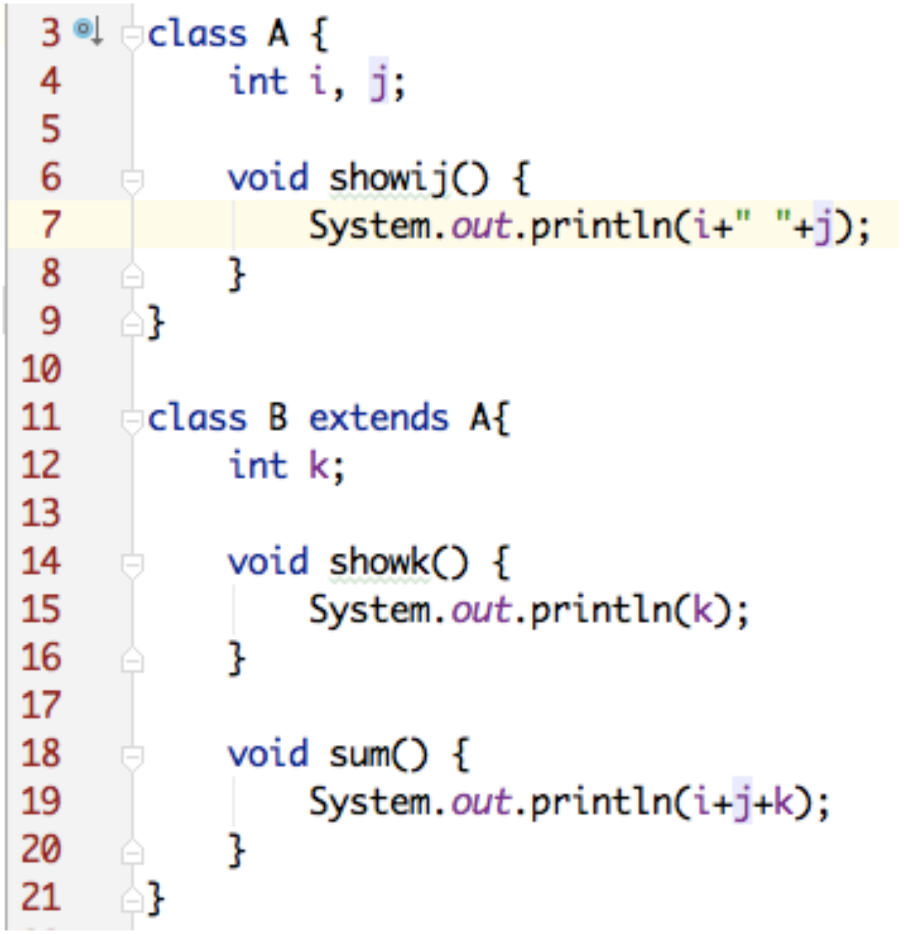
Java

Inheritance

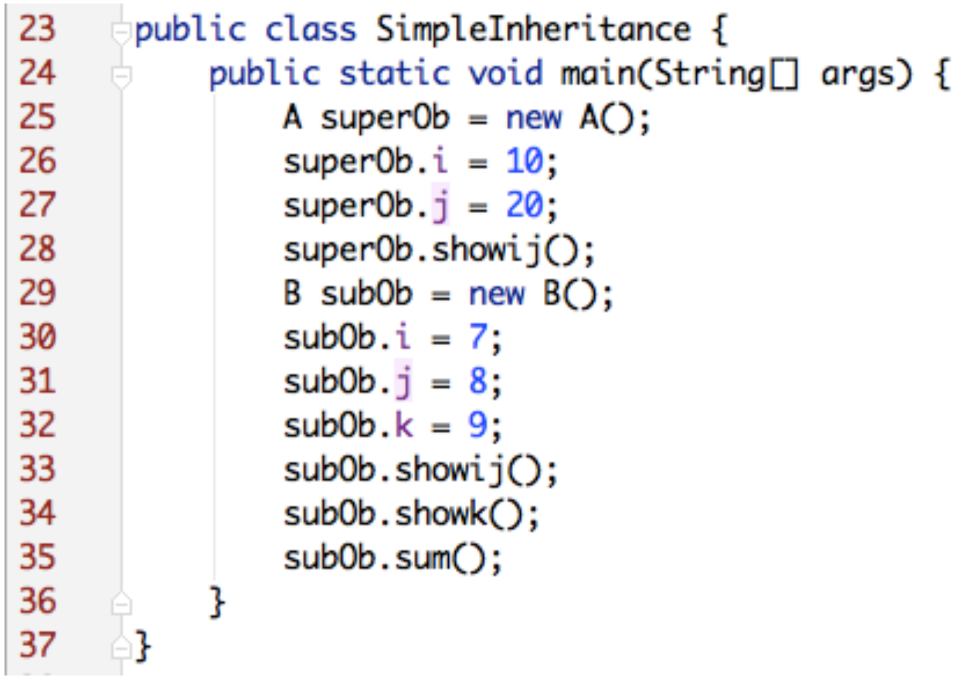
Inheritance

- 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 no ***multiple*** inheritance
  - There is a special way to call the superclass's ***constructor***
  - There is automatic ***dynamic method dispatch***

