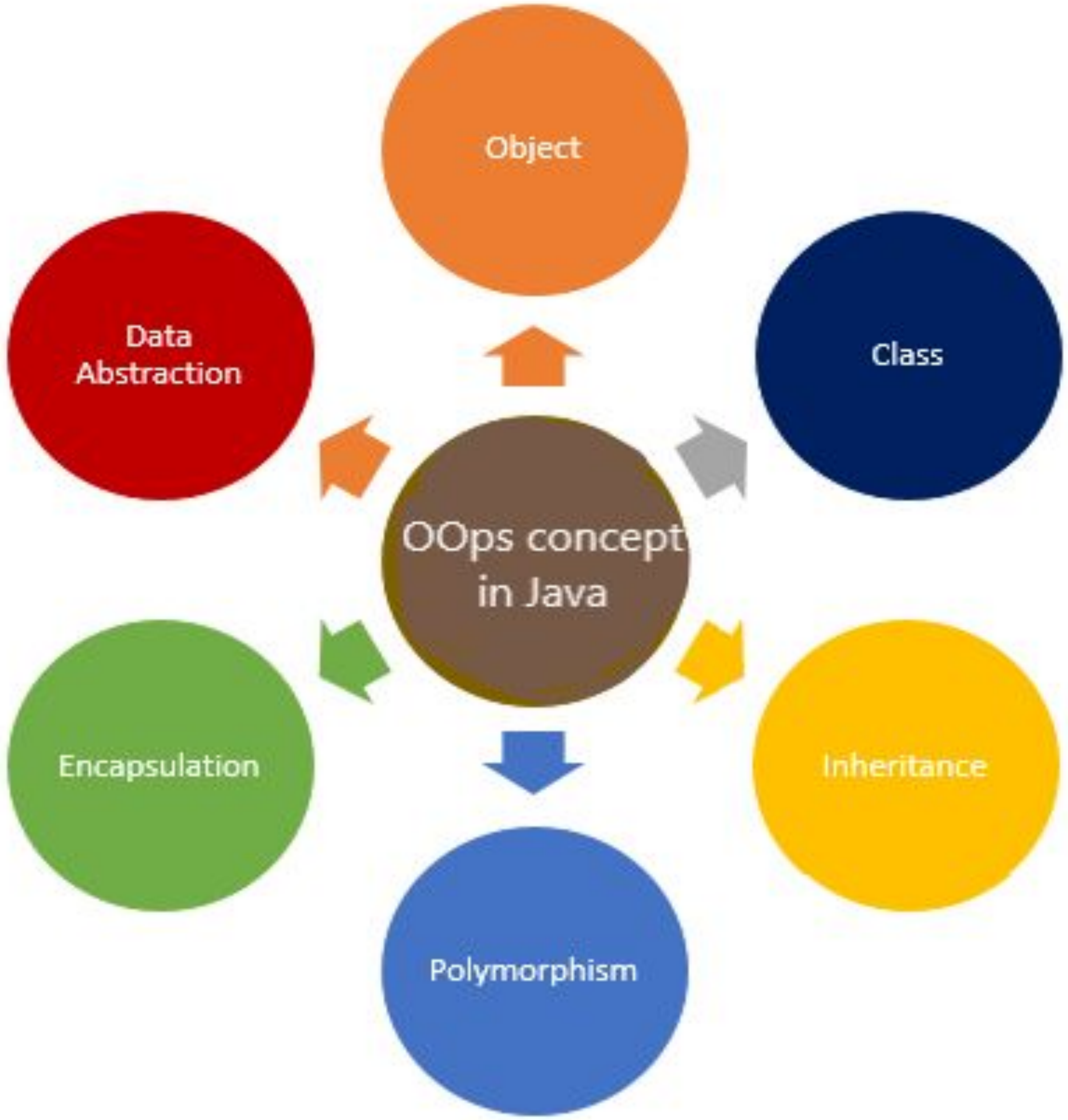


CS20004: Object Oriented Programming using Java

Lec-11

In this Discussion . . .

