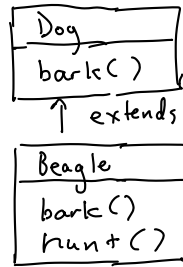


lecture 30

- polymorphism (review)

- interfaces
- abstract classes
- casting



```

Dog myDog = new Beagle("Buddy")
myDog.bark()
    
```

" Polymorphism "

```

Dog myDog = new Beagle("Buddy")
myDog.bark()
    
```

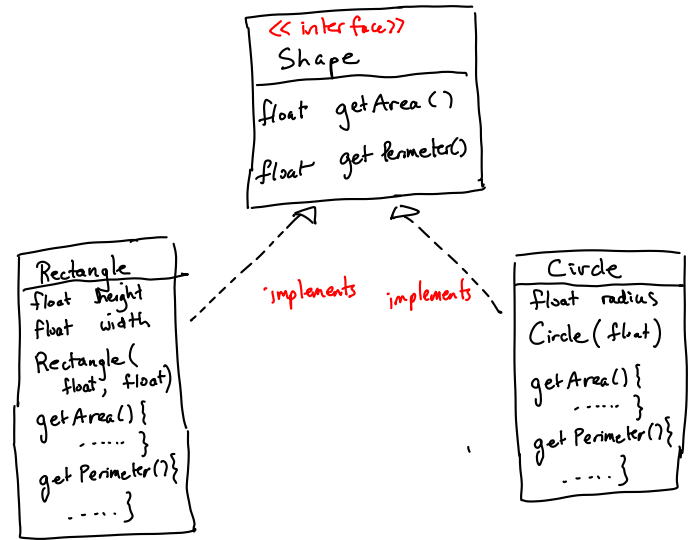
Compile time

- myDog can be assigned a Dog or any subtype of Dog

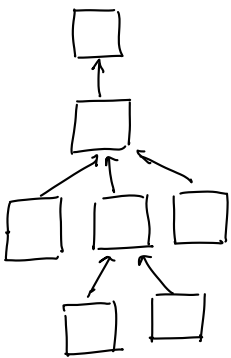
runtime

The method invoked is determined by the object's class (run time).

Interfaces - Example

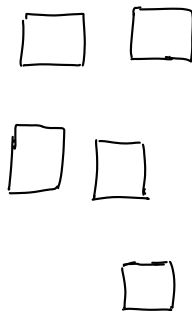


Classes



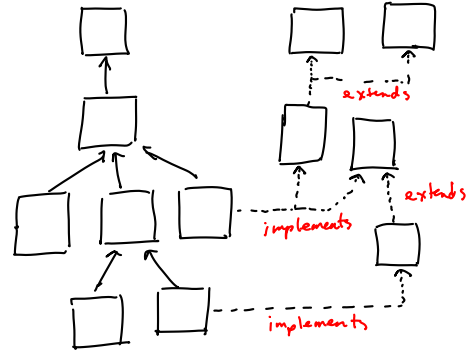
Class definitions form a tree (parent links, but no children links)

interfaces



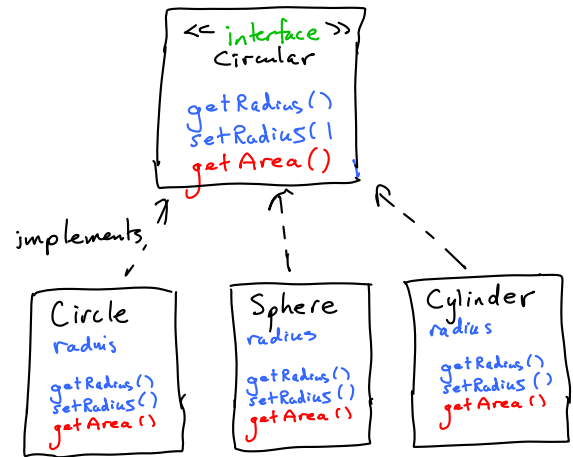
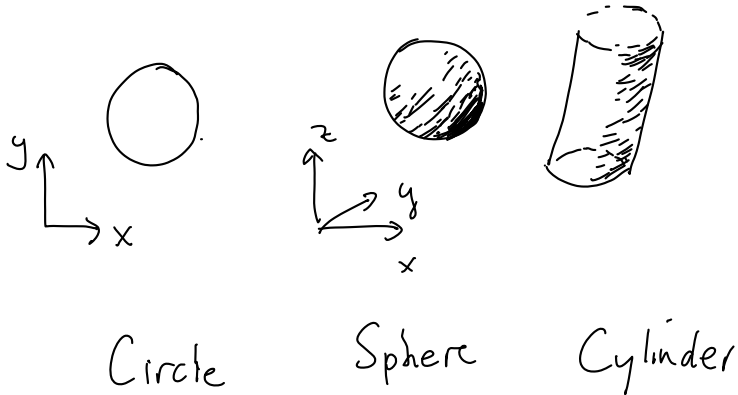
Interfaces contain only method signatures (and constants)

