Inheritance

- Extending Existing Classes
- OO Terminology
- Overrides
- Polymorphism

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/

Reusing Implementation

- The "Point" has been in the field for 20 years
- We need new code with a Point that has color
- Copy paste?

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

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

$\longrightarrow$

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

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

    private int color;

    public void printColor() {
        System.out.println(color);
    }
}
```

- Code Mutation

Extending Implementation

- Use "extends" and your new class starts with the target class as a base and adds to it.
- You can't take anything away.

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

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

    private int color;

    public void printColor() {
        System.out.println(color);
    }
}
```

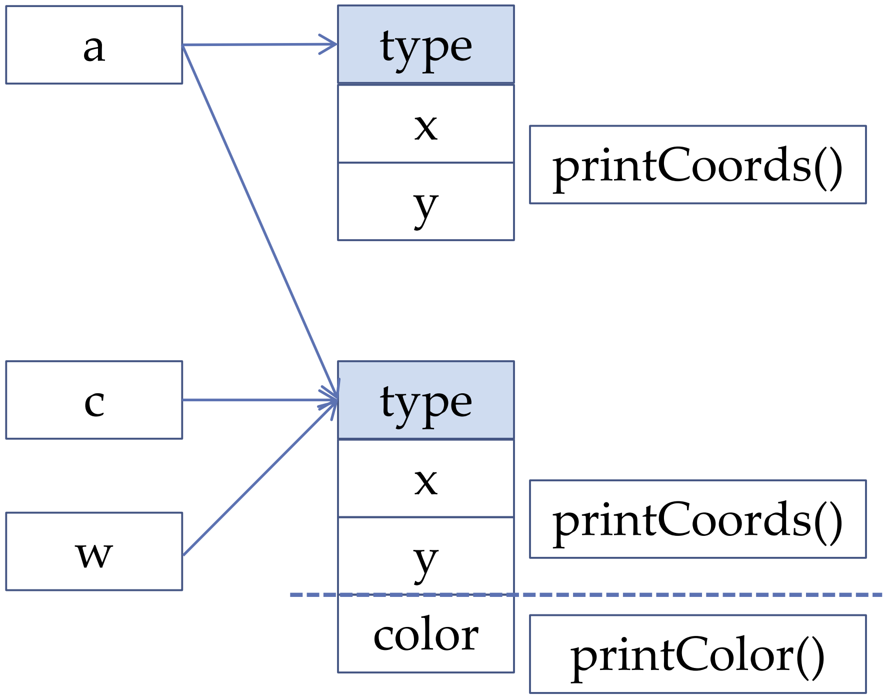
```
ColorPoint p = new ColorPoint();

p.x = 20; // If this were public

p.printCoords();
```

Memory Footprint

```
Point a = new Point();

ColorPoint c = new ColorPoint();

a = c; // Copy the pointer

ColorPoint w = a; // Are you sure?
w = (ColorPoint)a; // Yes ... I am sure

if(a instanceof ColorPoint) {
    w = (ColorPoint) a;
}
```

