Java

Inheritance, Interface & Exception

Inheritance

Inheritance

- Same Inheritance concept of C++ in Java with some modifications.
- In Java,
  - One class inherits the other using ***extends*** keyword.
  - The classes involved in inheritance are known as ***superclass*** and ***subclass***.
  - ***Multilevel*** inheritance but ***Multiple*** inheritance.
  - There is a special way to call the superclass's ***constructor.***
  - There is automatic ***dynamic method dispatch***.

Simple Inheritance

```
class A {
int i, j;
        void showij() {
        System.out.println(i + " " + j); }
        }
class B extends A {
int k;
        void showk() {
        System.out.println(k);
        }
        void sum() {
        System.out.println(i+j+k);
        }
}
```

```
class SimpleInheritance {
public static void main(String args[]) {
        A superOb = new A();
        B subOb = new B();
        superOb.i = 10;
        superOb.j = 20;
        superOb.showij();
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        subOb.showij();
        subOb.showk();
        subOb.sum();
}
}
```

Practical Example

```
class Box {
	double width, height, depth;
Box(Box ob) {
	width = ob.width;
	height = ob.height;
	depth = ob.depth;}
Box(double w, double h, double d) {
	width = w;
	height = h;
	depth = d;}
Box() {
	width=height=depth= -1;
}
Box(double len) {
width = height = depth = len;
}
double volume()
{
	return width * height * depth;
}
}
class BoxWeight extends Box {
double weight;
BoxWeight(double w, double h,
      double d, double m) {
	width = w;
	height = h;
	depth = d;
	weight = m;
}
```

Superclass variable reference to Subclass object

```
class RefDemo
{
public static void main(String args[]) {
    BoxWeight weightbox = new BoxWeight(3, 5, 7, 8.37);
    Box plainbox = new Box();
    double vol =  weightbox.volume();
    System.out.println();
    // assign BoxWeight reference to Box reference
    plainbox = weightbox;
    vol = plainbox.volume(); // OK, volume() defined in Box
    /* The following statement is invalid because plainbox does not define a
    weight member. */
    double wgt= plainbox.weight;
}
}
```

Using super

```
class BoxWeight extends Box
{
double weight; // weight of box
BoxWeight(BoxWeight ob) {
        super(ob);
        weight = ob.weight;
}
BoxWeight(double w, double h,
    double d, double m) {
        super(w, h, d);
        weight = m;
}
BoxWeight() {
super(); // must be the 1st statement
    in constructor
weight = -1;}
```

```
BoxWeight(double len, double m) {
    super(len);
    weight = m;
} }
class DemoSuper {
public static void main(String args[]) {
BoxWeight mybox1 = new
    BoxWeight(10, 20, 15, 34.3);
BoxWeight mybox2 = new
    BoxWeight(2, 3, 4, 0.076);
BoxWeight mybox3 = new
    BoxWeight();
BoxWeight mycube = new
    BoxWeight(3, 2);
BoxWeight myclone = new
    BoxWeight(mybox1); } }
```

Using super

```
class A {
int i;
}
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);
}}
```

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

Self Study

- Multilevel Inheritance[Page : 167]
- Constructor Calling Sequence [Page : 170]
- Method Overriding [Page : 171]
- Abstract Class [Page: 177]
  - ***abstract class A***
  - contains abstract method ***abstract method f()***
  - No instance can be created of an abstract class
  - The subclass must implement the abstract method
  - Otherwise the subclass will be a abstract class too
- Object Class [Page: 181]

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");}}
```

```
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 callme
	r = b; // r refers to a B object
	r.callme(); // calls B's callme
	r = c; // r refers to a C object
	r.callme(); //calls C‘s callme
	}
}
```

Using final

To prevent overriding.

```
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!");
}
}
```

To prevent Inheritance.

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

Interface

Interface

- We can call it a pure abstract class having no concrete methods.
- All methods declared in an interface are implicitly public and abstract.
- All fields (variables) declared in an interface are implicitly public, static and final.
- A class can only extend from a single class, but a class can implement multiple interfaces.
- When you implement an interface method, it must be declared as ***public.***

Interface

- By implementing an interface, a class signs a contract with the compiler saying that it will definitely provide implementation of the having particular signatures.
- If it fails to do so, the class will be considered as abstract.
- Then it must be declared as abstract and no object of that class can be created.

Interface

```
interface Callback
{
    void callback(int param);
}

class Client implements Callback
{
// Implement Callback's interface
    public void callback(int p)
    {
            System.out.println("callback
    called with " + p);
    }
}
```

Accessing Implementations through Interface References

```
class TestIface
{
public static void main(String args[])
{
    //Callback c1=new Callback();
    //c1.callback(21);
    Client c2 = new Client();
    c2.callback(42);
    Callback c3= new Client();
    c3.callback(84);
} }
```

Self Study

- Applying Interfaces [Page : 197]
- Variables in Interfaces [Page : 200]

Interface can be extended

```
interface A {
void meth1();
void meth2();}

interface B extends A {
void meth3();
}
// This class must implement all of A
    and B
class MyClass implements B {
public void meth1() {
System.out.println("Implement
    meth1().");
}
public void meth2() {
System.out.println("Implement
    meth2().");
}
public void meth3() {
System.out.println("Implement
    meth3().");
}}
class IFExtend {
public static void main(String arg[]) {
MyClass ob = new MyClass();
ob.meth1();
ob.meth2();
ob.meth3();} }
```

Exceptions

Exception

- Uncaught exceptions
- Caught exceptions
- try
- catch
- finally
- throw
- throws
- Creating own exceptions

End