Simple Inheritance

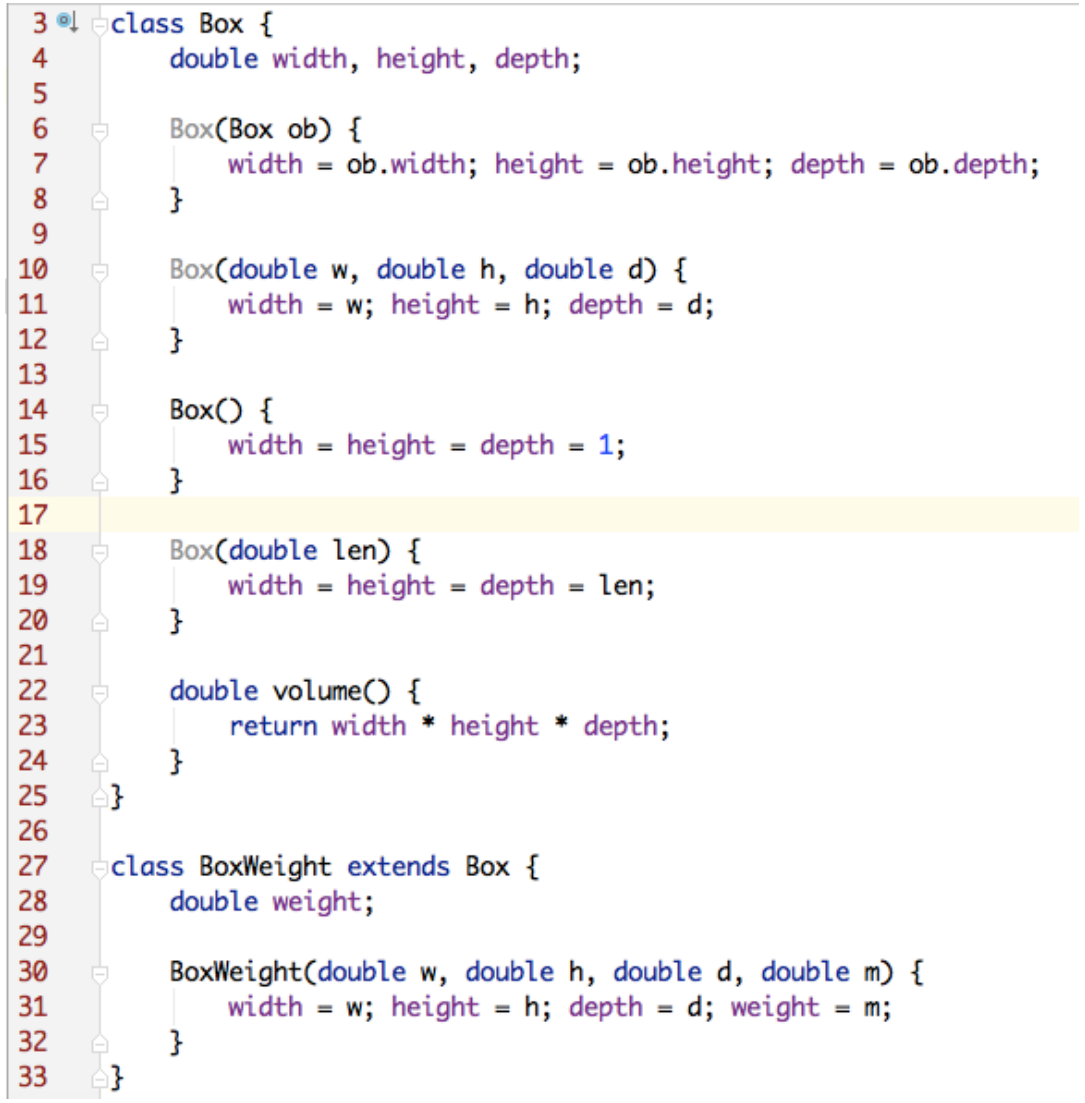


```
class A {
    int i, j;

    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);
    }
}
```



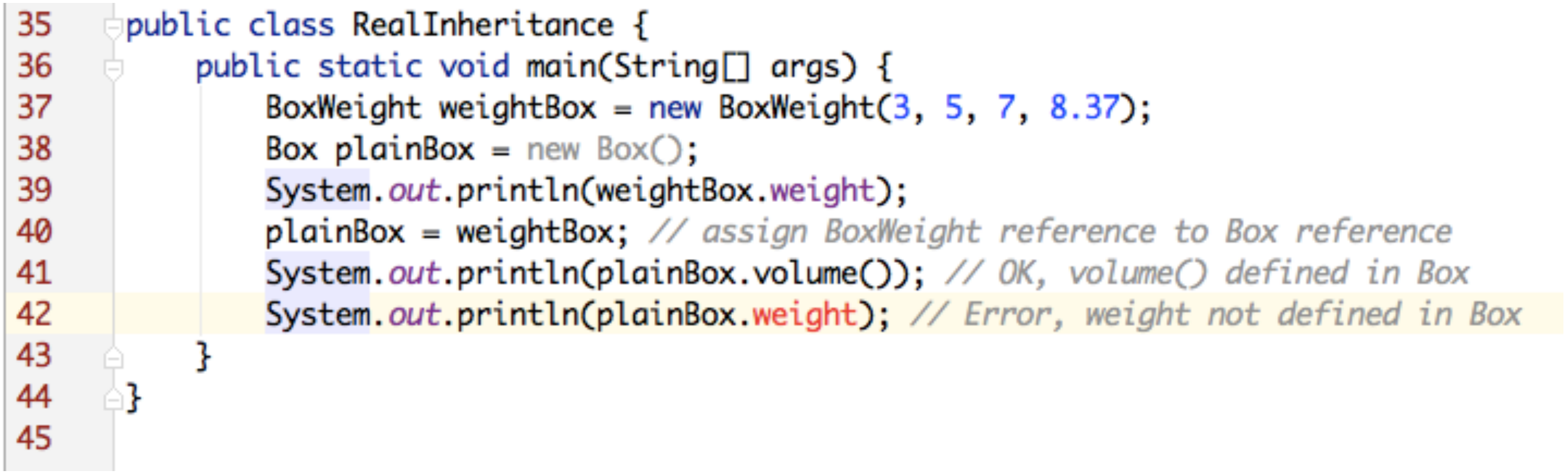
```
public class SimpleInheritance {
    public static void main(String[] args) {
        A superOb = new A();
        superOb.i = 10;
        superOb.j = 20;
        superOb.showij();
        B subOb = new B();
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        subOb.showij();
 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

Practical Example



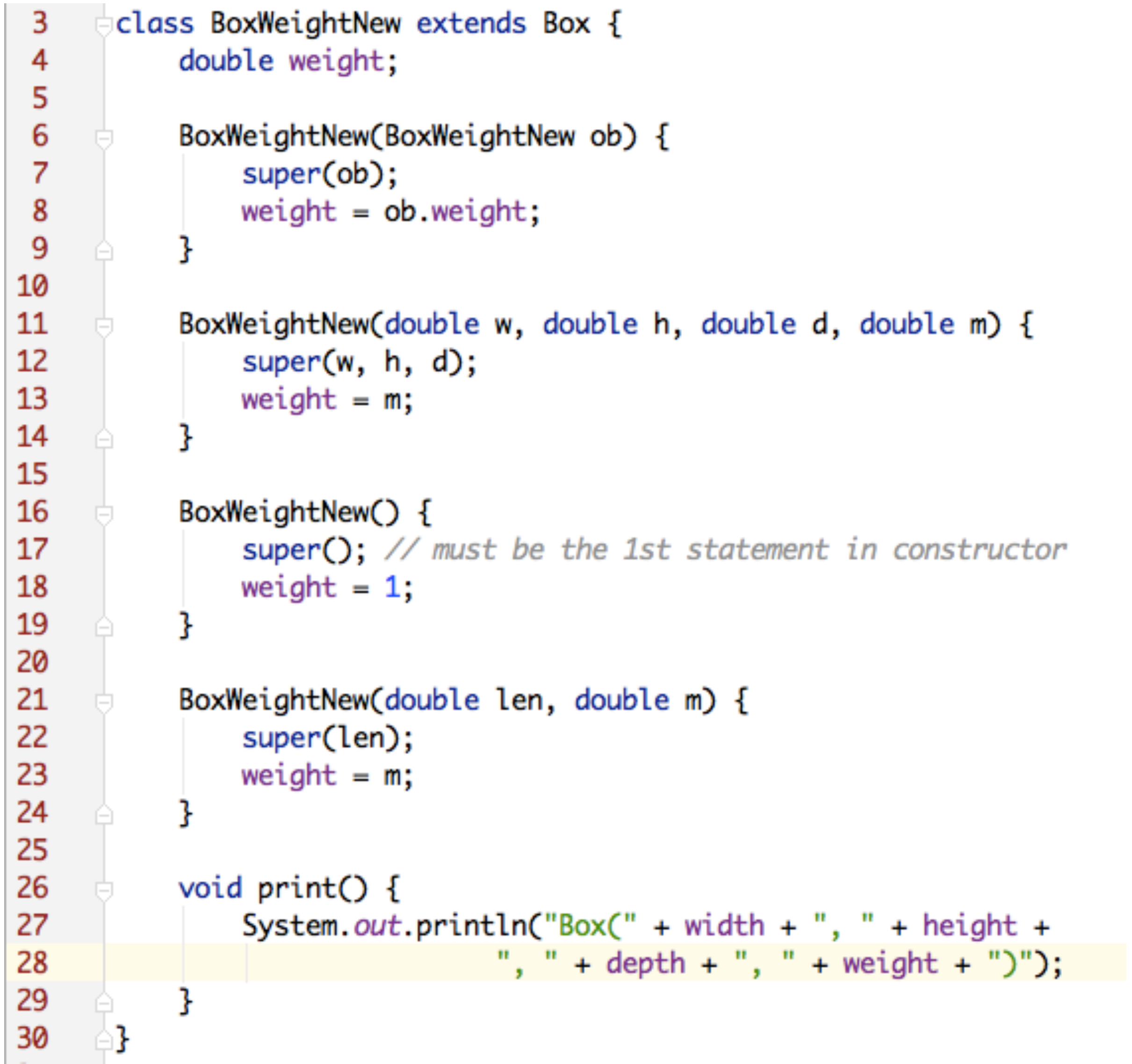
```java
class Box {
    double width, height, depth;