- Java Package
- Access Modifiers/Access Control Mechanisms
- Inheritance
- super keyword
- References

Java Package

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- However, here we will proceed to creating and using user-defined packages.

Advantages of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

Java Package Syntax

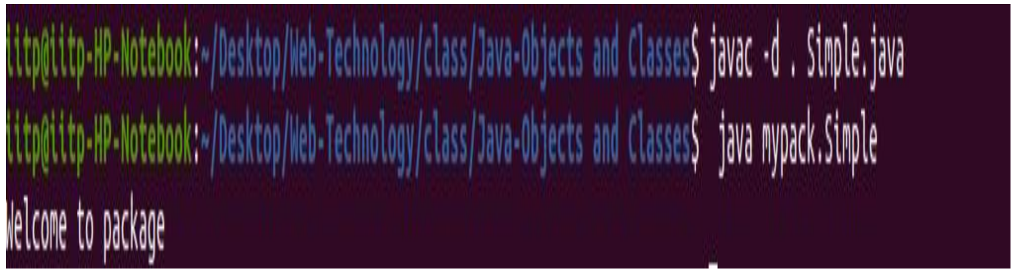
- The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple
{
        public static void main(String args[])
        {
                System.out.println("Welcome to package");
        }
}
```

Java Package Syntax

- The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple
{
        public static void main(String args[])
        {
                System.out.println("Welcome to package");
        }
}
```



```
itp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac -d . Simple.java
itp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$  java mypack.Simple
Welcome to package
```

Steps to ***Compile***: **javac -d . Simple.java** [The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot)]
Run**: java mypack.Simple**

Access Modifiers in Java

- There are two types of modifiers in Java: **access modifiers** and **non-access modifiers.**
- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

Access Modifiers in Java

- There are **four** types of Java access modifiers:
  - **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
  - **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
  - **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
  - **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
- There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc.

Inheritance

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When we inherit from an existing class, we can reuse methods and fields of the parent class. Moreover, we can add new methods and fields in your current class also.
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Why Inheritance?

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability

Common Terms used in Context of Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Subclass or Child class:** Subclass is a class which inherits another class. It is also called a derived class, extended class, or child class.
- **Super Class or Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability**: As the name specifies, reusability is a mechanism which facilitates to reuse the fields and methods of the existing class when new class is created. We can use the same fields and methods already defined in the previous class.

Inheritance in Java

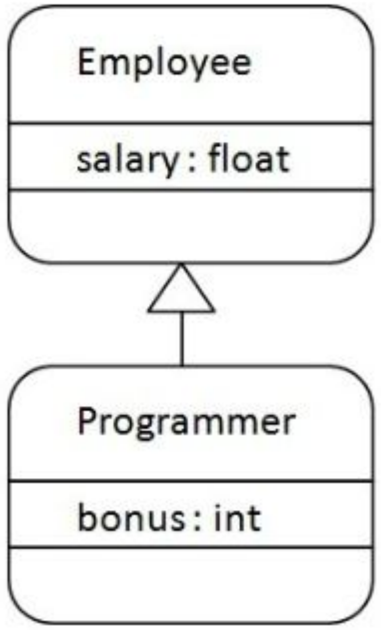
- The **extends** keyword indicates that we are making a new class which is derived from an existing class. Here "extends" means to increase the functionality.

Syntax

```
class Subclass_name extends Superclass_name
{
   //methods and fields
}
```

- In the java's terminology, a class from which is inheritance occurs is called a parent or superclass, while the derived or new class is called child or subclass.
-