Classes



A subclass can **extend** one super class.
 A class can **implement** multiple interfaces.
 An interface can **extend** multiple interfaces.

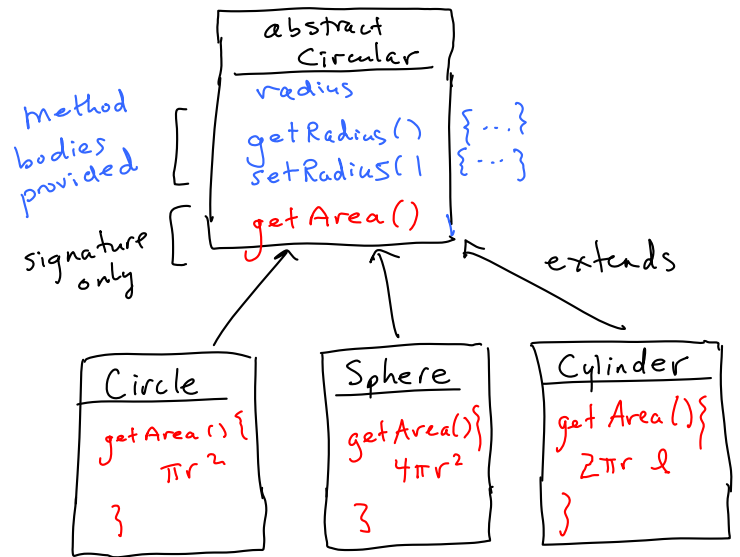
Example : Circular



Awkward : radius, getRadius(), setRadius() are redefined, yet identical.

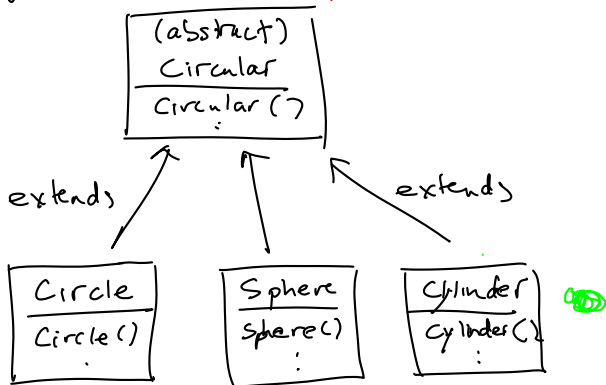
Abstract class

- like a class, it can have reference variable fields, methods with bodies
- like an interface, it can have (abstract) methods with only signatures



Abstract classes have no instances.

