

Regarding inheritance

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called inheritance.

In OOP, we often organize classes in hierarchy to avoid duplication and reduce redundancy. The classes in the lower hierarchy inherit all the variables (attributes/state) and methods (dynamic behaviors) from the higher hierarchies.

A class in the lower hierarchy is called a subclass (or derived, child, extended class). A class in the upper hierarchy is called a superclass (or base, parent class).

By pulling out all the common variables and methods into the superclasses, and leave the specialized variables and methods in the subclasses, redundancy can be greatly reduced or eliminated as these common variables and methods do not need to be repeated in all the subclasses. Re usability is maximum.

A subclass inherits all the member variables and methods from its superclasses (the immediate parent and all its ancestors). It can use the inherited methods and variables as they are. It may also override an inherited method by providing its own version, or hide an inherited variable by defining a variable of the same name.

Inheritance is a process from generalization ----> specialization.

It represents : IS A Relationship.

Why -- re usability.

Which is the keyword used for inheritance ? --extends

Summary : Sub class IS-A super class , and something more (additional state + additional methods) and something modified(behaviour --- method overriding)

eg :

Person,Student,Faculty

Emp,Manager,SalesManager,HRManager,Worker,TempWorker,PermanentWorker

Shape, Circle,Rectangle,Cylinder,Cuboid

BankAccount ,LoanAccount,HomeLoanAccount,VehicleLoanAccount,

Student,GradStudent,PostGradStudent

Fruit -- Apple -- FujiApple

A subclass inherits all the variables and methods from its superclasses, including its immediate parent as well as all the ancestors.

It is important to note that a subclass is not a "subset" of a superclass. In contrast, subclass is a "superset" of a superclass. It is because a subclass inherits all the variables and methods of the superclass; in addition, it extends the superclass by providing more variables and methods.

Types of inheritance

1. Single inheritance ---Supported

eg : class A {...} class B extends A{...}

2. Multi level inheritance ---Supported

eg : class A{...} class B extends A{...} class C extends B{...}

3. Hierarchical inheritance ---Supported

When more than one classes inherit a same class then this is called hierarchical inheritance.

eg : class A{..} class B extends A{ ..} class C extends A{...} class D extends A {...}

4. Multiple inheritance --- NOT supported

class A extends B,C{...} --provided B & C : classes -- compiler err

Why --For simplicity.

(Diamond problem)

We have two classes B and C inheriting from A. Assume that B and C are overriding an inherited method and they provide their own implementation. Now D inherits from both B and C doing multiple inheritance. D should inherit that overridden method. BUT which overridden method will be used? Will it be from B or C? Here we have an ambiguity. This is known as Diamond Problem.

To avoid this problem , Java does not support multiple inheritance through classes.

Constructor invocations in inheritance hierarchy -- single & multi level.

eg -- Based on class A -- super class & B its sub class.

Further extend it by class C as a sub-class of B.

super keyword usage

1. To access super class's visible members(data members n methods)

eg : p1 : package

```
class A { void show(){sop("in A's show");}}
```

package p1 :

```
class B extends A {
```

```
    //overriding form /sub class version
```

```
    void show(){sop("in B's show");
```

```
        super.show();
```

```
}
```

```
}
```

eg : B b1=new B();

b1.show();

2. To invoke immediate super class's matching constructor --- accessible only from sub class

constructor.(super(...))

eg : Organize following in suitable class hierarchy(under "inheritance" package) : tight encapsulation

Person -- firstName,lastName

Student --firstName,lastName,grad year,course,fees,marks

Faculty -- firstName,lastName,yrs of experience , sme

Confirm invocation of constructors & super.

Regarding this & super keywords

this(...) implies invoking constructor from the same class.

super(...) implies invoking constructor from the immediate super class

1. Only a constr can use this(...) or super(..)
2. It has to be 1st statement in the constructor
3. Any constructor can never have both ie. this() & super()
4. super & this (w/o brackets) are used to access (visible) members of super class or the same class.

Enter polymorphism

Polymorphism ---one functionality --multiple (changing) forms

What is method binding ?

Linking a method call to actual method definition

1. static -- compile time polymorphism--early binding ---resolved by javac.

Achieved via method overloading

rules -- can be in same class or in sub classes.

same method name

signature -- different (no of arguments/type of args/both)

ret type --- ignored by compiler.

eg --- In class Test :

```
void test(int i,int j){...}
```

```
void test(int i) {...}
```

```
void test(double i){..}
```

```
void test(int i,double j,boolean flag){..}
```

```
int test(int a,int b){...} //javac error
```

Can you overload static methods ? Yes (eg : Arrays.toString, PrintStream.print/println)

2. Dynamic(run time) polymorphism --- late binding --- dynamic method dispatch ---resolved by JRE.

Dynamic method dispatch -- which form of method to send for execution ---This decision can't be taken by javac(since method signature : SAME) --- BUT taken by JRE (JVM)
Achieved via -- method overriding

Method Overriding --- Means of achieving run-time polymorphism

What is it ?

When we declare the same method in child class which is already present in the parent class then this is called method overriding. In this case when we call the method from child class object, the child class version of the method is called.

eg : Fruit class : taste() : "No Specific Taste"

Orange extends Fruit : taste() : "Sour in taste"

Orange f=new Orange(...);

sop(f.taste());/"Sour in taste"

Important points

NO "virtual" keyword in java. (i.e all methods are implicitly virtual)

All java methods can be overridden to achieve run time polymorphism : if they are not marked as private or static or final

Super-class form of method - --- overridden method

sub-class form --- overriding form of the method

Rules : to be followed by overriding method in a sub-class

1. same method name, same signature, ret type must be same or its sub-type(co-variance)

eg of co-variance

```
package p1;
class A {
    A getInstance()
    {
        return new A();
    }
}
package p1;
class B extends A
{
    B getInstance()
    {
        return new B();
    }
}
```

2. scope---must be same or wider.