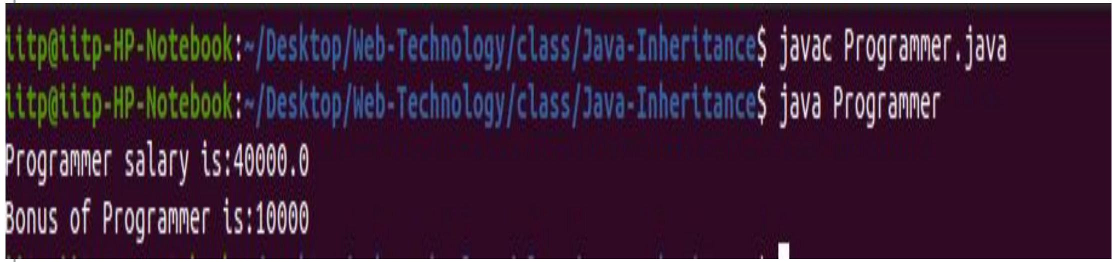
Inheritance in Java



As displayed in the left figure, **Programmer** is the **subclass** and **Employee** is the **superclass**. The relationship between the two classes is ***Programmer IS-A Employee***. It means that Programmer is a type of Employee.

Inheritance in Java

```
class Employee
{
        float salary=40000;
}
class Programmer extends Employee
{
        int bonus=10000;
        public static void main(String args[])
        {
                Programmer p=new Programmer();
                System.out.println("Programmer salary is:"+p.salary);
                System.out.println("Bonus of Programmer is:"+p.bonus);
        }
}
```

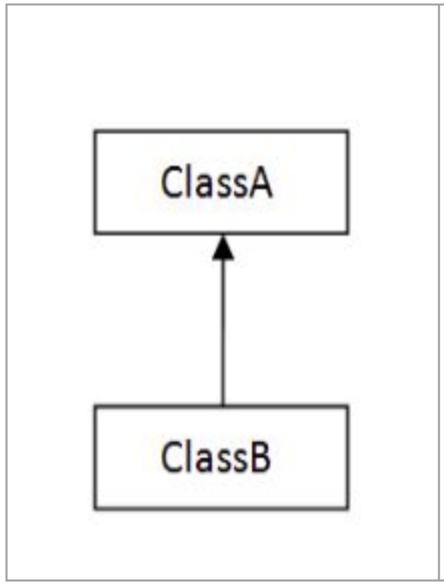


```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Programmer.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Programmer
Programmer salary is:40000.0
Bonus of Programmer is:10000
```

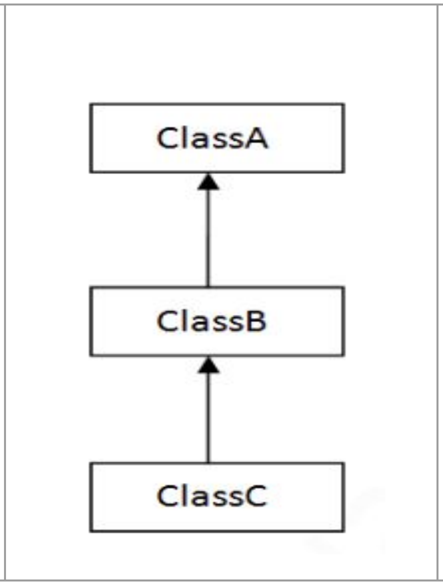
Programmer object “p” can access the field of own class, i.e, **bonus** as well as of Employee class i.e. **salary.**

Inheritance Types

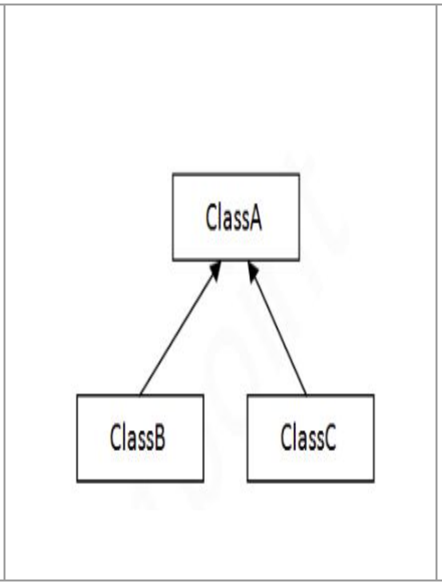
- Depending upon **classes**, there can be **three** types of inheritance in java: single, multilevel and hierarchical. **WHILE** both multiple and hybrid inheritance are supported through **interface** only [Will be covered later when discussing interfaces].
- **Note:** Multiple inheritance is not supported in Java through class. When one class inherits multiple classes, it is known as multiple inheritance.