BUT abstract classes do have constructors.
Why?



```

    abstract class Circular {
    private double radius
    Circular() { }
    Circular(double radius) {
      this.radius = radius;
    }
    void setRadius(double radius) {
      this.radius = radius;
    }
    void getRadius() {
      return radius;
    }
    abstract double getArea();
  }
  
```

```

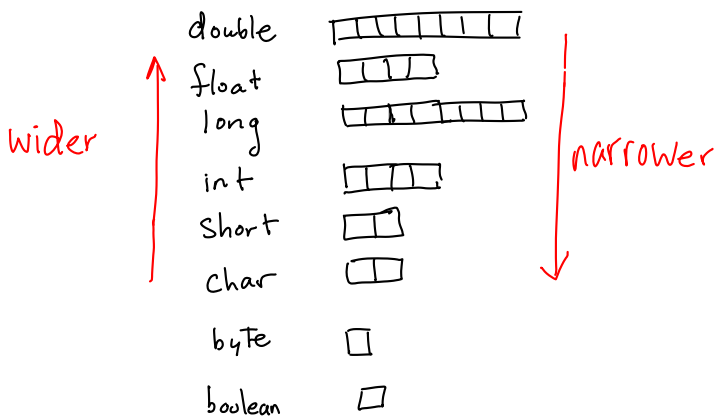
class Sphere extends Circular {
    Sphere() {}
    Sphere(double radius) {
        this.radius = radius;
    }
    double getArea() {
        return 4 * Math.PI *
            power(this.radius,
                2);
    }
}

```

lecture 30

- interfaces
- abstract classes
- casting

Primitive Type Conversion



```

int i = 3;
double d = 4.0;
d = i; // widening (implicit casting)

d = 4.3 * i; // "
            // (called "promotion")

i = (int) d; // narrowing (explicit casting required)
f = (float) d; //

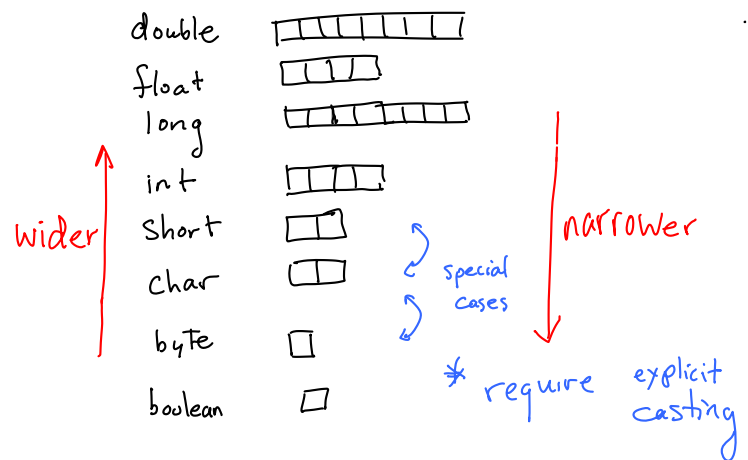
```

```

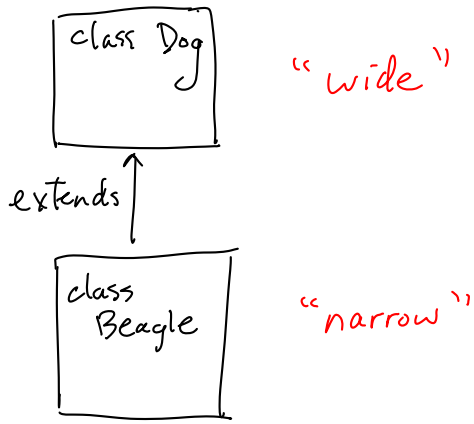
double d = 4.3;
float f = d; // compiler error
              (narrowing requires an explicit cast)

float f = (float) d;
long l = f; // compiler error
long l = (long) f;

```



Note: wider/narrower does not always correspond to relative number of bytes



Careful: Beagles have more bytes than Dogs

// "upcast", automatic (widening)

```
Dog myDog = new Beagle()
```

// "downcast", not automatic (narrowing)

```
Beagle myBeagle = (Beagle) myDog
```

↑
otherwise, compiler will give error

```
Dog myDog = new Beagle()
myDog.hunt()
```

// compiler error - hunt() is defined only in Beagle class

```
((Beagle) myDog).hunt()
```

// runtime error if myDog is a Poodle

```
Dog myDog = new Beagle()
```

```
if (myDog instanceof Beagle)
```

```
((Beagle) myDog).hunt()
```

```
else
```

```
.....
```