lecture 24

- interfaces
- abstract classes
- casting

Interfaces (review)

- `Shape`
  - `getArea()`
  - `getPerimeter()`

- `Rectangle`
  - `float width`
  - `getArea()`
  - `getPerimeter()`

- `Circle`
  - `float radius`
  - `getArea()`
  - `getPerimeter()`

`implements` `extends`

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

Example: Circular

Circle  Sphere  Cylinder

`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

```java
abstract class Circular {
    double radius; // field
    Circular(double radius){ // constructor
        this.radius = radius;
    }
    double getRadius(){ // implemented methods
        return radius;
    }
    void setRadius(double r){
        this.radius = r;
    }
    abstract double getArea(); // abstract method
}
```

class Circle extends Circular{
    Circle(double radius){ // constructor
        super(radius);
    }
    double getArea(){
        double r = this.getRadius();
        return Math.PI * r*r;
    }
}
class Cylinder extends Circular{
    double height;
    Cylinder(double radius, double h){ // constructor
        super(radius);
        this.height = h;
    }
    double getArea(){
        double r = this.getRadius();
        return 2 * Math.PI * r * height;
    }
}
```

lecture 24

- interfaces
- abstract classes
- casting

Primitive Type Conversion

Wider

narrower

Note: wider/narrower does not always correspond to relative number of bytes.
int i = 3;
double d;
d = i; // widening (implicit)
d = 4.3 * i; // promotion
i = (int) d; // narrowing
f = (float) d; // (explicit casting required, otherwise compiler error)

// upcast/widening (automatic, implicit)
Dog myDog = new Beagle();

// downcast/narrowing (not automatic, must be explicit otherwise, compiler will give error)
Beagle myBeagle = (Beagle) myDog

Dog myDog = new Beagle();
myDog. hunt();
// compiler error - hunt() is defined only in Beagle class
((Beagle) myDog). hunt();
// No compiler error but
// runtime error if myDog is a Poodle

Casting a primitive type (up or down)
leads to a change in representation
(int vs float vs double, etc)
and sometimes an approximation!

Casting a reference type does not
lead to a change in representation.
(Dog vs Beagle)
Example:

```
class Dog extends Animal {
    void bark() {
        println("woof");
    }
}

class Beagle extends Dog {
    void bark() {
        println("aauuuuuuuu");
    }
}
```

```java
dog = new Beagle("Buddy")
dog.bark() // "aauuuuuu"
```

**Polymorphism** *(next week)*

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

Compile time
- myDog can be assigned a Dog or any subtype of Dog (subclass)

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