More Inheritance

- super
- Constructors
- The Object Class
- Random Point

See Also

https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

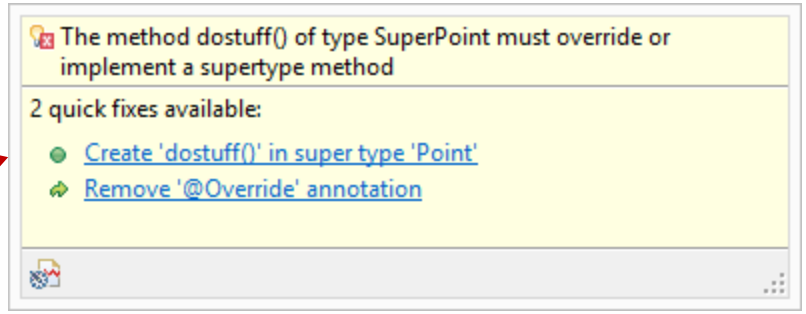
http://www.homeandlearn.co.uk/java/java_inheritance.html

http://beginnersbook.com/2013/05/java-inheritance-types/

@Override

- Case matters. “doStuff” and “dostuff” are different
- You can tell the compiler you mean to override by using an “annotation”
- The compiler will give you an error

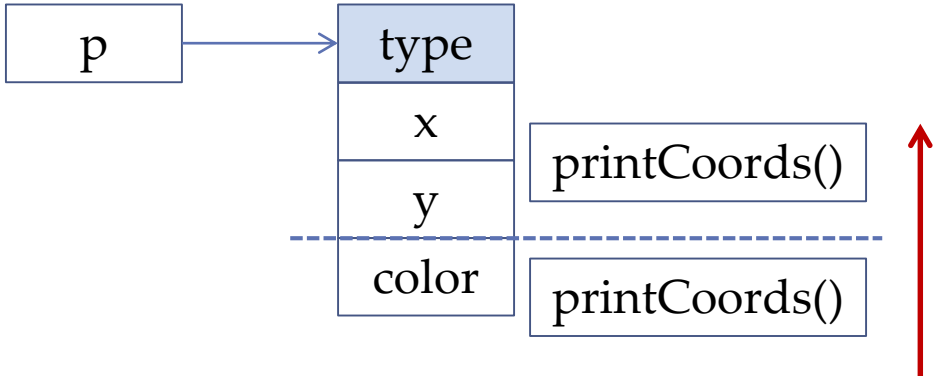
```
class Point {
    public void doStuff() {
        System.out.println("Hello");
    }
}

class SuperPoint extends Point{

    @Override
    public void dostuff() {
        System.out.println("SUPER");
    }
}
```



super

- "super" means "up" … up to the base class
- You can call methods in your direct base class

```
class Point {
    private int x;
    private int y;

    public void printCoord() {
        System.out.println(x+","+y);
    }
}

class ColorPoint extends Point {
    private int color;

    public void printCoord() {

        super.printCoord();

        System.out.println("Color is "+color);
    }
}
```




```
ColorPoint p = new ColorPoint();
p.printCoord();
```

```
0,0
Color is 0
```

Chained Constructors

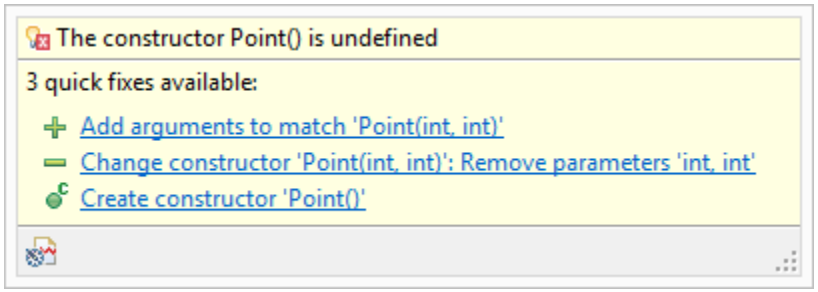
```
class Point {
    private int x;
    private int y;

    public Point() {

    }
}

class ColorPoint extends Point {
    private int color;

    public ColorPoint(int c) {
        super();
        color = c;
    }
}
```

- The compiler gives you a constructor if you don't define one.
- The FIRST thing your constructor must do is call one of the base class constructors (you can pick).
- If you don't explicitly call one then the compiler inserts a call to the base class's no-args constructor.

Chained Constructors

```
class Point {
    private int x;
    private int y;

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

class ColorPoint extends Point {
    private int color;

    public ColorPoint(int c) {
        super();
        color = c;
    }
}
```

?



- What if there isn't a no-args constructor in the base class?
- You will get a compile error

Chained Constructors

```
class Point {
    private int x;
    private int y;

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

class ColorPoint extends Point {
    private int color;

    public ColorPoint(int c) {
        super(0,0);
        color = c;
    }
}
```

- You must explicitly invoke a base constructor

```
class ColorPoint extends Point {
    private int color;

    public ColorPoint(int x, int y, int c) {
        super(x,y);
        color = c;
    }
}
```