    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



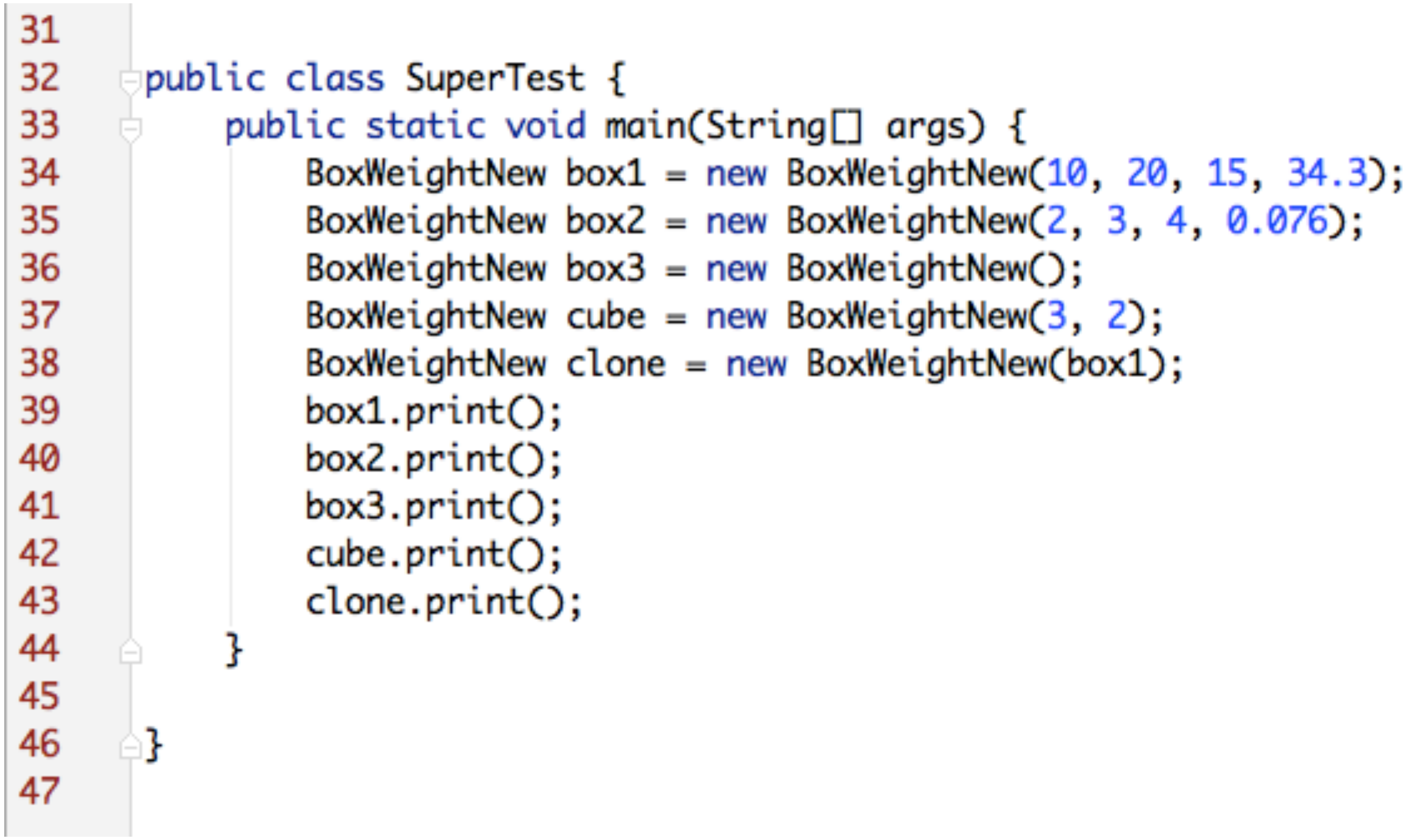
```java
public class RealInheritance {
    public static void main(String[] args) {
        BoxWeight weightBox = new BoxWeight(3, 5, 7, 8.37);
        Box plainBox = new Box();
        System.out.println(weightBox.weight);
        plainBox = weightBox; // assign BoxWeight reference to Box reference
        System.out.println(plainBox.volume()); // OK, volume() defined in Box
        System.out.println(plainBox.weight); // Error, weight not defined in Box
    }
}
```

