Below are the relationships that can exist between classes

1.Composition or **has a** or **Container** relationship :

```
class Employee{
int I;
Address obj;   / /Address is another class

void met1(){
//...statements
}
}
```

Above is has a relationship between class *Employee* and class *Address*, i..e class *Employee* has class *Address*

Inheritance Relationship:

Another relationship that can exist between classes is Inheritance relationship.

One class inherits data members and member methods of another class. Inheritance is also called is-a relationship. Or Generalization

extends keyword is used in inheritance

Class which is being inherited/derived is called Base class or Parent class or Super class.

Class which inherits from Base class is called Derived class or Child class or Sub class.

Advantage of Inheritance:

Improves code reusability.

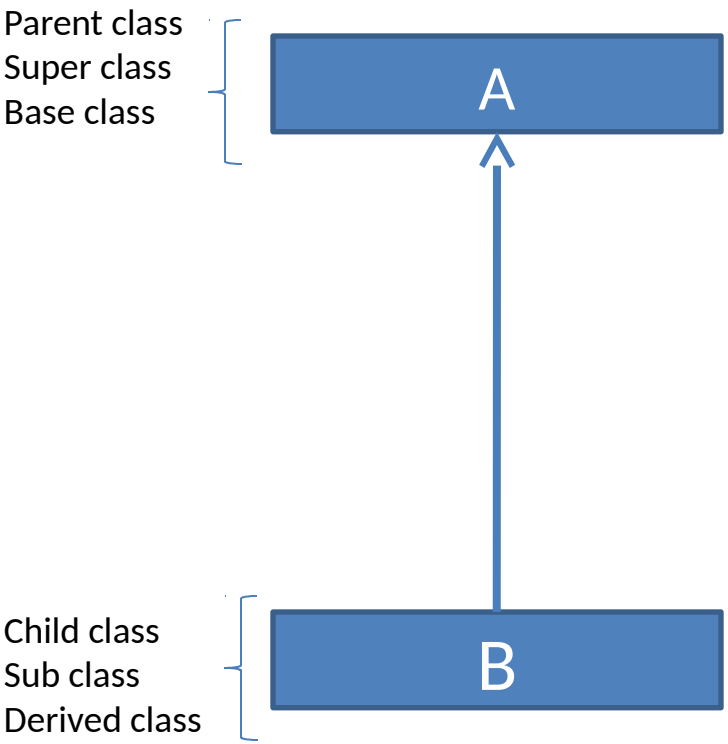
Decrease duplication of code.

Inheritance provides one more access specifier, which is protected

Below are different types of inheritance

1. Single or simple inheritance.
2. Multi Level inheritance
3. Hierarchical inheritance
4. Hybrid Inheritance
5. Multiple Inheritance(Not supported in Java)

Simple or Single inheritance: One class is derived from another class. Here class A is base class, class B is derived class. That means there is only one Base class and one Derived class in Single inheritance.



Syntax:

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

class B extends A{
//...
}
```

NOTE: class B has direct access to all non private members of class A .Arrow shows that B is dependent on A

Order of invocation of Constructors: When an object of B is created, first the constructor of A gets invoked(automatically), and then constructor of B is invoked.

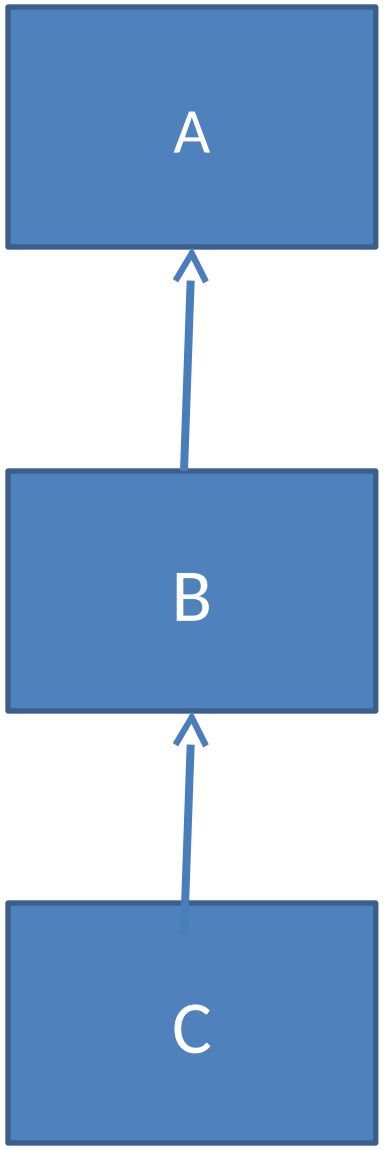
Multi Level inheritance: A Derived class acts as a Base class of some other class

Syntax:

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

class B extends A{
//...
}

class C extends B{
//....
}
```



