means not mandatory to use	Runtime polymorphism can achieve by using inheritance means it is mandatory to use

inheritance	inheritance for achieve run time polymorphism
Compile time polymorphism can support to tight coupling	Runtime polymorphism can support to lose coupling.

Interface

Q. What is interface in JAVA?

Interface is same like as abstract class in JAVA

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

There are some reasons

1. To achieve 100% abstraction
2. To achieve dynamic polymorphism
3. To implement multiple inheritance

How to use interface in JAVA?

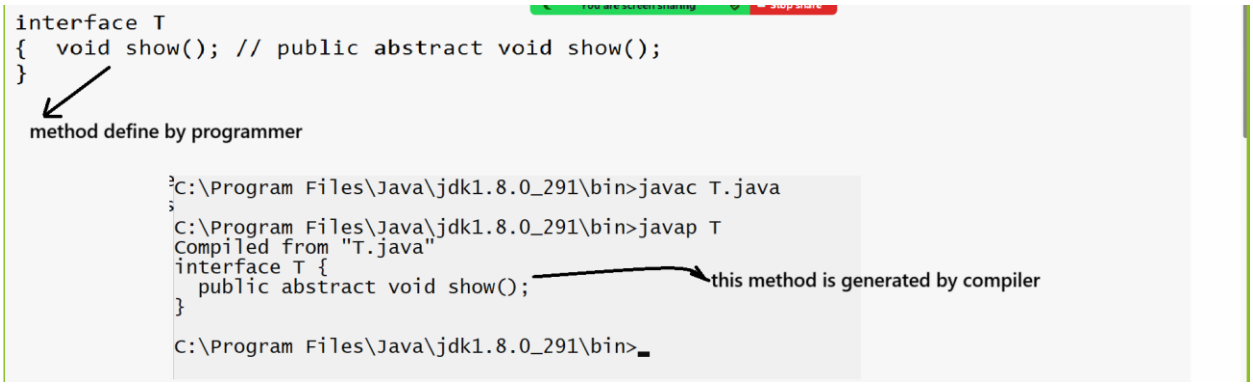
Syntax:

```
interface interfacename{
  returntype functionname(datatype variablename);
}
```

Example

```
interface Area
{ void setRadius(float radius);
}
```

Note: if we think about interface by default interface methods are public abstract



Q. How we can prove interface can achieve 100% abstraction & abstract class can achieve partial abstraction?

If we think about interface by default internally it is abstract class and by default every method of interface is abstract but if we think about abstract class method they are not by default abstract means we need to mark interface method as abstract manually and abstract contain non abstract method also so we can say interface can achieve 100% abstraction and abstract class not
Note: from JDK 1.8 version of java we can define method within interface using static or default **keyword but static keyword not support to abstraction**

Q. Where we can write logic of interface method?

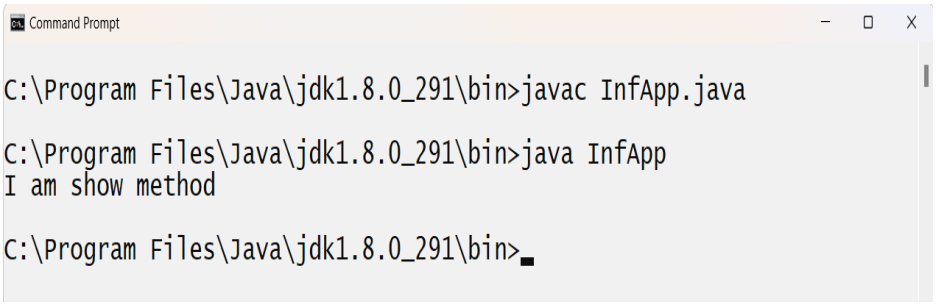
If we want to write logic of interface method then we need to implement interface in another class and override the method of interface and write its logic and if we want to implement the interface we have implements keyword.