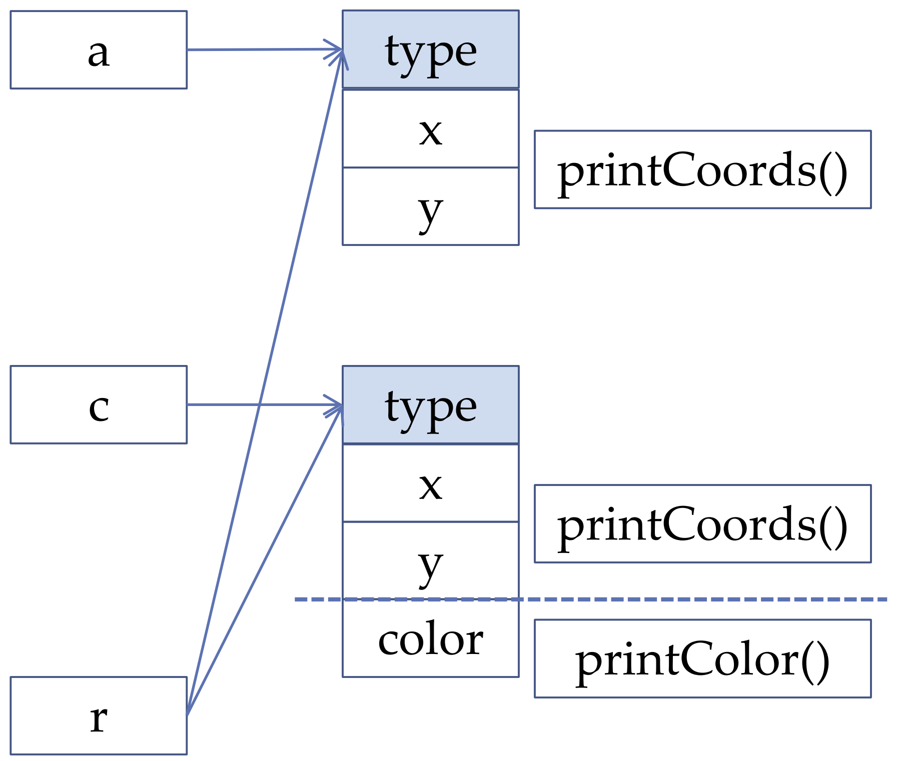
Type mismatch: cannot convert from Point to ColorPoint

- “upcasts” always work. They are automatic.
- “downcasts” require your assurance (but the runtime still checks)
- “instanceof” operator



Memory Footprint

```
Point a = new Point();

ColorPoint c = new ColorPoint();

haveFun(a);

haveFun(c);

public static void haveFun(Point r) {
    r.printCoords();
    r.printCoords();
}
```



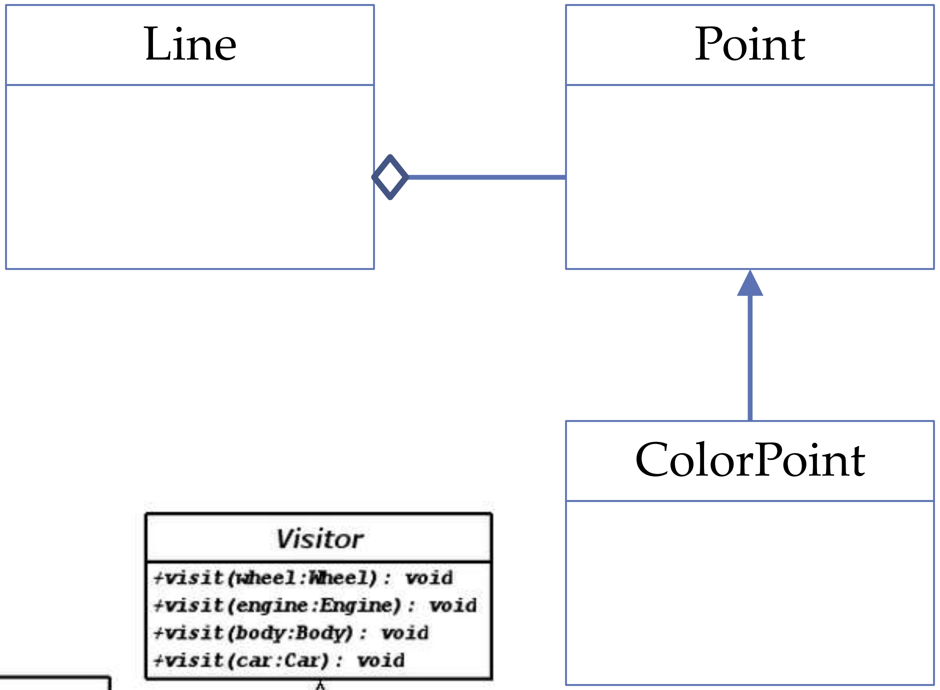
- Upcasts are automatic
- "haveFun" ignores the extra functionality tacked onto the end of the object