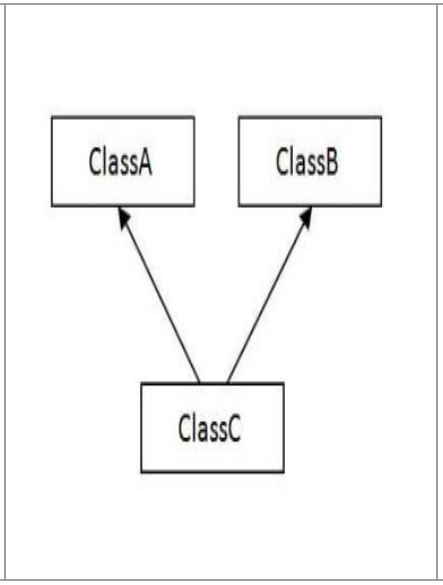
Single



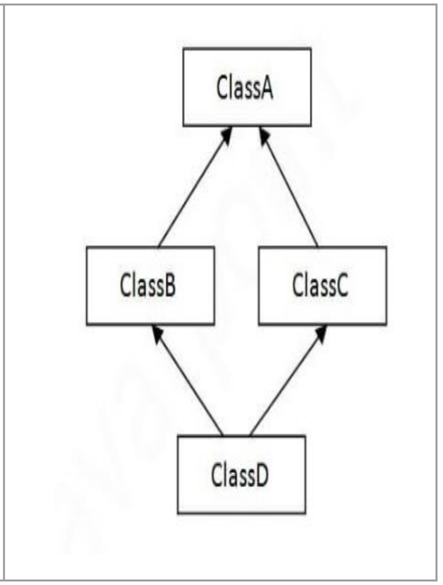
Multilevel



Hierarchical



Multiple



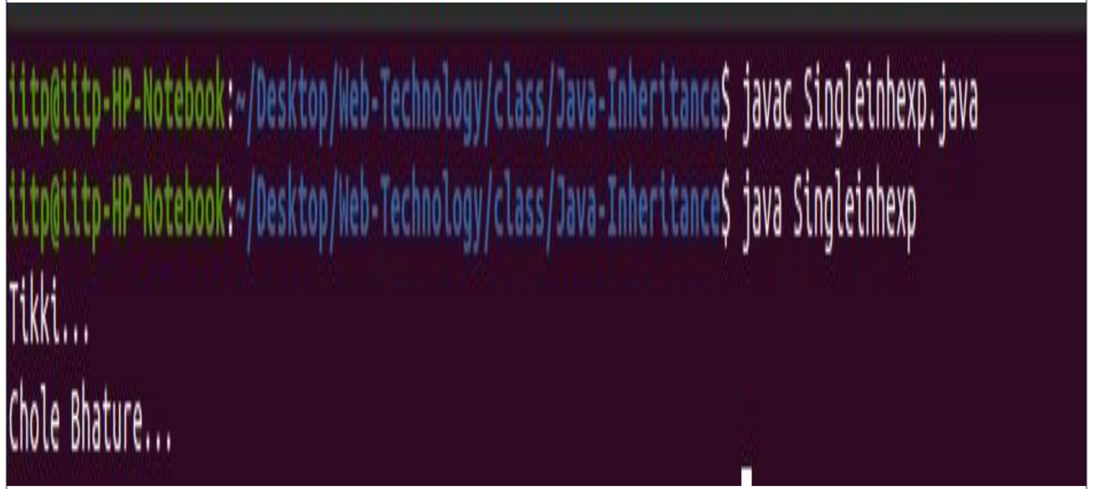
Hybrid

Single Inheritance

- When a class inherits another class, it is known as a ***single inheritance.***

```
class D
{
        void eat()
        {
                System.out.println("Chole Bhature...");
        }
}
class M extends D
{
        void bar()
        {
                System.out.println("Tikki...");
        }
}
class Singleinhexp
{
        public static void main(String args[])
        {
                M m=new M();
                m.bar();
                m.eat();
        }
}
```

Here class M inherits the class D, so there is the single inheritance.



