Chapter 8

Inheritance

Declaration

- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

Syllabus

- Inheritance in OO design

Topics

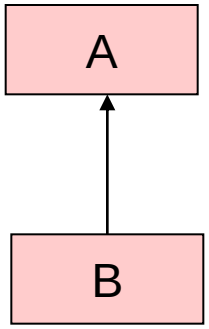
- Inheritance
- Types
- Super
- When Constructor are called
- Method Overridding
- Dynamik Method Dispatch
- Abstract class
- final and inheritance
- The object class

Inheritance

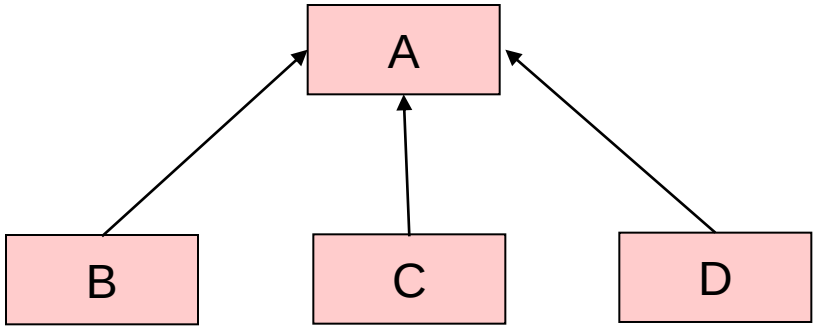
- The mechanism of deriving one class from an old one is called inheritance.
- The old class is known as base class or super class or parent class.
- The new class is called the sub class or derived class or child class.

Types

- Single inheritance



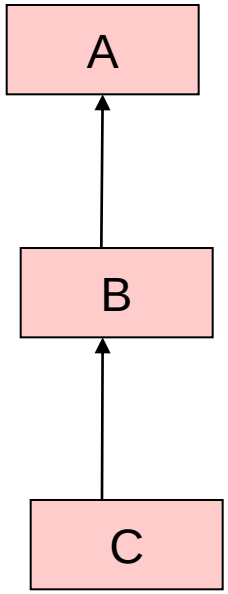
Types



- Hierarchical inheritance is a kind of inheritance where more than one class is inherited from a single parent or base class.

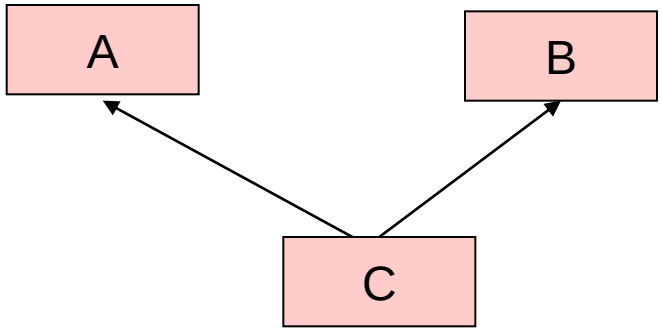
Types

- Multilevel inheritance



Types

- Multiple inheritance



- Java does not support multiple inheritance.

Example

```
// A simple example of inheritance.

// Create a superclass.
class A
{
  int i, j;

  void showij()
  {
    System.out.println("i and j: " + i + " " + j);
  }
}
```

Example

```
// Create a subclass by extending class A.
class B extends A
{
  int k;

  void showk()
  {
    System.out.println("k: " + k);
  }
  void sum()
  {
    System.out.println("i+j+k: " + (i+j+k));
  }
}
```

Example

```
class SimpleInheritance
{
  public static void main(String args[])
  {
    A superOb = new A();
    B subOb = new B();

    // The superclass may be used by itself.
    superOb.i = 10;
    superOb.j = 20;
    System.out.println("Contents of superOb: ");
    superOb.showij();
    System.out.println();
```

Example

```
/* The subclass has access to all public members of
      its superclass. */
   subOb.i = 7;
   subOb.j = 8;
   subOb.k = 9;
   System.out.println("Contents of subOb: ");
   subOb.showij();
   subOb.showk();
   System.out.println();

   System.out.println("Sum of i, j and k in subOb:");
   subOb.sum();
  }
}
```

Another Example

```
// This program uses inheritance to extend Box.
class Box {
  double width;
  double height;
  double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }
```

Another Example

```
// constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1;  // use -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }
```

Another Example

```
// constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

Another Example

```
// Here, Box is extened to include weight.
class BoxWeight  extends Box {
  double weight; // weight of box

  // constructor for BoxWeight
  BoxWeight(double w, double h, double d, double m) {
    width = w;
    height = h;
    depth = d;
    weight = m;
  }
}
```

Another Example

```
class DemoBoxWeight {
  public static void main(String args[]) {
    BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
    BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
    double vol;

    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);
    System.out.println("Weight of mybox1 is " + mybox1.weight);
    System.out.println();