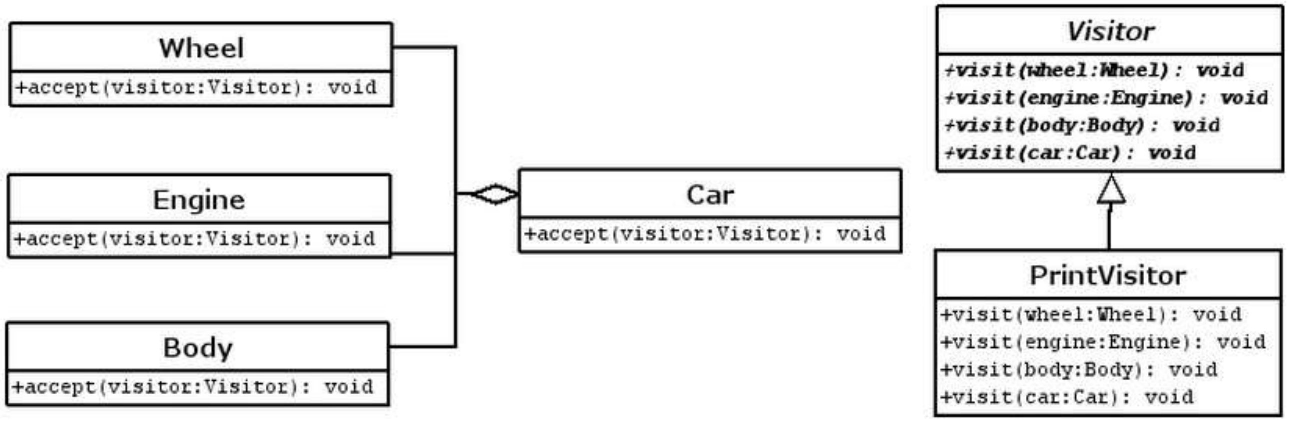
OO Terminology

- ColorPoint extends Point
- ColorPoint inherits from Point
- Point is the base class
- ColorPoint is the derived class

- ColorPoint is-a Point

- Line has-a Point (two of them)





Wheel
+accept(visitor:Visitor): void
Engine
+accept(visitor:Visitor): void
Body
+accept(visitor:Visitor): void
Car
+accept(visitor:Visitor): void
Visitor
+visit(wheel:Wheel): void
+visit(engine:Engine): void
+visit(body:Body): void
+visit(car:Car): void
PrintVisitor
+visit(wheel:Wheel): void
+visit(engine:Engine): void
+visit(body:Body): void
+visit(car:Car): void

Method Overrides

- Your derived class can “override” methods in the base class by redefining the implementation.
- You can’t hide methods by making them more private. This would be “taking away” – a no, no.
- How would you like this code to behave?

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

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

}
```

```java
class ColorPoint extends Point {

    private int color;

    public void printCoord() {
        System.out.println("Color is "+color);
    }

}
```

Method Overrides

```
Point a = new Point();
a.printCoord();
```

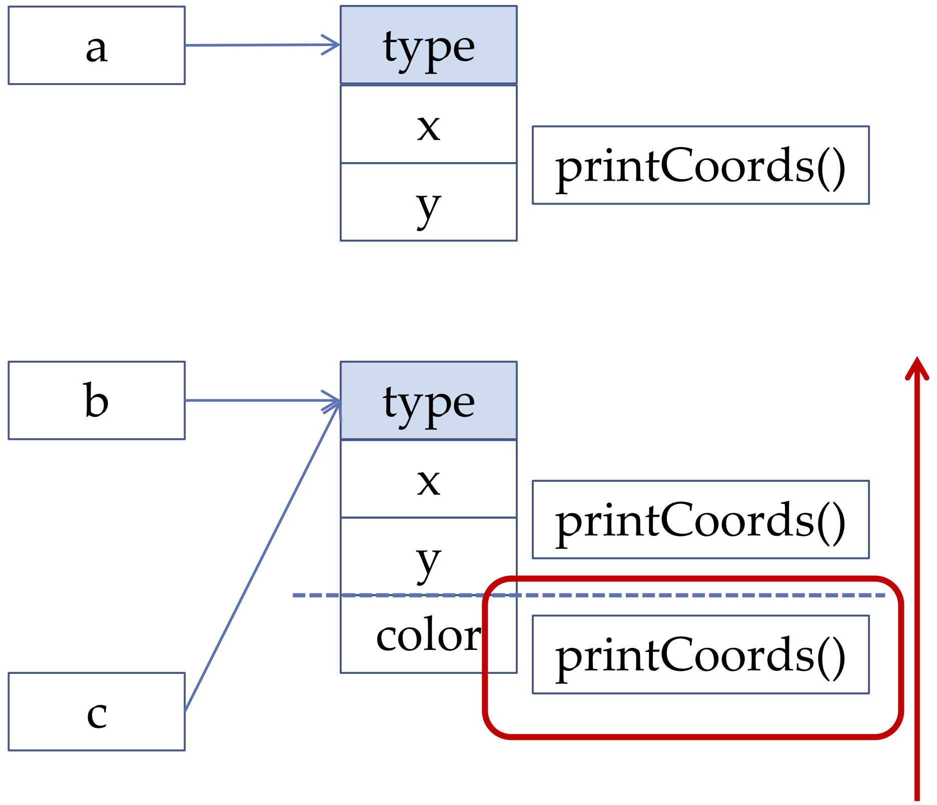
```
0, 0
```

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