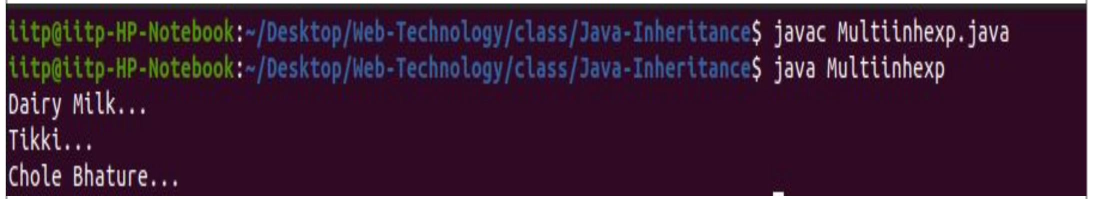
```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Singleinhexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Singleinhexp
Tikki...
Chole Bhature...
```

Multilevel Inheritance

- When there is a chain of inheritance, it is known as *multilevel inheritance*.

```
class D
{
        void eat()
        {System.out.println("Chole Bhature...");
        } }
class M extends D
{
        void bar()
        {System.out.println("Tikki...");
        }  }
class K extends M
{
        void mun()
        {System.out.println("Dairy Milk...");
        }}
class Multiinhexp
{
        public static void main(String args[])
        {  K kk=new K();
                kk.mun();
                kk.bar();
                kk.eat();
        }}
```

Here class K inherits the class M, which in turn inherits the class D, so there is the Multilevel inheritance.



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Multiinhexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Multiinhexp
Dairy Milk...
Tikki...
Chole Bhature...
```

Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

```
class D
{
        void eat()
        {System.out.println("Chole Bhature...");}
}
class M extends D
{
        void bar()
        {System.out.println("Tikki...");}
}
class K extends D
{
        void mun()
        {System.out.println("Dairy Milk...");}
}
class Hierinhexp
{
        public static void main(String args[])
        {
                K kk=new K();
                kk.mun();
                kk.bar();
                kk.eat();  }}
```

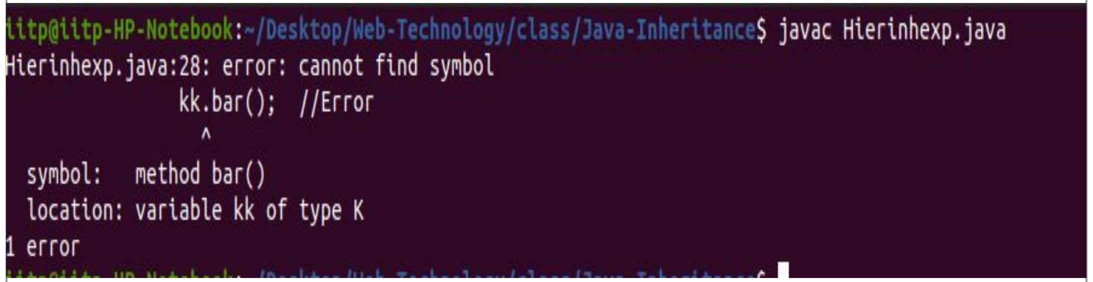
Here both the class K and M inherit from the class D. So, there is the Hierarchical inheritance.

Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

```
class D
{
        void eat()
        {System.out.println("Chole Bhature...");}
}
class M extends D
{
        void bar()
        {System.out.println("Tikki...");}
}
class K extends D
{
        void mun()
        {System.out.println("Dairy Milk...");}
}
class Hierinhexp
{
        public static void main(String args[])
        {
                K kk=new K();
                kk.mun();
                kk.bar();
                kk.eat();  }}
```

Here both the class K and M inherit from the class D. So, there is the Hierarchical inheritance.



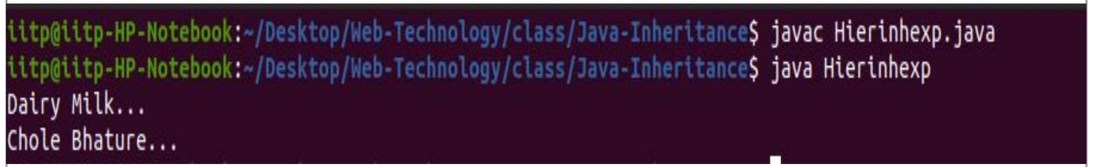
```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Hierinhexp.java
Hierinhexp.java:28: error: cannot find symbol
                kk.bar();  //Error
                  ^
  symbol:   method bar()
  location: variable kk of type K
1 error
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$
```

Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