Syntax: class classname implements interfacename{
}

File Edit View

```
interface T
{  void show(); // public abstract void show();
}
class Tchild implements T
{
   public void show()
   { System.out.println("I am show method");
   }
}
public class InfApp
{
    public static void main(String x[])
    {
        Tchild tc = new Tchild();
        tc.show();
    }
}
```

Ln 17, Col 2 278 characters



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

---

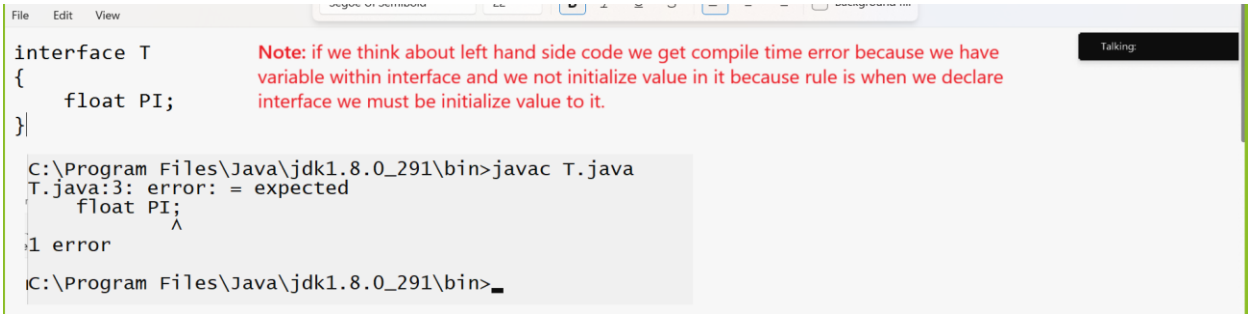
1. Interface cannot create is object: Because interface is internally abstract class and abstract class cannot create its object so interface cannot create its object.

2. Interface method cannot have logics: Because when we declare method within interface by default it is public and abstract and abstract method cannot have logics

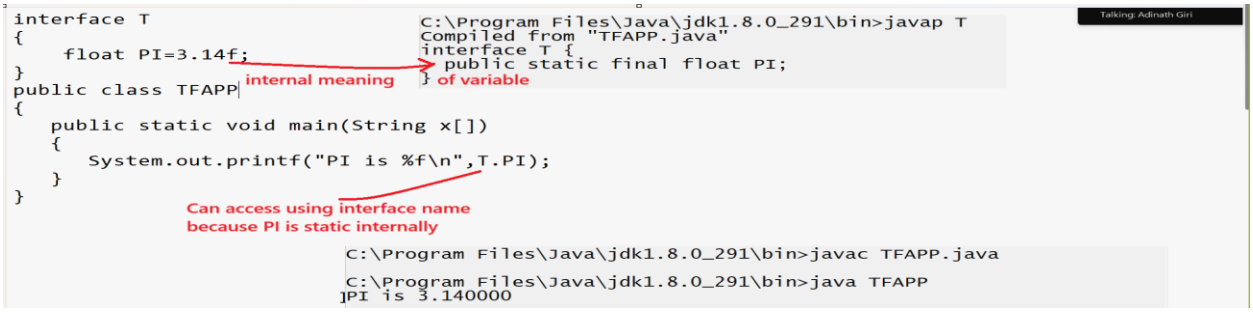
3. Interface variables are by default public static final

Means when we declare the variable within interface we must be initialize some value in it if we declare the variable in interface and not initialize value in it then we get compile time error.

Means normally we declare variable in interface those want to mark as constant and access without object means access by using interface