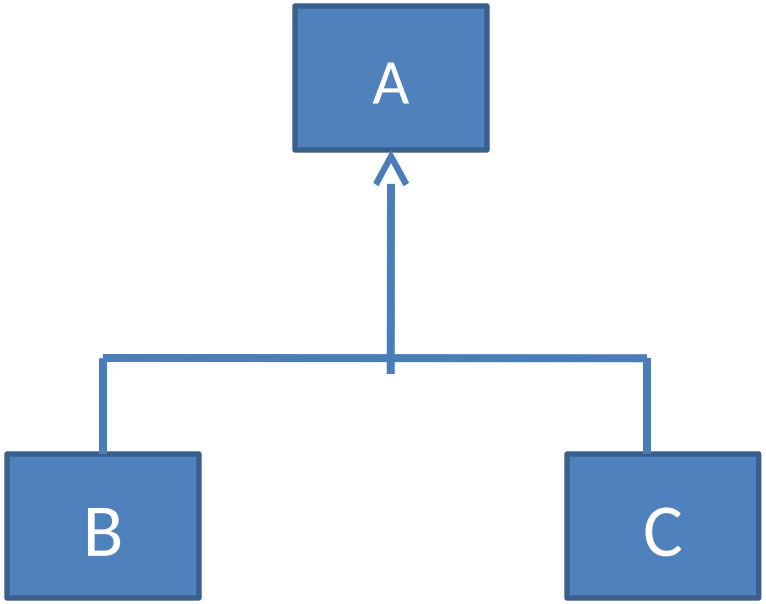
Order of invocation of Constructors:

When an object of C is created, first constructor A is called, then constructor B is called, and then constructor C is called.

Even though you explicitly invoke constructor of only class C, class C is dependent on class B, class B is dependent on class A.

NOTE: There can be even 4 or more levels

Hierarchical inheritance: More than one derived class has a common Base class



Syntax:

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

class B extends A{
//...
}

class C extends A{
//....
}
```

Note:

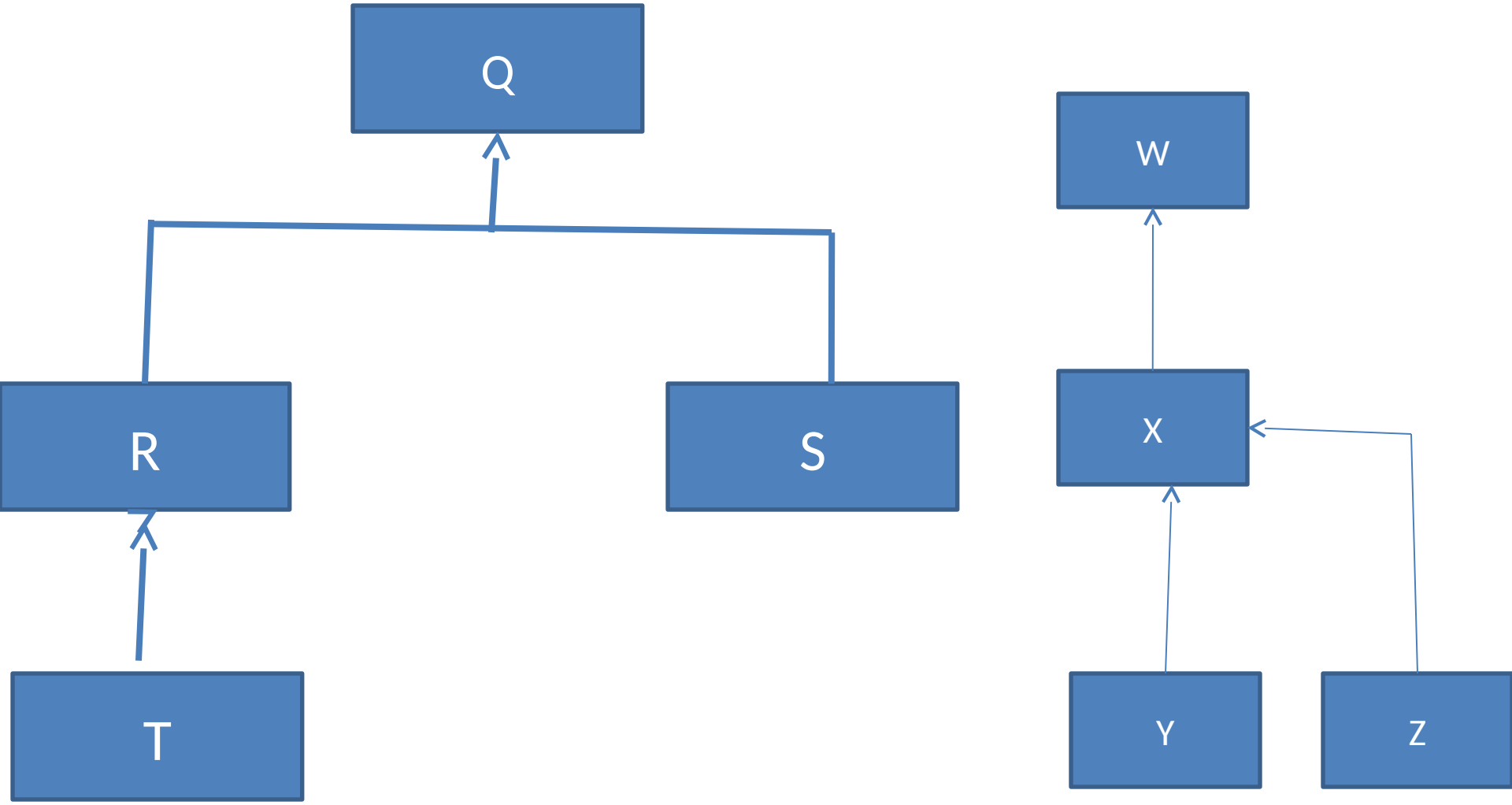
Class C and class B are not dependent on each other.

Relationship between B and C is siblings.

In any type of inheritance, all the classes having inheritance relationship can be spread across multiple .java files or even multiple packages.

Static members are also inherited.

Hybrid inheritance: combination of more than one inheritance. In below example Multi level inheritance exist between Q, Rand T. And Hierarchical inheritance exist between Q, R and S.



Protected Access specifier

A class member declared as protected, can be accessed within the class, and in derived class, and within current package

	private	No modifier (default)	protected	public
Within class	yes	yes	yes	yes
Same package subclass	no	yes	yes	yes
Same package non sub class	no	yes	yes	yes
Different package sub class	no	no	yes	yes
Different package non sub class	no	no	no	yes

final keyword (in Inheritance)

As already known final keyword can be used with data member or local variable/object. There final was used to declare constants.

final keyword can also be used with a class or a method.

A class declared as final can never be a base class. For eg. Inbuilt String class is final, also wrapper classes like Integer, Float, etc… are final

A method declared as final cannot be overridden by derived classes.

NOTE:

1.interface or interface methods cannot be declared final

2.Inner class also can be declared final.

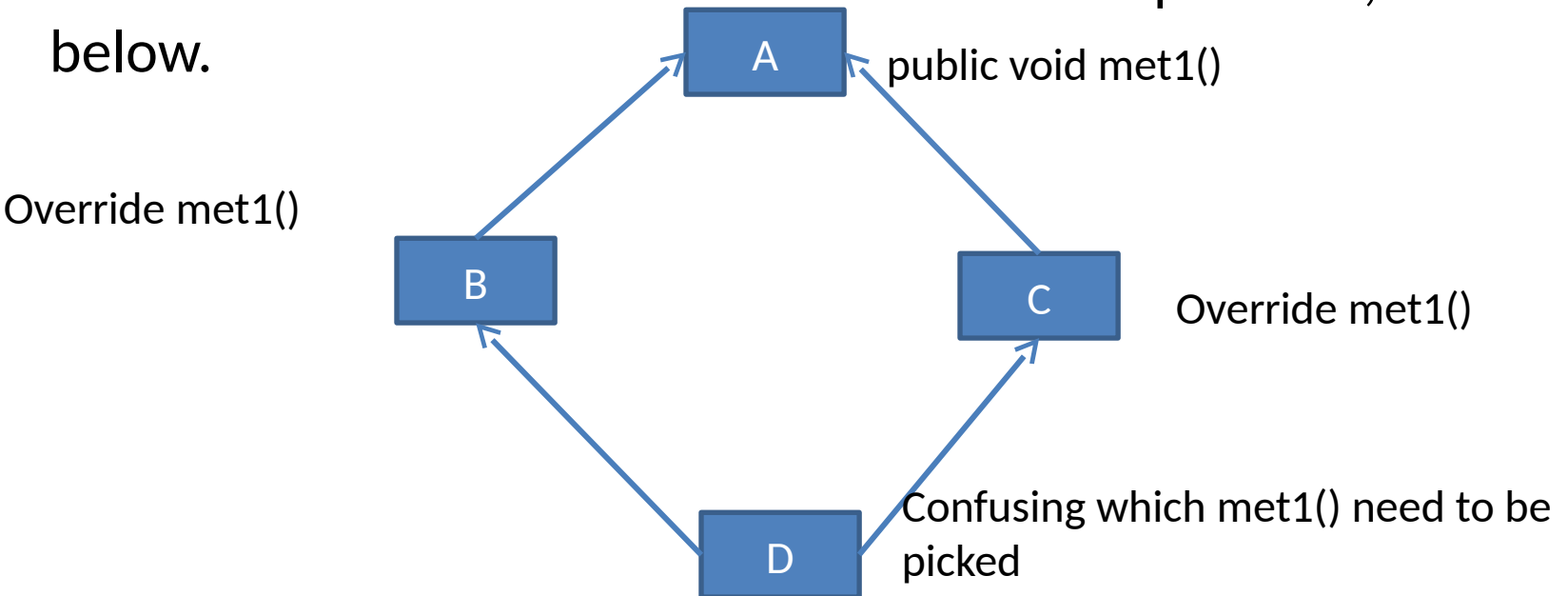
Constructor cannot be final

A class or a method can be made final due to Business or Technical reasons(like optimization, etc…)

Why multiple inheritance not supported in Java?

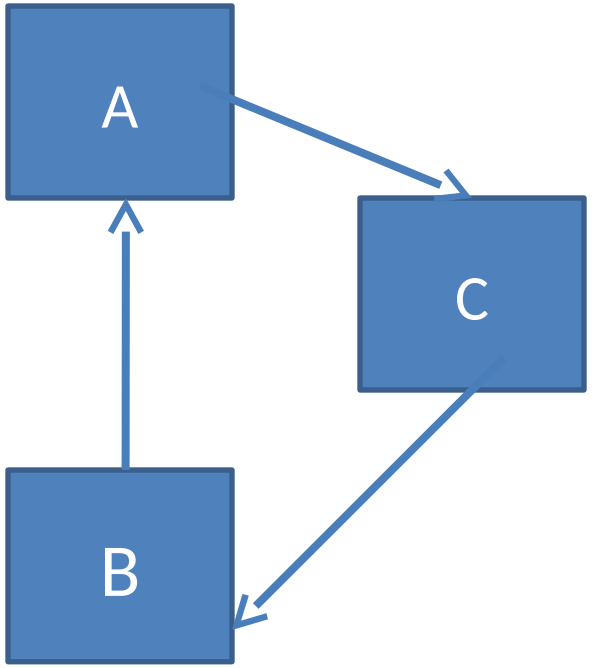
Multiple inheritance is not supported in Java, due to ambiguity reasons, and almost there is nothing which cannot be achieved without Multiple inheritance.