```
class D
{
        void eat()
        {System.out.println("Chole Bhature...");}
}
class M extends D
{
        void bar()
        {System.out.println("Tikki...");}
}
class K extends D
{
        void mun()
        {System.out.println("Dairy Milk...");}
}
class Hierinhexp
{
        public static void main(String args[])
        {
                K kk=new K();
                kk.mun();
                //kk.bar();
                kk.eat();  }}
```

Here both the class K and M inherit from the class D. So, there is the Hierarchical inheritance.

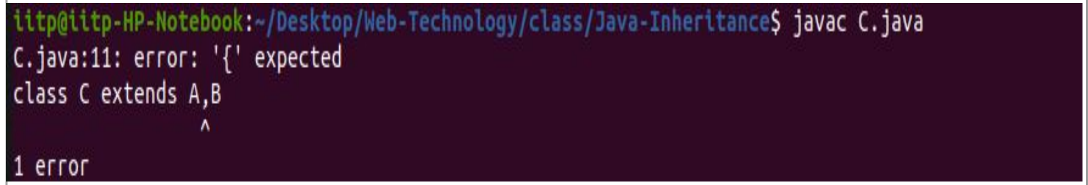


```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Hierinhexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Hierinhexp
Dairy Milk...
Chole Bhature...
```

Why multiple inheritance is not supported in java?

- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and we call it from child class object, there will be ambiguity to call the method of A or B class.
- As compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether we have same method or different, there will be compile time error.

```
class A
{  void eat(){System.out.println("Hello Java");}
}
class B
{  void eat(){System.out.println("Hello Prime");}
}
class C extends A,B{//suppose if it were
public static void main(String args[]){
        C obj=new C();
        obj.eat();//Now which eat() method would be invoked?
} }
```

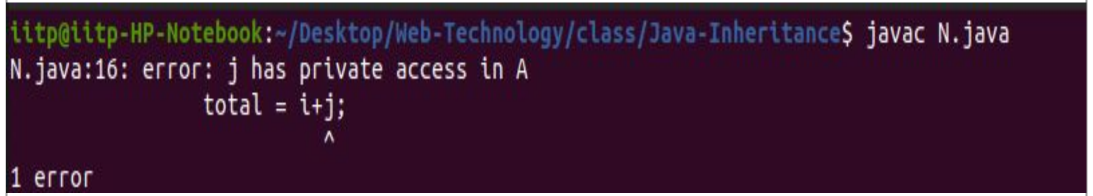


```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac C.java
C.java:11: error: '{' expected
class C extends A,B
                 ^
1 error
```

Inheritance and Private Members

- A class may declare some of its members private
- A subclass has no access to the private members of its super-class

```
class A
{
        int i;
        private int j;
        void setij(int x, int y)
        {
                i = x;
                j = y;
        }
}
class N extends A
{
        int total;
        void sum()
        {
                total = i+j;
        }
}
```



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac N.java
N.java:16: error: j has private access in A
                total = i+j;
                          ^
1 error
```