```
Color is 0
```

```
Point c = b;
c.printCoord();
```

```
Color is 0
```



- The method used is ALWAYS the most derived one
- The pointer DOES NOT MATTER

Polymorphism

- Different objects can react to the same method in different ways. Same name – different function.
- You call methods on objects without knowing (or caring) what they REALLY are.
- Your code works with a variety of different objects – even objects that haven't been written yet.

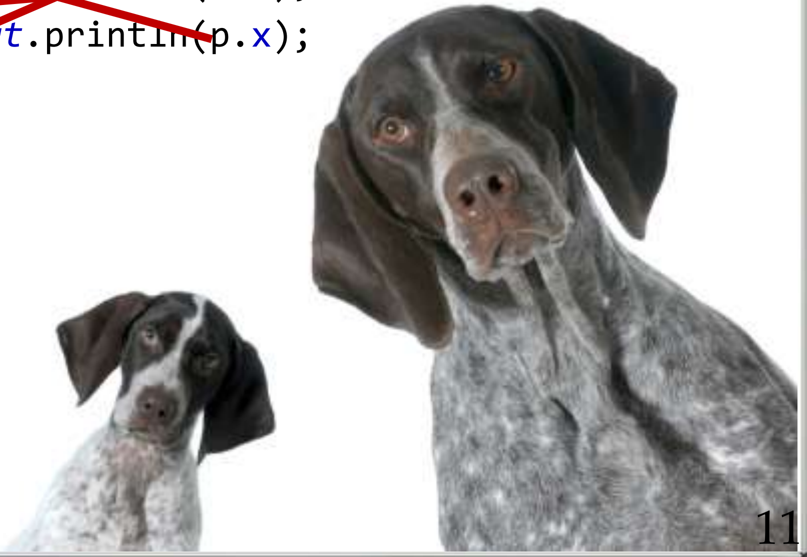
```
void comeAndGo(Animal animal) {
    animal.sayHi();
    animal.sayBye();
}

comeAndGo(cat);
comeAndGo(dog);
comeAndGo(robot);
```

Pointers and Data

- Data is NOT polymorphic
- In the case of data the pointer type DOES matter

```java
class Point {
    public int x = 10;
    public void printOne() {
        System.out.println(x);
    }
}

class SuperPoint extends Point{
    public void printTwo() {
        System.out.println(x);
    }
}

class DuperPoint extends SuperPoint {
    public int x = 20;
    public void printThree() {
        System.out.println(x);
    }
}
```

```java
DuperPoint d = new DuperPoint();
SuperPoint s = d;
Point p = d;

System.out.println(d.x);
System.out.println(s.x);
System.out.println(p.x);
```



Your Turn

- Implement Animal, Dog, Cat and Robot from this lesson.
- Add a “numberOfLegs” to the baseclass.
- Write static function “legCheck” that prints the number of legs of any animal passed to it.