CSE 120/130

Introduction to Programming Languages and Techniques

Fall 2000

Handout 8

Java and the Homework

- The homework focuses on the Classes you need to add/modify
- We're going to look at how those classes are used with the `Shapes` interface. This will review earlier lectures in the context of the homework, and therefore provide further understanding of what you are doing for the homework.
- Some details on the coordinate system that will be useful for doing the homework.
- The due date *is* Wednesday, December 6.
- A sample solution is up on the homeworks page
- An announcement:

```
The Dining Philosophers is holding an end-of-semester

Study Break

This Sunday, December 3rd,
from 5:30 to 7pm
in the Moore Lounge.

There will be food, including hoagies, wings,
and other good stuff from Lee's.

We're going to be raffling away some Microsoft Development
Software, and possibly some programming books.

The event is kindly sponsored by iPhrase Technologies.
```


Overview of hw10.java

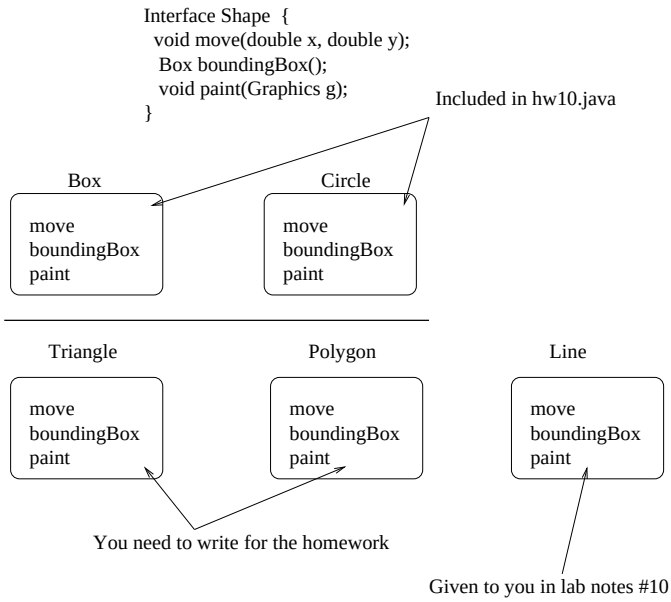
- The details of how the code is interfaced to a web document are done for you. This is "event-driven" programming - there is no `main` method in the code!
- Four Parts:
  - The definition of the `Shape` interface
  - The definition of the Classes that implement `Shape`: `Box` and `Circle` are given to you, based on the ones shown before in the lecture notes. You need to write two more, `Triangle` and `Polygon`. Note that in handout 7, `Box` did not implement `Shape`. Instead it was a Class used only to return the boundingBox. Now it is used for both purposes.
  - A Class `Student` which handles various actions like moving the shapes and what happens when a button is pressed. `Student` uses the interface `Shape` to communicate with the Classes `Box` and `Circle`. You need to make some modifications to this.
  - Class `hw10` - do not touch! This handles the GUI (graphical user interface) and you do not need to understand it for this homework.

The `Shape` Interface

```
interface Shape {
  void move(double x, double y);
  Box boundingBox();
  void paint(Graphics g);
 // we got rid of expand, and added paint
}
class Box implements Shape {
...
}
class Circle implements Shape {
...
}
```

- The `Box` and `Circle` Classes "implement" `Shape`, which means they must have methods matching the `Shape` specification.
- These methods will be called by the `Student` class. `Box` and `Circle` are *subtypes* of the interface `Shape`. (see also handout 7, p. 51).

Different Classes that Implement the `Shape` Interface



The `Shape` Interface - A Closer Look

```
interface Shape {
  void move(double x, double y);
  Box boundingBox();
  void paint(Graphics g);
}
```

- `move`: as discussed before
- `boundingBox`: After the user selects a button, the GUI passes coordinates indicating the mouse-click(s) to the `Student` object. The `Student` object uses `bounding Box()` to determine what objects were referenced by the mouse-click(s). This indicates which objects to move, and will also be used by your delete code to indicate which objects to delete.
- `paint`: g is part of the "magic", specifying an area to paint. The code for `Box` and `Circle` each determine how objects of that Class will be painted.

Review: Objects and Classes - A Rough Analogy

You can't do anything with `int` except use it to declare a variable:

```
int i=5;
```

The primitive types `int`, `float`, etc. can be considered 'templates' that are used to define variables. The `int` type tells Java that the size of the `int` variable is 4 (or whatever) bytes long and is treated as an interger, `float` is 8 (or whatever) bytes long and is treated as a float, etc.

- Classes are analogous to types, and objects to variables.
- Classes can be considered 'templates' used to define objects just as types are 'templates' used to define objects.

An Example:

`String` is actually a Class, not a primitive type. The statement:

```
String a = "test";
String b = "this!";
```

is actually short for:

```
String a = new String("test");
String b = new String("this!");
```

String is the class, and `a` and `b` are two Objects of the Class `String`

How the `Student` Class communicates with the Shapes

The `Student` Class keeps an array of `Shape` objects, using `count` to keep track of how many there are (similar to what you did for implementing a `Stack` in the last homework):

```
Shape[] shapes = new Shape[10000];
int count = 0;
```

- Recall (handout 7, slide 46) that Interfaces, like Classes, can be used as the types of variables.
- The `Shape` array is used to store objects that are `Box`es or `Circle`s. There can be any number of `Box`es or `Circle`s each, up to 10000.