Using super



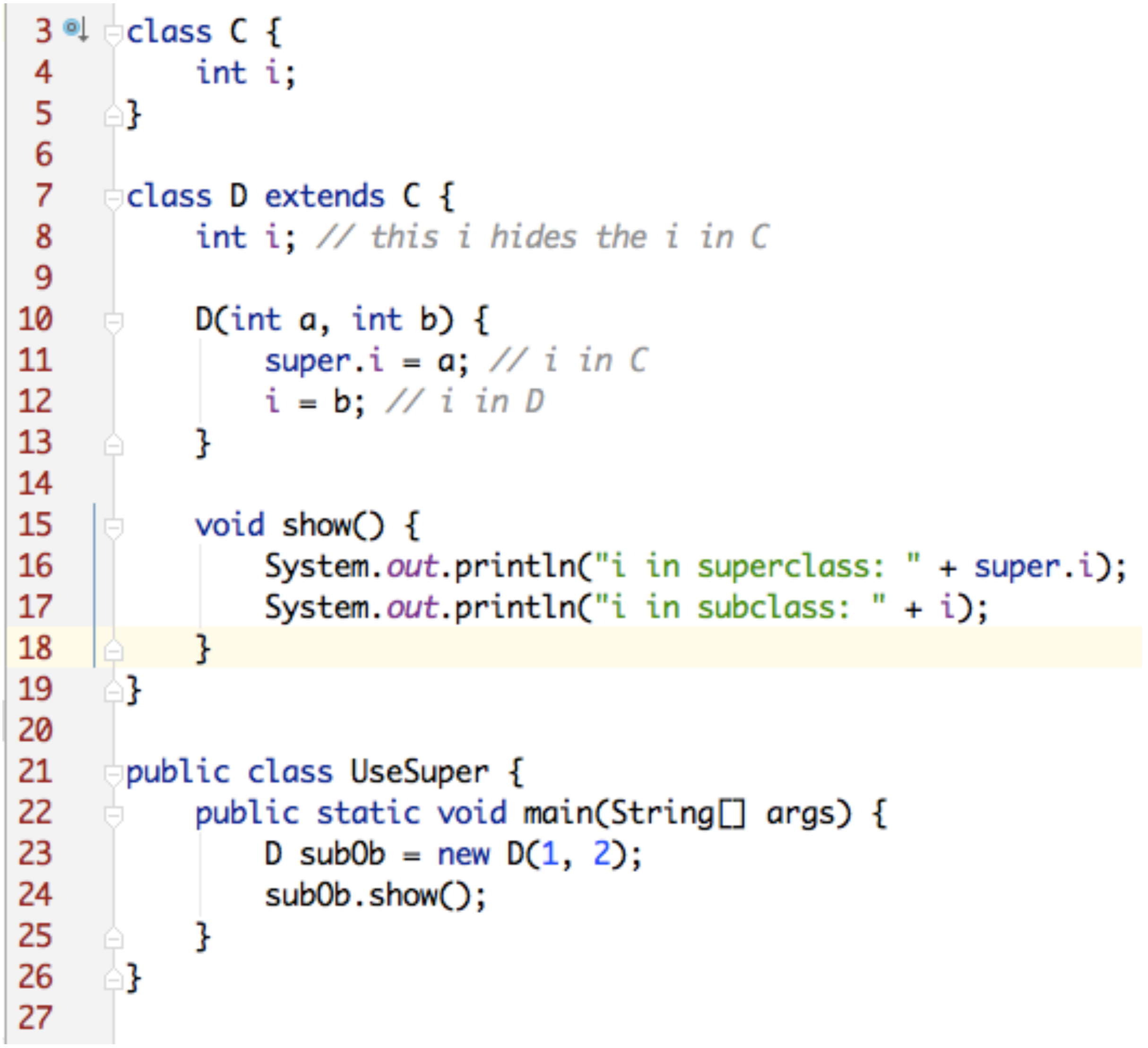
```java
class BoxWeightNew extends Box {
    double weight;

    BoxWeightNew(BoxWeightNew ob) {
        super(ob);
        weight = ob.weight;
    }

    BoxWeightNew(double w, double h, double d, double m) {
        super(w, h, d);
        weight = m;
    }

    BoxWeightNew() {
        super(); // must be the 1st statement in constructor
        weight = 1;
    }

    BoxWeightNew(double len, double m) {
        super(len);
        weight = m;
    }

    void print() {
        System.out.println("Box(" + width + ", " + height +
                        ", " + depth + ", " + weight + ")");
    }
}
```