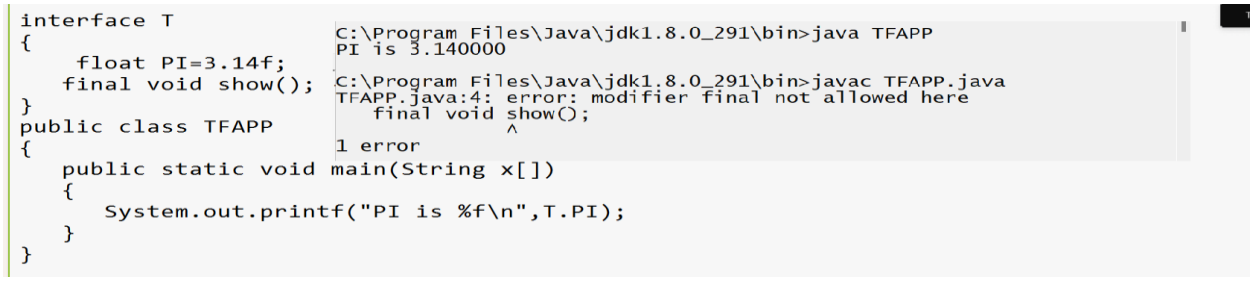


So if we want to solve above compile time error we must be initialize value to variable PI as per our example shown in following code.



Q. Can we declare interface method as final?

No we cannot declare interface method as final because internally interface method is abstract and we cannot declare abstract method as final so we cannot declare the interface method as final and if we try to declare it as final we get compile time error shown in following code.



Q. Can we declare interface method as static?

No we cannot declare interface method as static because interface methods internally abstract and we cannot mark or declare abstract method as static so we cannot declare interface method as static and if we try to declare it as static then we get compile time error.

```
interface T
{
    float PI=3.14f;
   static void show();
}
public class TFAPP
{
   public static void main(String x[])
   {
      System.out.printf("PI is %f\n",T.PI);
   }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac TFAPP.java
TFAPP.java:4: error: missing method body, or declare abstract
   static void show();
               ^
1 error

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

Q. Can we define interface method as static?

Yes we can define interface method as static from JDK1.8 version of JAVA

```
interface T
{
    float PI=3.14f;
   static void show(){
     System.out.println("I am static method");
   }
}
public class TFAPP
{
   public static void main(String x[])
   {
      T.show();
   }
}
```

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

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

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

Note: we can define static method within interface and we call the static by using interface name and not need to implement interface if we want to access static method

Q. Can we define interface method as private?

We cannot declare interface method as private because there are some reasons.

a) private cannot support to inheritance and overriding and interface method must be override so cannot declare as private

b) Interface method is by default public and when we declare it as private then internally it is consider as private public and in java there is no private public either we have public or private separately not in combination

```
interface A
{                  // internal meaning
  void show();  //     public abstract void show();
}

if we write method as private then internally it is consider as

interface A
{
   private void show(); //private public abstract void show();
}
```

cannot use private public in combination with same method so we get compile time error.

Q. Can we declare interface method as protected?

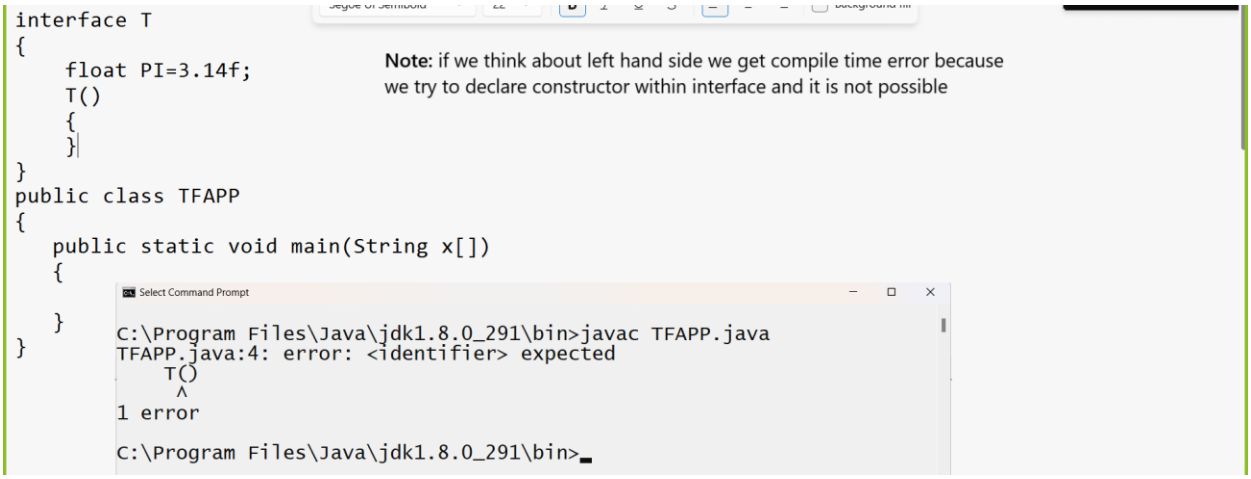
No we cannot declare interface method as protected because when we declare method in interface then internally it is public abstract and when we write protected interface method then it is consider as protected public abstract and we cannot use protected and public at same time with single method so we get compile time error so we can say we cannot declare interface method as protected and if we try to declare it as protected then we get compile time error.