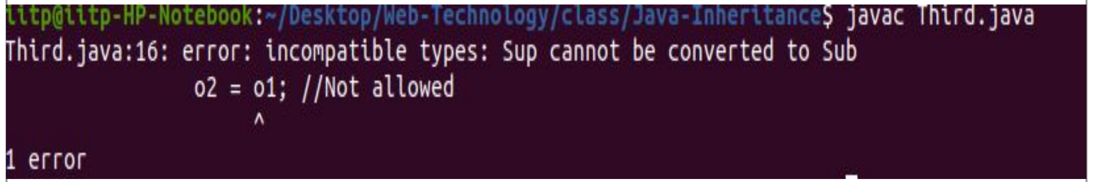
Referencing Subclass objects

- A variable of a super-class type may refer to any of its subclass objects

```
class Sup
{
        //
}
class Sub extends Sup
{
}

class Third
{
        public static void main(String args[])
        {
                Sup o1;
                Sub o2 = new Sub();
                o1 = o2;
        }
}
```

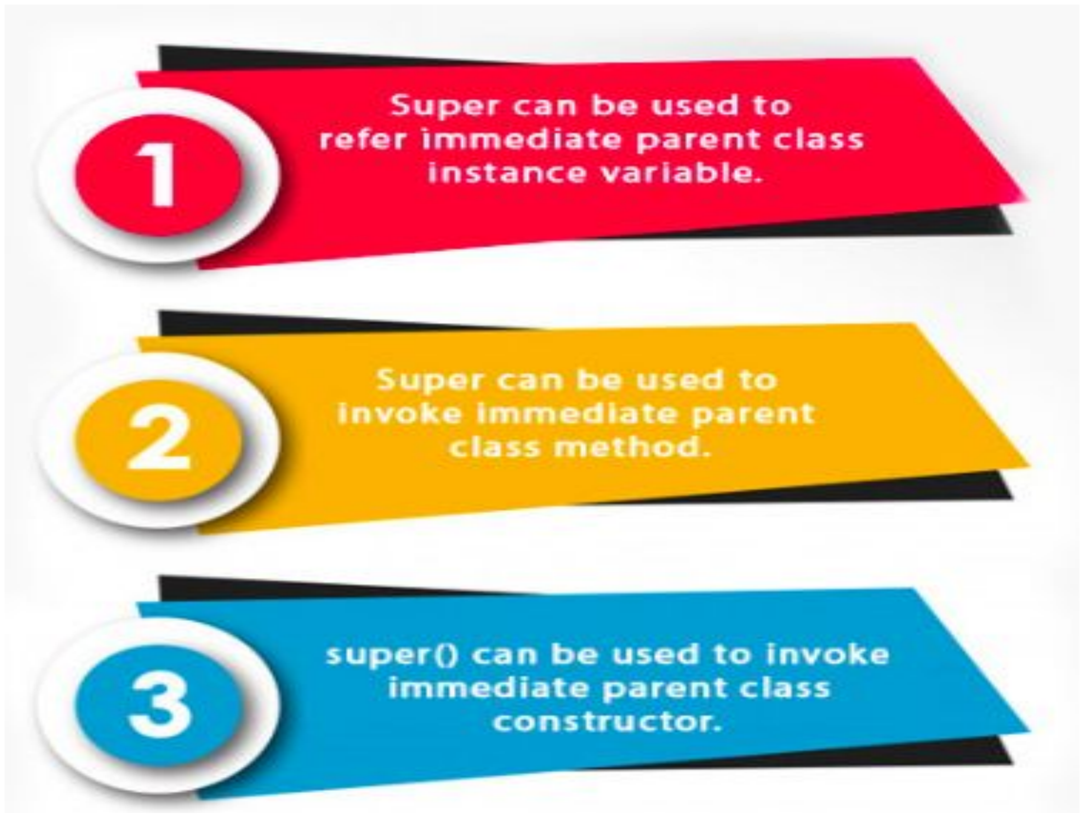
```
class Sup
{
        //
}
class Sub extends Sup
{
}
class Third
{
        public static void main(String args[])
        {
                Sup o1;
                Sub o2 = new Sub();
                o2 = o1; //Not allowed
        }
}
```