3. Will be discussed in exception handling.

Can not add in its throws clause any new or broader checked exceptions.

BUT can add any new unchecked excs.

Can add any subset or sub-class of checked excs.

```
class A
{
    void show() throws IOExc
    {...}
}
class B extends A
{
    void show() throws Exc
    {...}
}
```

Can't add super class of the checked excs.

Annotations

From JDK 1.5 onwards : Annotations are available --- metadata meant for Compiler or JRE.(Java tools)

Java Annotation is a tag that represents the metadata i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in java are used to provide additional information, so it is an alternative option for XML.

eg @Override,@Deprecated,@SuppressWarnings,@FunctionalInterface.....

@Override --

It is an annotation meant for javac.

It's Method level annotation ,that appears in a sub class

It's Optional BUT recommended.

eg :

```
public class Orange extends Fruit {
```

```
@Override
public void taste(String name) {...} //javac err
}
}
```

Meaning

While overriding the method --- if you want to inform the compiler that : following is the overriding form of the method use :

@Override

method declaration {...}

Run time polymorphism or Dynamic method dispatch in detail

Super -class ref. can directly refer to sub-class object(direct=w/o type casting) as its the example of up-casting(similar to widening auto. conversion) .

When such a super class ref is used to invoke the overriding method: which form of the method to send for execution : this decision is taken by JRE & not by compiler. In such case --- overriding form of the method(sub-class version) will be dispatched for exec.

Javac resolves the method binding by the type of the reference & JVM resolves the method binding by type of the object it's referring to.

Super -class ref. can directly refer to sub-class inst BUT it can only access the members declared in super-class -- directly.

eg : A ref=new B(); ref.show() ---> this will invoke the sub-class: overriding form of the show () method

Applying inheritance & polymorphism

Important statement

Java compiler resolves method binding by type of the reference & JVM resolves it by the type of the object, reference is referring to.

java.lang.Object --- Universal super class of all java classes including arrays.

Object class method

public String toString() ---Returns string representation of object.

Returns --- Fully qualified class Name @ hash code

hash code --internal memory representation.(hash code is mainly used in hashing based data structures -- will be done in Collection framework)

Why override toString?

To replace hash code version by actual details of any object.

Objective -- Use it in sub classes. (override toString to display Account or Point2D or Emp details or Student / Faculty)

Object class method

public boolean equals(Object o)

Returns true --- If 'this' (invoker ref) & o ---refers to the same object(i.e reference equality) i.e this==o , otherwise returns false.

Need of overriding equals method ?

To replace reference equality by content identity equality , based upon prim key criteria.

eg : In Car scenario

(Primary key -- int registration no)

Objective : use it for understanding downcasting n instanceof keyword

instanceof -- keyword in java --used for testing run time type information(RTTI)

refer : regarding instanceof

Special note on protected

Protected members act as default scope within the same package.

BUT outside pkg -- a sub-class can access it through inheritance(i.e just inherits it directly) & CAN'T be accessed by creating super class instance.

Do subclasses inherit private data members from it's superclass?

NO !

Explanation :

As per the java language specification :

Members of a class that are declared private are not inherited by subclasses of that class. Only members of a class that are declared protected or public are inherited by subclasses declared in a package other than the one in which the class is declared.

BUT what we mean here by inheritance is , are private members accessible in a subclass ?

That answer is NO

BUT , sub class instance DOES CONTAIN private fields of their superclasses .

eg : Person has data members : private firstName , lastName

Student extends Person

It has ADDITIONAL data members :private gradYear,course,fees,marks

Answer this !

Can you access firstName & lastName from Student class ? NO

Student IS-A Person

So when you create an instance of a Student : firstName n lastName will be present in Student object , mem allocated in heap.

So how many slots will you show in Student object ?

CP + 6 slots .

Upcasting

The most important aspect of inheritance is the relationship expressed between the new class and the base class. This relationship can be summarized by saying,

The new class "IS A" type of the existing class.

eg : Student is of Person type or Faculty is of Person type.

This description is not just a fancy way of explaining inheritance—it's supported directly by the language.

Meaning :

Can we say ?

Person p=new Student(...); //YES --upcasting

sop(p); //dynamic method dispatch

As another example, consider a base class called Fruit that represents any fruit, and a derived class called Mango. Because inheritance means that all of the methods in the base class are also available in the derived class, any message you can send to the base class can also be sent to the derived class. If the Fruit class has a taste() method, so will Mango.

This means we can accurately say that a Mango object is also a type of Fruit.

Why static methods can't be overridden in java ?

Method overriding is a way to achieve dynamic method dispatch(i.e run time polymorphism)

Meaning which behaviour to choose or which method to choose for execution --this decision is taken at the run time depending upon type of the object by the JVM(late binding). Since it depends upon the type of the object , for static methods this concept is not applicable. (since they are not associated with any object)

Overriding depends on having an instance of a class. The point of polymorphism is that you can subclass a class and the objects implementing those subclasses will have different behaviors for the same methods defined in the superclass (and overridden in the subclasses). A static method is not associated with any instance of a class so the concept is not applicable.