```
interface T
{
    float PI=3.14f;
    protected void show();

}
public class TFAPP
{
   public static void main(String x[])
   {
   }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac TFAPP.java
TFAPP.java:4: error: modifier protected not allowed here
    protected void show();
                   ^
1 error

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

Q. Can we declare constructor within interface?

No we cannot declare constructor within interface because the main goal of constructor is to initialize the member/variable/state of class but if we think about interface when we declare variable in it then must be initialize so not need to initialize after declaration so according to coding standard there is no need of constructor so this is major reason interface cannot have constructor even it is internally abstract but abstract class need constructor.



Q. Can we create reference of interface?

Yes we can create reference of interface and for that we required to create object of its implementer class and the goal of interface reference is achieve dynamic polymorphism or loose coupling.

Example with source code

```
interface Num
{  void setValue(int no);
   int getResult();
}
class Rev implements Num
{   int no;
   public void setValue(int no)
   {  this.no=no;
   }
   public int getResult()
   {  int rev=0;
     while(no!=0)
     {
            int rem=no%10;
            no=no/10;
            rev=rev*10+rem;
     }
     return rev;
   }
}
class CountDigit implements Num
{ int no;
   public void setValue(int no)
   {  this.no=no;
```

```
  }
  public int getResult()
  {  int count=0;
    while(no!=0)
    {
          int rem=no%10;
          no=no/10;
          ++count;
    }
    return count;
  }
}
public class DyInfApp
{
  public static void main(String x[])
  {
          Num n = new Rev();
          n.setValue(1234);
          int result=n.getResult();
          System.out.println("Reverse Number is "+result);
          n=new CountDigit();
          n.setValue(1234);
          result=n.getResult();
          System.out.println("Digit count is  "+result);

  }
}
```

Note: if interface contain more than one abstract method then all method must be override all methods where interface get implement and if we not required some method we need to override it as blank.

```
interface P
{  void s1();
   void s2();
   void s3();
}
class First implements P
{
  public void s1()
  { System.out.println("I required s1");
  }
  public void s2()
   {
```

```
  }
  public void s3(){
  }
}
class Second implements P
{public void s1()
 {
 }
 public void s2()
  { System.out.println("I required s2");
  }
  public void s3(){
  }
}
public class TestInfApp
{
  public static void main(String x[])
  {
          First f=new First();
          f.s1();
          Second s = new Second();
          s.s2();
  }
}
```

Note: if we want to solve above problem we have solution name as adapter class.

Example using adapter class

```
interface P
{  void s1();
  void s2();
  void s3();
}
class ADP implements P
{
  public void s1(){
  }
  public void s2(){
  }
  public void s3() {
  }
}
class First extends ADP
```

```
{
  public void s1()
  { System.out.println("I required s1");
  }

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

}
public class TestInfApp
{
   public static void main(String x[])
   {
          First f=new First();
          f.s1();
          Second s = new Second();
          s.s2();
   }
}
```

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

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

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

Q. Can we inherit interface to interface?

Yes we can perform inheritance between interface to interface and for that we use extends keyword when we want to inherit one interface to another interface.

```
interface A
{  void show();
}
interface B extends A
{ void display();
}
class C implements B
{
  public void show()
```

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

Rules of inheritance

---

1) If parent is class and child is also class then use extends keyword
2) If parent is interface and child is class use implements keyword
3) If parent is interface and child is also interface then use extends keyword
4) If parent is class and child is interface then there is no inheritance generate compile time error.

