COMP 250

Lecture 31

abstract classes, type conversion

Nov. 23, 2016
RECALL: interfaces

interface Shape

float getArea()
float getPerimeter():

implements
class Rectangle

Rectangle() { ...
float getArea() { .... }
float getArea() { .... }

implements
class Circle

Circle() { ...
float getArea() { .... }
float getArea() { .... }
classes
(tree, parent links only)

interfaces
A subclass can extend one superclass.

A class can implement multiple interfaces.

An interface can extend multiple interfaces.
Example: Circular

Circle     Sphere     Cylinder
Can we avoid repeating these method definitions?
Abstract Class

• Like a class, it can have fields and methods with bodies

• Like an interface, it can have methods with only signatures.
abstract class Circular

double radius

double getRadius() { return radius; }

void setRadius(double r) { radius = r; }

abstract double getArea()

---

class Circle

Circle(double radius) { ... }

double getArea() { ... }

class Sphere

Sphere(double radius) { ... }

double getArea() { ... }

class Cylinder

double length

Cylinder(double radius, double len) { ... }

double getArea() { ... }
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);     // superclass field
    }

    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 * radius * height;
    }
}
}
abstract class Shape

abstract double getArea() { return 0; }

class Circle
Circle(double radius) { ... }
double getArea() { ... }

class Square
Square(double width) { ... }
double getArea() { ... }

class Triangle
Triangle(double height, double base) { ... }
double getArea() { ... }

MY BAD Example: Assignment 4
It should have been:

```java
interface Shape
{
    double getArea();
}

class Circle
{
    Circle(double radius){ ... }
    double getArea() { ... }
}

class Square
{
    Square(double width) { ... }
    double getArea() { ... }
}

class Triangle
{
    Triangle(double height, double base){ ... }
    double getArea() { ... }
}
```
COMP 250

Lecture 31

abstract classes, type conversion

Nov. 23, 2016
Primitive Type Conversion

double
float
long
int
short
char
byte
boolean

In COMP 273, you will learn exactly how these number representations are related to each other.

But you should have some intuitive ideas....

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
**Primitive Type Conversion**

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

Wider usually (but not always) means more bytes.
Examples

```java
int i = 3;
double d = 4.2;
d = i;                 // widening
```
Examples

```c
int i = 3;
double d = 4.2;
d = i;          // widening

d = 5.3 * i;   // widening (by "promotion")`
Examples

```plaintext
int i = 3;
double d = 4.2;
d = i; // widening

d = 5.3 * i; // widening (by "promotion")
i = (int) d; // narrowing (by casting)
float f = (float) d; // narrowing (by casting)
```
Examples

```java
int i = 3;
double d = 4.2;
    d = i; // widening
    d = 5.3 * i; // widening (by "promotion")
i = (int) d; // narrowing (by casting)
float f = (float) d; // narrowing (by casting)

char c = 'g';
int index = c; // widening
    c = (char) index; // narrowing
```

For narrowing conversions, you get a compiler error if you don’t cast.
Widening or narrowing *sometimes* produce an approximation. When?

Primitive type conversion causes the bit representation of the number to change. How?

*(Covered in first month of COMP 273.)*
Heads up! Although Beagle (subclass) is narrower, it has more bytes than Dog (superclass).
Dog   myDog   =   new Beagle();

   // upcast, widening

This is similar to:

double   myDouble   =   3;       //   from int to double.
Dog       myDog = new Beagle();       // Upcasting.

Poodle    myPoodle = myDog;          // what happens?
Dog    myDog = new Beagle();    // Upcasting.

Poodle myPoodle = myDog;    // Compiler error.

    // implicit downcast Dog to Poodle not allowed.

myDog.show()
Dog      myDog = new Beagle();   // Upcasting.

Poodle   myPoodle = myDog;       // Compiler error.
/// implicit downcast Dog to Poodle not allowed.

myDog.show()                            //  Compiler error.
/// Poodle has show() method,  
/// but Dog  does not.
Dog   myDog = new Beagle();     // Upcasting.

Poodle  myPoodle = (Poodle) myDog;

    // ??
Dog    myDog = new Beagle();       // Upcasting.

Poodle myPoodle = (Poodle) myDog;

    // allowed by compiler

myPoodle.show()       // ????
Dog myDog = new Beagle();  // Upcasting.

Poodle myPoodle = (Poodle) myDog;

   // allowed by compiler

myPoodle.show();  // allowed by compiler
  // What happens at runtime?
Dog    myDog = new Beagle();       // Upcasting.

Poodle myPoodle = (Poodle) myDog;

    //    allowed by compiler

myPoodle.show()             //    allowed by compiler
    //    Runtime error:  Dog object
    //    does not have show() method

((Poodle) myDog).show()

    //    ?????
Dog  myDog = new Beagle();  // Upcasting.

Poodle  myPoodle = (Poodle) myDog;

    //  allowed by compiler

myPoodle.show();  //  allowed by compiler
    //  Runtime error:  Dog object
    //  does not have show() method

((Poodle) myDog).show();
    //  both allowed,  but will generate runtime error
    //  if  actual object doesn’t have a show method.
How to avoid such runtime errors?

if (myDog instanceof Poodle){
  ( (Poodle) myDog ).show();
}

if (myPoodle instanceof Poodle){
  myPoodle.show();
}