ambiguity with Diamond problem, consider a class A has met1() method and then B and C derived from A and has their own met1() implementation and now class D derive from B and C using multiple inheritance and if we refer just met1() compiler will not be able to decide which met1() it should invoke. This is also called Diamond problem, as shown below.



Cyclic relationship not allowed in inheritance

As shown in beside diagram, Cyclic dependency is not allowed in inheritance. Java Compiler give error, when such cycles exist in your source code.



Type casting between class, and return, parameter passage

As known, a Base class reference can refer Derived class object.
A method accepting an object as parameter, allows objects of any derived classes as parameter.
The same applies to return, and composition relationship, as well.

super keyword is used

Super keyword

1.To invoke constructor of immediate Base class

super(); //parameters also can be passed

super(10,”Naveen”);

super() can be invoked only from constructors, and that too it should be first line of constructor. super() is mandatory, when default/implicit constructor, is not available in Base class

2.To avoid same name collisions between members of the class and members of immediate Base class super.i = i;

super.show();

super is just like this keyword. this keyword refers to the object of current class, where as super refers to object of immediate base class

super keyword cannot be used in static methods.

Object class

Object class(in java.lang package) is base class of all classes in Java, by default. Java provides basic functionality required by each and every class thru methods of Object class.

Below are methods provided in Object class.

Object class

boolean equals(Object obj); -- checks if the objects are equal

int hashCode(); -- returns unique hash code of the object, which is used to identify an object, if required.

final Class getClass(); -- returns Class object of the class