    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);
    System.out.println("Weight of mybox2 is " + mybox2.weight);
  }
}
```

Super

- Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
- Super has two general forms
  - To call superclass constructor.
  - Access to member of the superclass that has been hidden by a member of a subclass.

Another Example

```
// Using super to overcome name hiding.
class A {
  int i;
}
// Create a subclass by extending class A.
class B extends A {
  int i; // this i hides the i in A
  B(int a, int b) {
    super.i = a; // i in A
    i = b; // i in B
  }
  void show() {
    System.out.println("i in superclass: " + super.i);
    System.out.println("i in subclass: " + i);
  }
}
```

Another Example

```
class UseSuper {
  public static void main(String args[]) {
    B subOb = new B(1, 2);
    subOb.show();
  }
}
```

When Constructor are called

- In a class hierarchy, constructor are called in order of derivation, from superclass to subclass.

```
// Demonstrate when constructors are called.
// Create a super class.
class A {
  A() {
    System.out.println("Inside A's constructor.");
  }
}
// Create a subclass by extending class A.
class B extends A {
  B() {
    System.out.println("Inside B's constructor.");
  }
}
```

Example

```
// Create another subclass by extending B.
class C extends B {
  C() {
    System.out.println("Inside C's constructor.");
  }
}

class CallingCons {
  public static void main(String args[]) {
    C c = new C();
  }
}
```

- Output

```
Inside A's constructor.
Inside B's constructor.
Inside C's constructor.
```

Method Overridding

- In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass.
- When a overridden method is called from within a subclass, it will always refer to the version of that defined by the subclass.
- If you wish to access the superclass version of an overridden function, you can do so by using super.

Example

```
// Method overriding.
class A {
  int i, j;

  A(int a, int b) {
    i = a;
    j = b;
  }

  // display i and j
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}
```

Example

```
class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

  // display k -- this overrides show() in A
  void show() {
    System.out.println("k: " + k);
  }
}
```

Example

```
class Override {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);


    subOb.show(); // this calls show() in B
  }
}
```

- Output

k: 3

Dynamik Method Dispatch

- Method overriding forms the basic of it.
- It is a mechanism by which a call to an overridden function is resolved at run time, rather than compile time.
- By this java implement runtime polymorphism.

- Superclass reference variable can refer to a subclass object.
- Java determines which version of that method to execute based upon the type of object being referred to at the time the call occurs.
- In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be called.
- When different types of objects are referred to through superclass reference variable, different versions of the method are executed.

Example

```
// Dynamic Method Dispatch
class A {
   void callme() {
     System.out.println("Inside A's callme method");
  }
}
class B extends A {
  // override callme()
  void callme() {
    System.out.println("Inside B's callme method");
  }
}
class C extends A {
  // override callme()
  void callme() {
    System.out.println("Inside C's callme method");
  }
}
```

Example

```
class Dispatch {
  public static void main(String args[]) {
    A a = new A(); // object of type A
    B b = new B(); // object of type B
    C c = new C(); // object of type C
    A r; // obtain a reference of type A
    r = a; // r refers to an A object
    r.callme(); // calls A's version of callme
    r = b; // r refers to a B object
    r.callme(); // calls B's version of callme
    r = c; // r refers to a C object
    r.callme(); // calls C's version of callme
  }
}
```

Example

- Output

Inside A's callme method

Inside B's callme method

Inside C's callme method

Abstract class

- Methods not defines in the superclass are known as abstract method.
- These methods are to be overridden in the subclass.
- Any class that contains one or more abstract method are known as abstract class.
- There can be no object of abstract class.
- You cannot declare abstract constructors, or abstract static method.
- Although abstract classes cannot be used to instantiate objects, they can be used to create object references, because java's approach to runtime polymorphism is implemented through the use of superclass references.

Example

```
// A Simple demonstration of abstract.
abstract class A {
  abstract void callme();
  // concrete methods are still allowed in abstract classes
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}
class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  }
}
```

Example

```
class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();
    b.callme();
    b.callmetoo();
  }
}
```

final and inheritance

- Methods declared as final cannot be overridden.

```
class A {
  final void meth() {
    System.out.println("This is a final method.");
  }
}
class B extends A {
  void meth() { // ERROR! Can't override.
    System.out.println("Illegal!");
  }
}
```

final and inheritance

- Class declared as final cannot be inherited.

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

// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
  // ...
}
```

The object class

- Special class, object class, defined by java.
- All other classes are subclasses of object class.
- Object class defines the following methods, which means that they are available in each object.
  - object clone()
  - boolean equals(Object.object)
  - void finalize()
  - class getClass()
  - int hashCode()
  - void notify()
  - void notifyAll()
  - string toString()
  - void wait()
  - void wait(long milliseconds)
  - void wait(long milliseconds, int naniseconds)

End of Chapter 8

Questions?