The Student Class has a method `addShape`:

```
public void addShape (Shape s) {
  shapes[count] = s;
  count++;
}
```

and is called within Student in different ways, depending on what subtype of `Shape` is being created:

```
...
addShape(new Box(minx, miny, maxx-minx, maxy-miny));
...
addShape(new Circle(x[0], y[0], radius));
```

(these parameters indicate where the new `Box` or `Circle` is to be placed. We'll explain shortly where these come from)

The code in the `Student` Class uses the fact that each object in the `shapes` array must have method `boundingBox` and `move`, since they are all objects that are subtypes of `Shape`

```
// dx is the change in the x coordinate
// dy is the change in the y coordinate
// x[0],y[0] are the coordinates of the mouse click
for (int i = 0; i < count; i++) {
  if (shapes[i].boundingBox().contains (x[0], y[0])) {
     shapes[i].move(dx, dy);
  }
}
```

For each shape, if the bounding box of the shape contains the mouse click, then move that object.

You will do something very similar for the delete button in problem 3 of the homework. For each shape, if the bounding box of the shape contains the mouse click, then delete that object. It's deleted by adjusting the elements of the `shapes` array accordingly.

The code in the `Student` Class also uses the `Shape` interface for the `paint` method:

```
public void paint (Graphics g) {
  g.setColor(Color.red);
  for (int i = 0; i < count; i++) {
    shapes[i].paint(g);
  }
}
```

The `Student.paint` method is called by the GUI when appropriate. In turn, it goes through each shape in the `shapes` array and calls its `paint` method.

You do not need to modify the code `paint` in the `Student` class. However, for each of the two Classes you write, `Triangle` and `Polygon`, you need to write a paint method to display them properly.

The easiest way to do this is to look at the `paint` code for `Box`, `Circle`, and `Line`(from the lab notes #10), and imitate, along with an understanding of how the coordinate system works.

But first, a question

Type-casting and Interfaces

While objects that are subtypes of `Shape` must have the three methods specified in `Shape`, they of course can have others. For example, objects of Class `Box` have a method `contains`, which was used in the previous slide. But consider:

```
if (shapes[0].contains (x[0], y[0])) {...
```

The compiler will complain:

```
t.java:36: Method contains(double, double) not found in interface Shape.
        if (shapes[0].contains(1.0,2.0)) {
```

While it's possible that shapes[0] may be a `Box` object, it could also be a `Circle` or any other subtype of `Shape`. You could instead *cast* it to be of type `Box`:

```
if (((Box)shapes[0]).contains (x[0], y[0])) {...
```

This is a declaration to the typechecker that we "know" that the actual type of the object stored in `shapes[0]` at runtime will be `Box`. At compile time, the typechecker just believes what we tell it, and at runtime it double-checks that it really is a `Box`.

```
if (((Box)shapes[0]).contains (x[0], y[0])) {...
```

Contrast this to the earlier:

```
if (shapes[i].boundingBox().contains (x[0], y[0])) {...
```

Why the difference?

How You Deal with the Coordinate System

- The issue: Different operations require different amounts of information from the mouse. To create a `Box`, you need two sets of (x,y) coordinates. To create a `Polygon`, there can be any number of (x,y) coordinates, as long as there's at least one. To do a `delete`, only one (x,y) coordinate is needed.
- The GUI keeps track of where the user clicks the mouse. It accumulates the coordinates of each mouse click in two arrays, for the x and y coordinates.
- The method `handleButton` in the `Student` Class receives this information from the GUI. It is up to you to extend this method for the proper handling of the new classes and methods that you need to write.

```
public void handleButton (String name, double[] x, double[] y) {
...
}
```

- `(x[0],y[0])` is the position of the first mouse click, `(x[1],y[1])` is the position of the second mouse click, etc. `name` is the name of the button which was pressed(e.g., "Box", "Move")

How `handleButton` for "Box' works:

```
public void handleButton (String name, double[] x, double[] y) {
  if (name == "Box" && x.length >= 2) {
    double minx = Math.min(x[0],x[1]); double maxx = Math.max(x[0],x[1]);
    double miny = Math.min(y[0],y[1]); double maxy = Math.max(y[0],y[1]);
    addShape(new Box(minx, miny, maxx-minx, maxy-miny));
```

The constructor for `Box` expects the coordinates of the lower left corner, together with the width and height:

```
Box(double x, double y, double w, double h){
  xpos = x; ypos = y; wid = w; ht = h;
}
```

Example: two mouse clicks at (3,4), (6,6). `handleButton` receives the arguments `x` and `y` with

```
x[0]=3, [1]=6
y[0]=4, [1]=6
```

`handlebutton` makes the call `new Box(3,4,3,2)`

Continuing the Example: Inside the `Box` Object

`handlebutton` just made the call

```
new Box(3,4,3,2)
```

This will set up the instance variables (fields) of the new `Box` object appropriately:

```
class Box implements Shape {
  double xpos;  // coordinates of corner
  double ypos;
  double wid;   // width
  double ht;    // height

  Box(double x, double y, double w, double h){
    xpos = x; ypos = y; wid = w; ht = h;
  }
```

```
class Box implements Shape {
  double xpos;  // coordinates of corner
  double ypos;
  double wid;   // width
  double ht;    // height

  Box(double x, double y, double w, double h){
    xpos = x; ypos = y; wid = w; ht = h;
  }
```

But in the `paint` method for `Box`, these need to be *cast* from `double` to *int* (as the homework says to do for `Triangle` and `Polygon`):

```
public void paint (Graphics g) {
  g.drawRect ((int)xpos, (int)ypos, (int)wid, (int)ht);
}
```

This is the same basic idea as the other use of casting, but here the compiler will force `xpos`, etc. to be integers no matter what, even by truncating any non-integer part.