String toString(); -- returns String format of the object

final void wait();

final void wait(int seconds);

final void wait(long milliseconds);

final void notify();

final void notifyAll();

Used for thread communication

void finalize(); -- invoked by **Garbage Collector**, when destroying the object

Since Object class is base class of all classes in Java, pls mind that below statements are valid.

1. Object obj = any_object;

i..e a reference of an Object class can refer to object of any class.

2. Similarly a method accepting Object as parameter, can accept object of any class as parameter.

3. An object of Object class can be type casted to object of any class.

A oa = (A)Obj;

Base class reference

Base class reference can refer to Derived class object. The reason is Derived class object is of Base class type. Hence if A is Base class, and B is derived class, below is valid

A obj = new B(); *//valid*

B ob = new A(); //invalid

Method Overriding

When Base and Derived classes has a method with same declaration(i..e same name, parameter types, and return types). Then the method which gets invoked(with base class reference) in run time depends on the object to which base class reference is pointing to.

This is also called **Run time Polymorphism**, or **Dynamic Method dispatching**. **Here the method which need to be invoked is decided in run time and not during compile time.**(Note that method overloading is Compile time Polymorphism)

When Base and Derived classes have a method with same name, but with different parameters, they participate in method overloading and not in overriding.

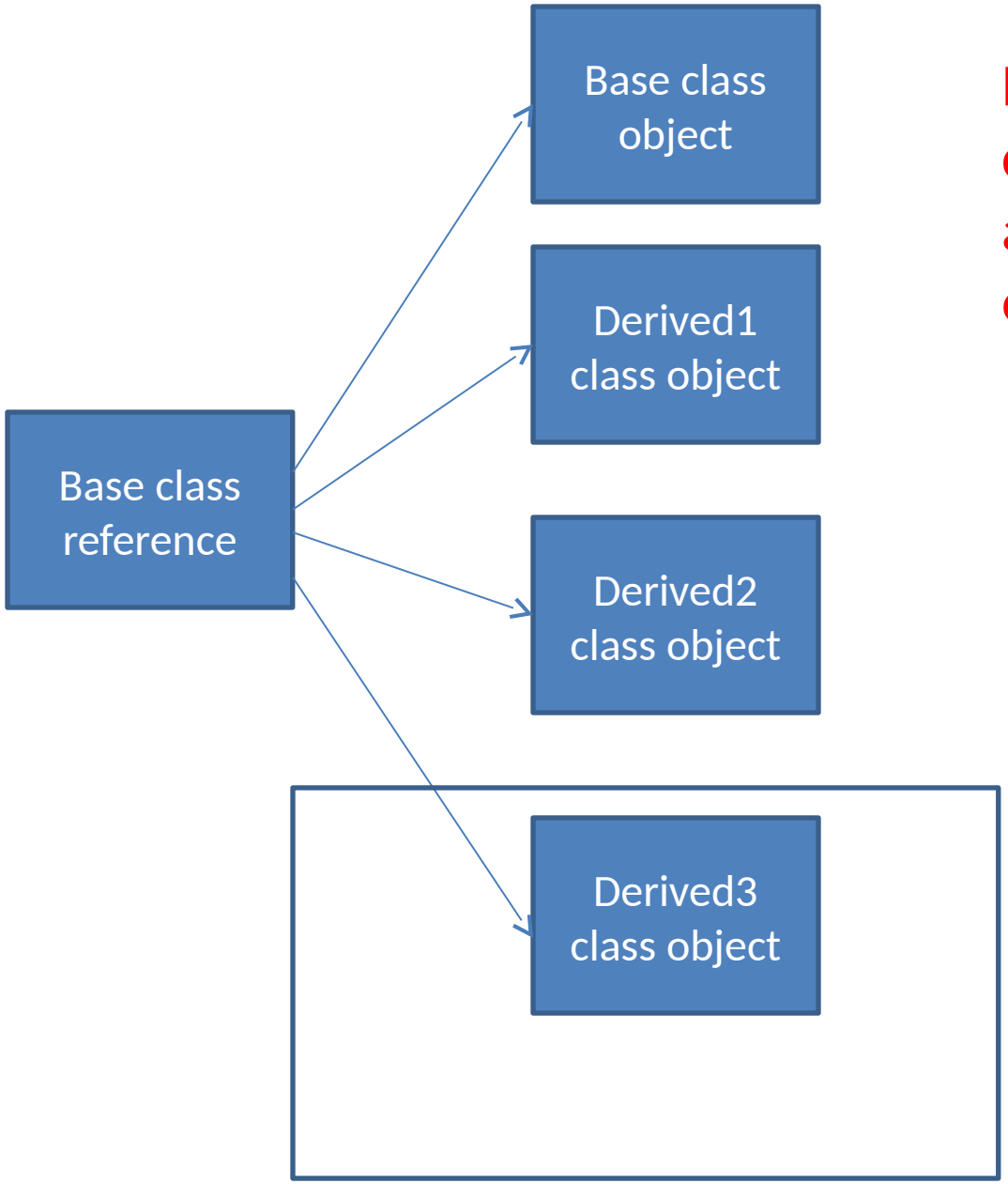
Method Overiding helps to dynamically invoke methods of any Base class without compiling with Base class together.

Here Base and Derived classes can be developed by different developers, they can work together without compiling them together, and by just sharing binary files.

Below is valid Syntax