Using super



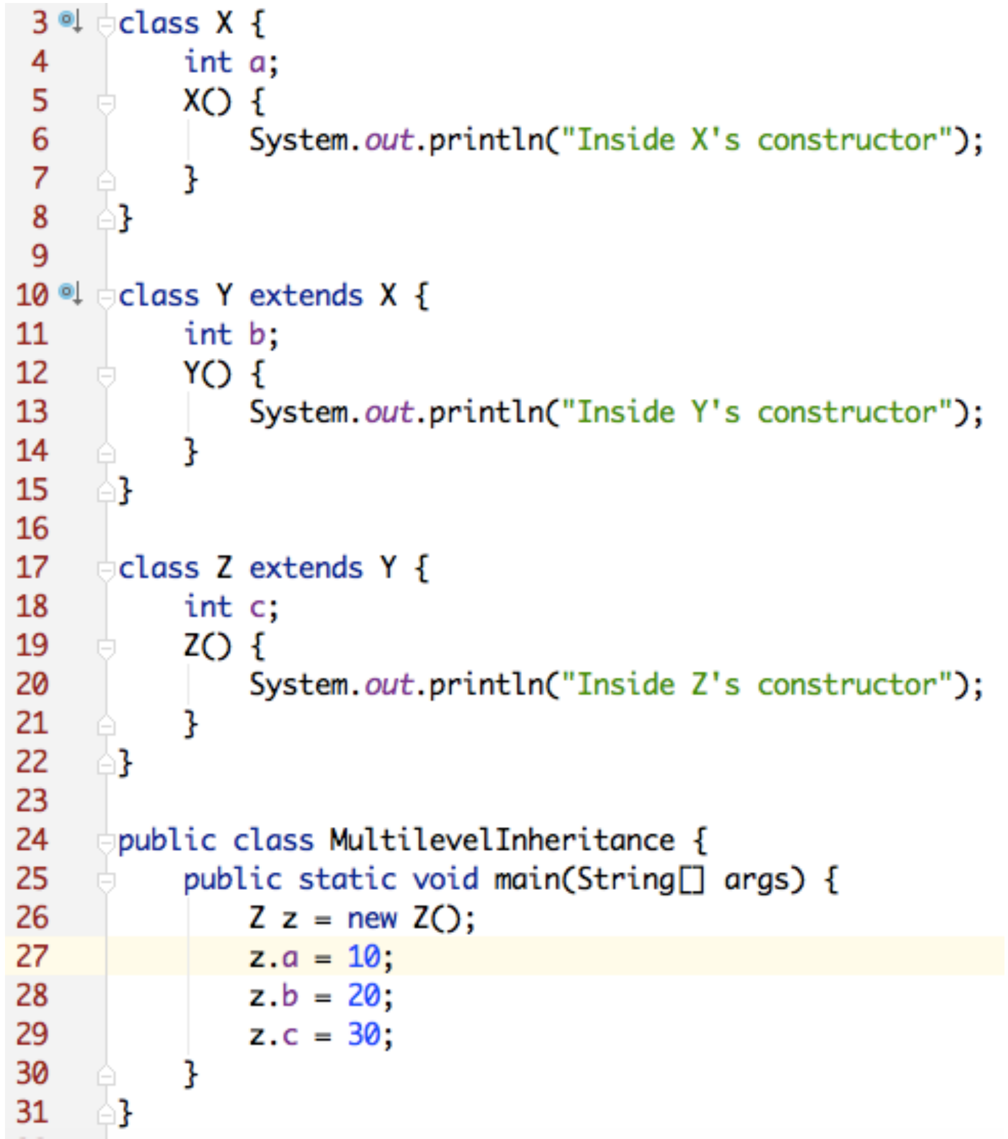
```
public class SuperTest {
    public static void main(String[] args) {
        BoxWeightNew box1 = new BoxWeightNew(10, 20, 15, 34.3);
        BoxWeightNew box2 = new BoxWeightNew(2, 3, 4, 0.076);
        BoxWeightNew box3 = new BoxWeightNew();
        BoxWeightNew cube = new BoxWeightNew(3, 2);
        BoxWeightNew clone = new BoxWeightNew(box1);
        box1.print();
        box2.print();
        box3.print();
        cube.print();
        clone.print();
    }

}
```

Using super



```
class C {
    int i;
}

class D extends C {
    int i; // this i hides the i in C

    D(int a, int b) {
        super.i = a; // i in C
        i = b; // i in D
    }

    void show() {
        System.out.println("i in superclass: " + super.i);
        System.out.println("i in subclass: " + i);
    }
}

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

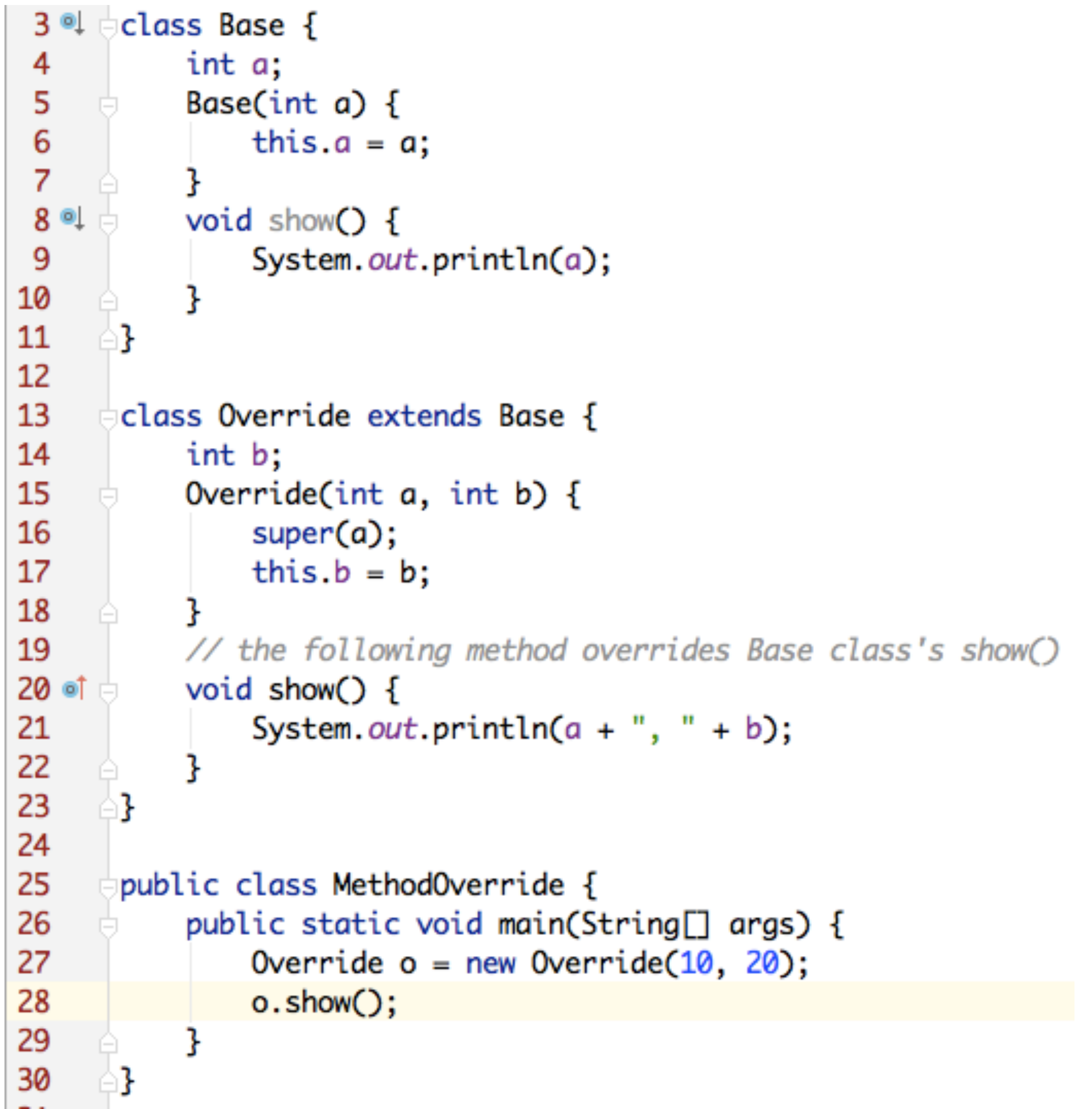
Multilevel Inheritance