Q. Can we declare class within interface?

---

yes can declare class within interface

Example with source code

```
interface T
{  void show();
  class P implements T
  {
    public void show()
    { System.out.println("I am show method");
    }
  }
}
class C extends T.P
{
  public void display()
  { System.out.println("I am display method");
  }
}
public class TestNestInf
```

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

Output

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

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

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

or you can use above code like as

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

Output

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

C:\Program Files\Java\jdk1.8.0_291\bin>java TestNestInf
I am show method of T
I am display method
I am test method of class P

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

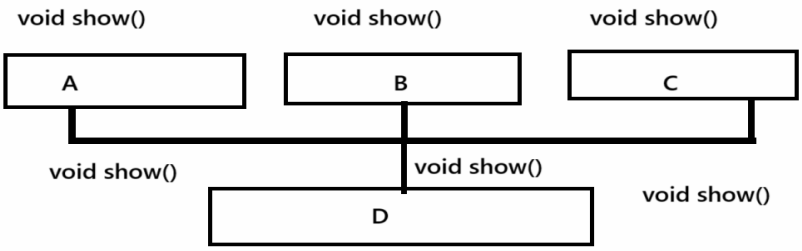
Q. Why java not uses classes for achieve multiple inheritances?

Because of diamond problem

Q. What is diamond problem?

---

diamond problem means in multiple inheritance there is more than one parent classes and single child class so there is possibility different parent may be contain same name method and the method copy available in child class from every parent class and if we create object of child class try to call the method whose name same in different parent then your compiler may be get confused called as diamond problem




```
D d1 = new D();
d1.show();
```

Note: if we think about above diagram we have child class name as D which contain three methods name show() from different parent classes and when we create object of class D and try to call show() then compiler cannot predicate which parent version should execute called as diamond problem and for resolve this problem java suggest us use interface for achieve multiple inheritance.

How to achieve multiple inheritances using interface

So if we think about multiple inheritances there is more than one parent and single child but from multiple parent there must be only one parent class and rest remaining parents should be interface.

Syntax:

class childclassname extends parentclassname implements interface1…..interface…n{

}

```
interface A
{ void show();
}
interface B
{  void show();
}
class C
{  void show()
   { System.out.println("I am show method");
   }
}
class D extends C implements A,B
{
   public void show()
   {  System.out.println("I am show in D");
   }
}
public class MinhApp
{  public static void main(String x[])
   { D d1 = new D();
        d1.show();
   }
}
```

Note: there is ambiguity problem because we have three parent A,B,C and two are interfaces and all parent contain method name as show() and we implement interface A and B in class D as per rule of interface we must be override interface method so we override show() method in class D so here interface A consider his method get override and B consider his own method get override and when we create object of class D i.e child class and call overridden method then by default child logics get executed so there is change of ambiguity so we can say interface resolve diamond problem and can implement multiple inheritance so java us interface for achieve multiple inheritance

Output

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

C:\Program Files\Java\jdk-23\bin>java MinhApp
I am show in D

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

Q. What is difference between interface and abstract class?

Abstract class	**Interface**
Need to use abstract keyword for mark abstract	Internally by default it is abstract not need to use abstract keyword
For create abstract method in abstract class need to use abstract keyword	By default every method is public abstract internally
Variable of abstract class is not static and final by default	Variable in interface are by default public static final
Abstract class is used for achieve partial abstraction	Interface is used for achieve 100% abstraction
Abstract class can have constructor	Interface cannot have constructor
Abstract class need to extends keyword for inherit	Interface can use implements keyword for use
Abstract class cannot implement multiple inheritance	Interface can implement multiple inheritance
Abstract method can declare as protected in abstract class	Abstract method cannot declare as protected in interface.