```
class Q{
Q obj;

}
```



Base class reference can refer to itself or any Derived class object.

abstract keyword

abstract keyword can be used with methods or classes. An abstract method is a method, which has only declaration, and method definition(or body) need to be provided by the derived class.
An abstract class, cannot be instantiated, as it is not complete.

An abstract class can have
constructor.
private, public, protected, default, final, static members

Though abstract class cannot be instantiated, it is possible to use it as reference.
Extends is used between classes
Implements is used between an interface and a class
NOTE: An interface and methods declared in interface are abstract by default. Hence mentioning abstract keyword for them makes no difference.
An interface cannot have constructor, private, protected members.
An interface is a pure abstract class

final keyword

Final keyword can be used with
any variable or objects

A method declared as final cannot be overridden. For eg. wait(), notify(), notifyAll() methods in java.lang.Object are final and cannot be overridden.

A class declared as final cannot be a Base class. For eg. String class is final, and cannot be overridden

Static method and overriding

static methods are not overridden, and do not take part in run time polymorphism

One interface can be extended from one or more interfaces, hence multiple inheritance is supported in Java.

Java Reflection

Reflection is a feature in Java, using which it is possible to analyze characteristics of any object or class during run time. It is possible to find number of constructors, methods etc… and even invoke the methods, during run time. This feature may be useful to load a library and analyze, dynamically during run time.