```
class X {
    int a;
    X() {
        System.out.println("Inside X's constructor");
    }
}

class Y extends X {
    int b;
    Y() {
        System.out.println("Inside Y's constructor");
    }
}

class Z extends Y {
    int c;
    Z() {
        System.out.println("Inside Z's constructor");
    }
}

public class MultilevelInheritance {
    public static void main(String[] args) {
        Z z = new Z();
        z.a = 10;
        z.b = 20;
        z.c = 30;
    }
}
```

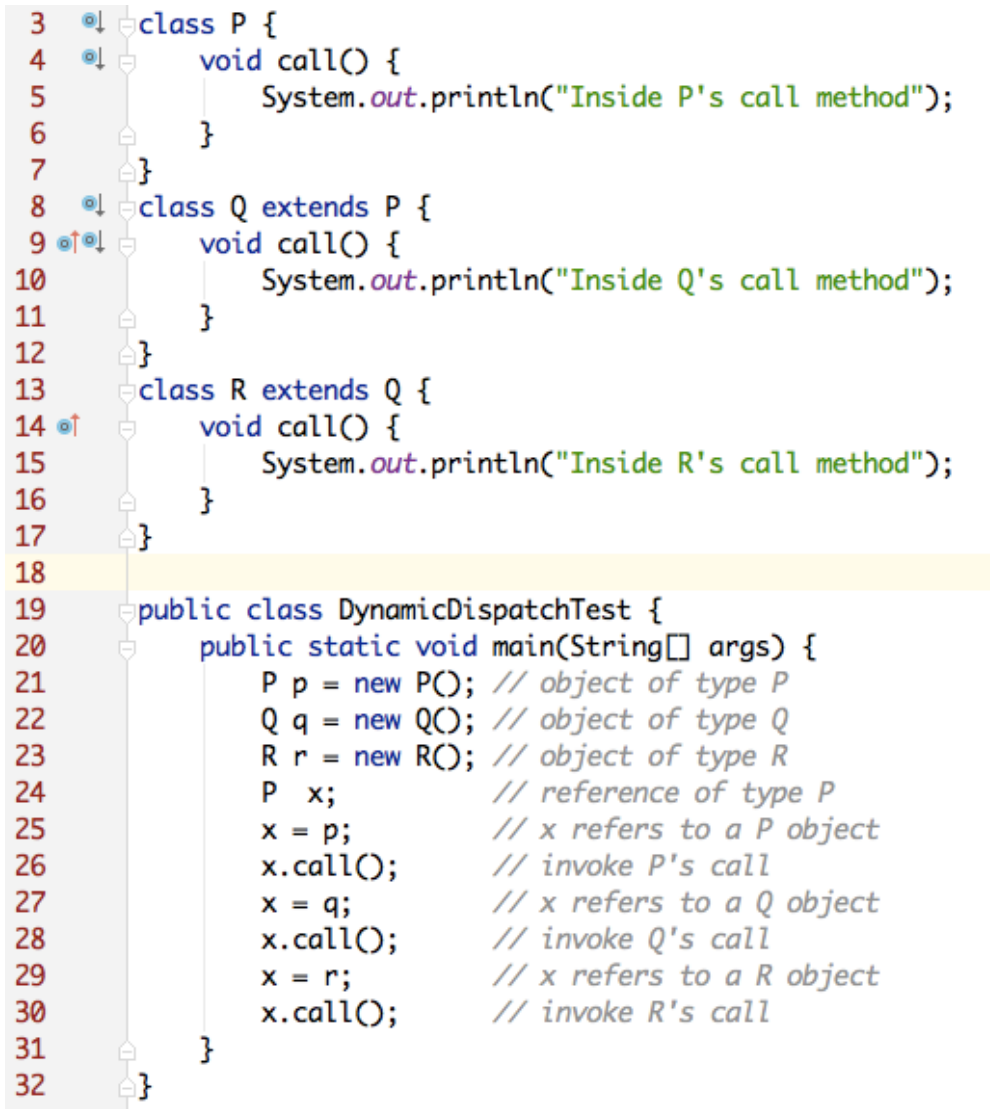
```
Inside X's constructor
Inside Y's constructor
Inside Z's constructor
```

Method Overriding



```
class Base {
    int a;
    Base(int a) {
        this.a = a;
    }
    void show() {
        System.out.println(a);
    }
}

class Override extends Base {
    int b;
    Override(int a, int b) {
        super(a);
        this.b = b;
    }
    // the following method overrides Base class's show()
    void show() {
        System.out.println(a + ", " + b);
    }
}