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Third.java
Third.java:16: error: incompatible types: Sup cannot be converted to Sub
                o2 = o1; //Not allowed
                     ^
1 error
```

Super keyword

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever we create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
- Usages:
    -

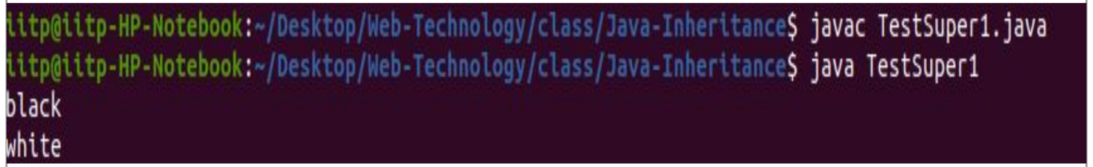

1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

Super keyword : referring immediate parent class instance variable

- We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal
{
        String color="white";
}
class Dog extends Animal
{
        String color="black";
        void printColor()
        {
                System.out.println(color);//prints color of Dog class
                System.out.println(super.color);//prints color of
Animal class
        }
}
class TestSuper1
{
        public static void main(String args[])
        {
                Dog d=new Dog();
                d.printColor();
        }}
```

In the program, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac TestSuper1.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java TestSuper1
black
white
```

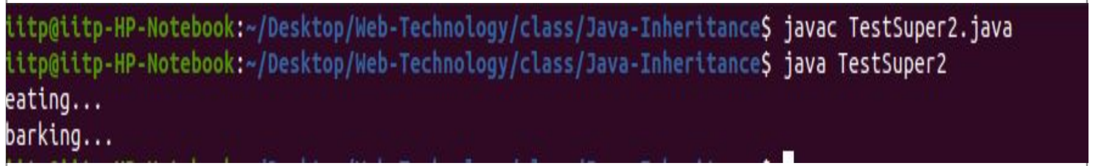
Super keyword : invoke parent class method

- The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal
{
        void eat()
        {System.out.println("eating...");
        }  }
class Dog extends Animal
{
        void eat()
        {System.out.println("eating bread...");}
        void bark()
        {System.out.println("barking...");}
        void work()
        {  super.eat();
                bark();  }
}
class TestSuper2
{
        public static void main(String args[])
        {
                Dog d=new Dog();
                d.work();
        }}
```

In the program Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.



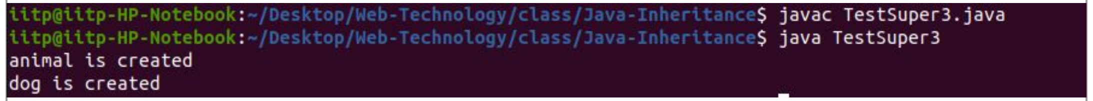
```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac TestSuper2.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java TestSuper2
eating...
barking...
```

Super keyword : invoke parent class constructor.

- The super keyword can also be used to invoke the parent class constructor.

  Note: super() is added in each class constructor automatically by compiler if there is no super() or this().

```
class Animal
{
        Animal()
        {System.out.println("animal is created");}  }
class Dog extends Animal
{
        Dog()
        {
                super();  System.out.println("dog is created");
        }
}
class TestSuper3
{
        public static void main(String args[])
        {  Dog d=new Dog();  }}
```

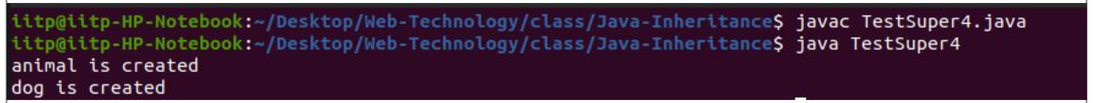


```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac TestSuper3.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java TestSuper3
animal is created
dog is created
```

As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

Super keyword : super() is provided by the compiler implicitly.

```
class Animal
{
        Animal()
        {
                System.out.println("animal is created");
        }
}
class Dog extends Animal
{
        Dog()
        {
                System.out.println("dog is created");
        }
}
class TestSuper4
{
        public static void main(String args[])
        {
                Dog d=new Dog();
        }}
```



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac TestSuper4.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java TestSuper4
animal is created
dog is created
```

References

1. https://www.javatpoint.com/super-keyword
2. https://www.javatpoint.com/inheritance-in-java
3.