Object

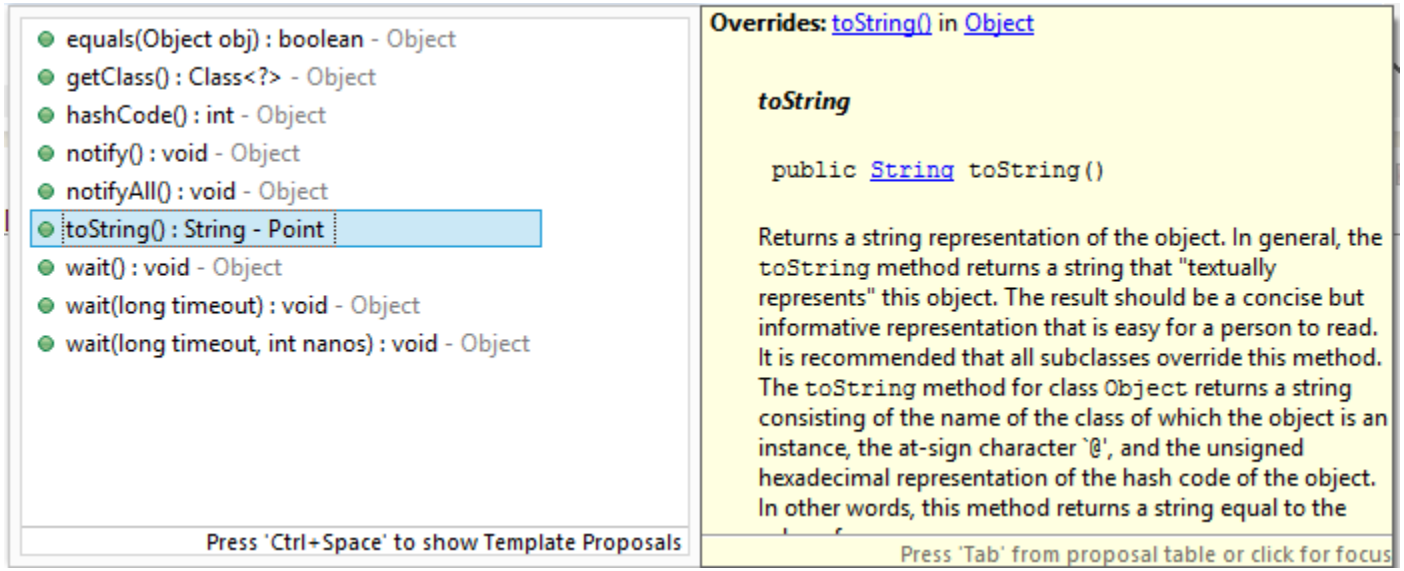
```
class Point extends Object {

    private int x;
    private int y;

    public Point() {
        super();
    }

}
```

- Every class you make MUST extend some base class
- If you don't explicitly give one then the compiler will insert "extends Object"
- Object has no public data. But it has some useful methods to use and override.
- Object is the base of ALL objects. You can call the methods defined in Object on any and all objects you create.

Object

- You can create instances of Object

```
Object ob = new Object(); // in java.lang
ob.
```



toString()

```
class Point {
    private int x;
    private int y;

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

Point p = new Point();
System.out.println(p.toString());

System.out.println(p);
```

- The "toString" method returns a string representation of the object
- The default shows the "hashcode", which is (mostly) unique for each object
- You can override the Object version
- The compiler knows every object has a "toString". If you print a "pointer" then it inserts the "toString".

```
Point@5270cdd2
```

```
0,0
```

Methods of Object

`public String toString()`	Return a string representation (often used in debugging)
`public boolean equals(Object obj)`	Agreed upon name for your "compareTo" method like we did earlier
`public int hashCode()`	Returns mostly-unique value used in hash-maps (later)
`public Class<?> getClass()`	Runtime-type information for reflection

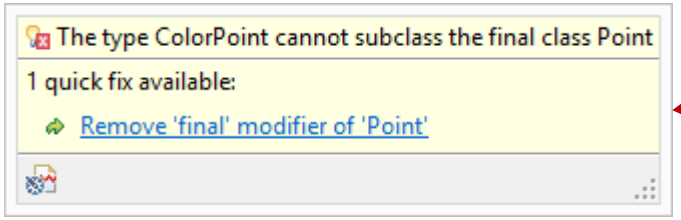
`public void notify()`
`public void wait()`
`public void wait(long timeout)`
`public void wait(long timeout, int nanos)`
	Used in multi-threading

"final" and "protected"

```
class Point {
    protected int x;
    protected int y;
}


class ColorPoint extends Point {

    public void doStuff() {
        x = 20; // Permission ...
        y = 40; // ... granted
    }

}
```

- "protected" data and methods are "public" to derived classes
- A "final" class cannot be extended

```
final class Point {
    protected int x;
    protected int y;
}


class ColorPoint extends Point {


}
```



The Random Point

```java
class Point {

    int x;
    int y;

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

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

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

}
```

```java
Point p = new Point(1,2);
System.out.println(p.toString());

RandomPoint r = new RandomPoint();
System.out.println(r);
System.out.println(r);
System.out.println(r);
```


```java
class RandomPoint extends Point {

    public RandomPoint() {
        super(0,0);
    }
    @Override
    public int getX() {
        Random rand = new Random();
        return rand.nextInt();
    }
    @Override
    public int getY() {
        Random rand = new Random();
        return rand.nextInt();
    }
}
```

The Random Point

```java
class Point {

    int x;
    int y;

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

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    @Override
    public String toString() {
        return getX()+", "+getY();
    }

}
```

```java
Point p = new Point(1,2);
System.out.println(p.toString());

RandomPoint r = new RandomPoint();
System.out.println(r);
System.out.println(r);
System.out.println(r);
```

```
1, 2
840783894, -631984485
-2080906217, 1852971609
-874011925, -548106830
```

```java
class RandomPoint extends Point {

    public RandomPoint() {
        super(0,0);
    }
    @Override
    public int getX() {
        Random rand = new Random();
        return rand.nextInt();
    }
    @Override
    public int getY() {
        Random rand = new Random();
        return rand.nextInt();
    }
}
```

Your Turn

- Code up Point and ColorPoint. Give them constructors (with arguments) and "toString" methods.
- Create some Point and ColorPoints. Print their string representations.
- Add an "equals" method to Point. Create several Point objects and compare them.