public class MethodOverride {
    public static void main(String[] args) {
        Override o = new Override(10, 20);
        o.show();
    }
}
```

Dynamic Method Dispatch

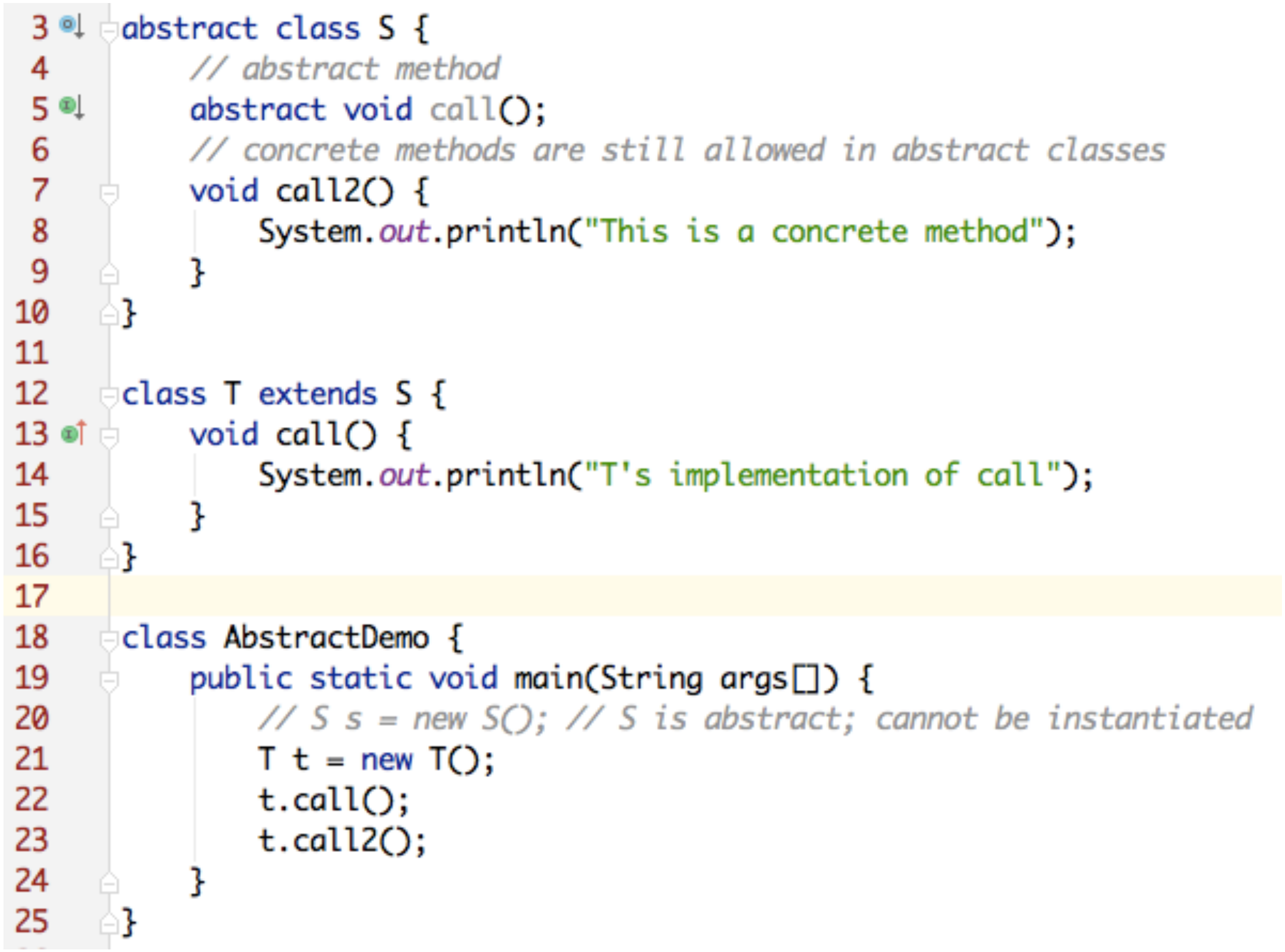


```java
class P {
    void call() {
        System.out.println("Inside P's call method");
    }
}
class Q extends P {
    void call() {
        System.out.println("Inside Q's call method");
    }
}
class R extends Q {
    void call() {
        System.out.println("Inside R's call method");
    }
}

public class DynamicDispatchTest {
    public static void main(String[] args) {
        P p = new P(); // object of type P
        Q q = new Q(); // object of type Q
        R r = new R(); // object of type R
        P  x;          // reference of type P
        x = p;         // x refers to a P object
        x.call();      // invoke P's call
        x = q;         // x refers to a Q object
        x.call();      // invoke Q's call
        x = r;         // x refers to a R object
        x.call();      // invoke R's call
    }
}
```

Abstract Class

- ***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

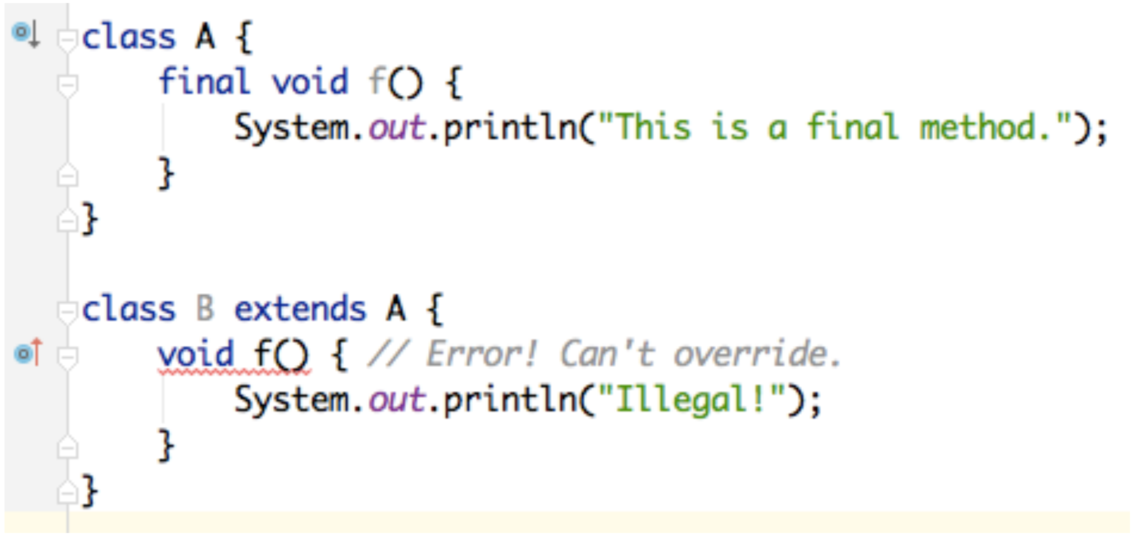
Abstract Class



```java
abstract class S {
    // abstract method
    abstract void call();
    // concrete methods are still allowed in abstract classes
    void call2() {
        System.out.println("This is a concrete method");
    }
}

class T extends S {
    void call() {
        System.out.println("T's implementation of call");
    }
}

class AbstractDemo {
    public static void main(String args[]) {
        // S s = new S(); // S is abstract; cannot be instantiated
        T t = new T();
        t.call();
        t.call2();
    }
}
```

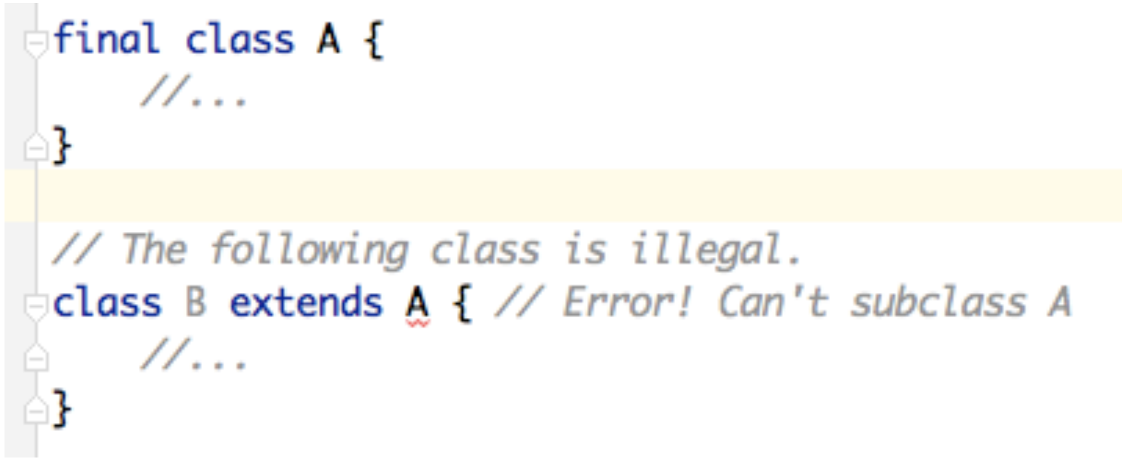
Using final with Inheritance

To prevent overriding



```
class A {
    final void f() {
        System.out.println("This is a final method.");
    }
}

class B extends A {
    void f() { // 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
    //...
}
```

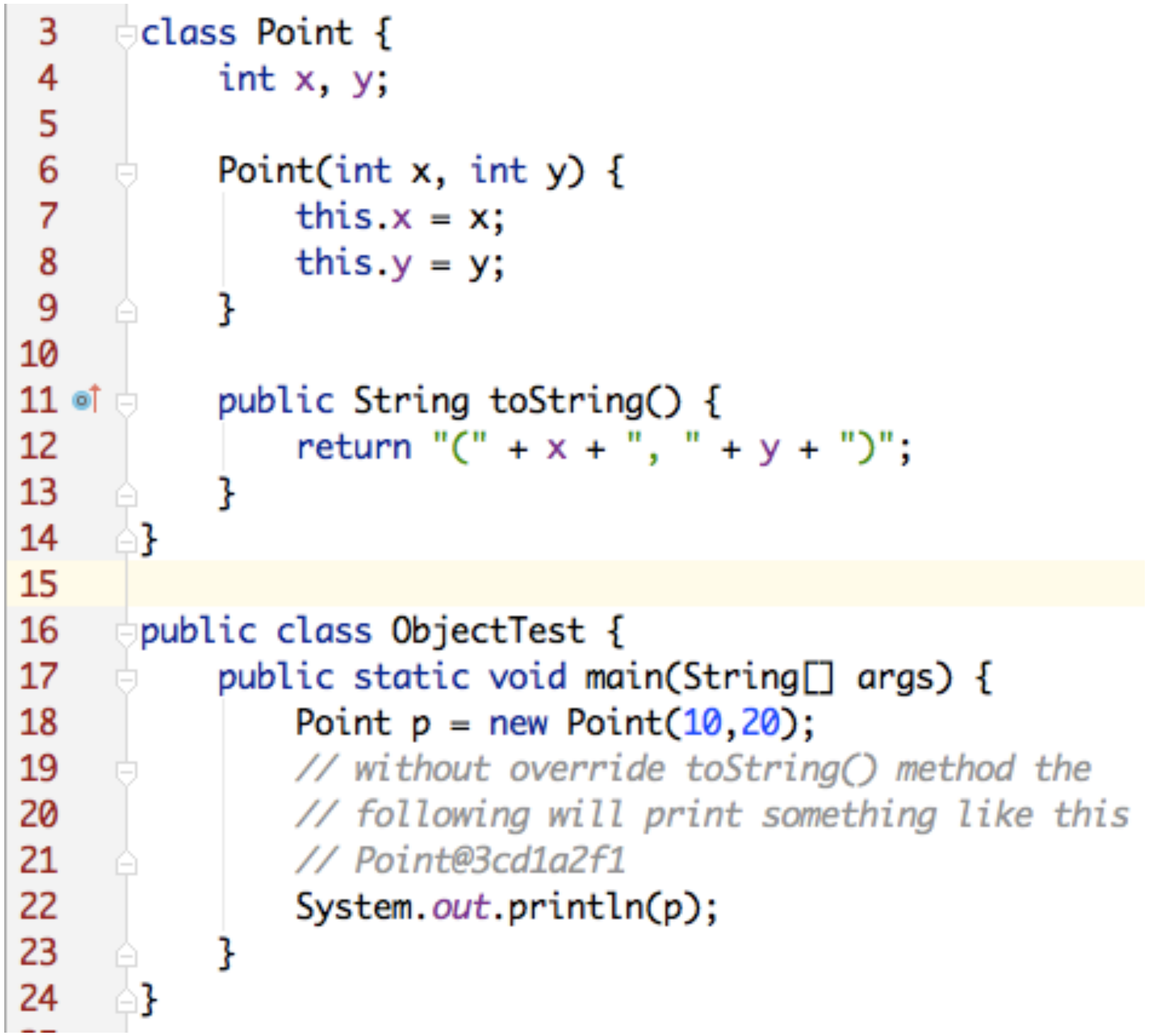
Object Class

- There is one special class, Object, defined by Java
- All other classes are subclasses of Object
- That is, Object is a superclass of all other classes
- This means that a reference variable of type Object can refer to an object of any other class
- Also, since arrays are implemented as classes, a variable of type Object can also refer to any array

Object’s toString()

- The toString( ) method returns a string that contains a description of the object on which it is called
- Also, this method is automatically called when an object is output using println()
- Many classes override this method
- Doing so allows them to tailor a description specifically for the types of objects that they create

Object's toString()



```
class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}

public class ObjectTest {
    public static void main(String[] args) {
        Point p = new Point(10,20);
        // without override toString() method the
        // following will print something like this
        // Point@3cd1a2f1
        System.out.println(p);
    }
}
```