SE 450 Winter 2002
Final Exam

March 19, 2002

Name: _____

Directions: Write your name in the blank at the top of this page.
You have **two hours and thirty minutes** (2:30) to take this exam.
It is **closed book and closed notes**.
Write your answers **on the exam**.
There are 14 questions, some with mutiple parts.
If I can not read your answer clearly, then it will be marked as incorrect.

Question	Value	Score
1	8	
2	8	
3	7	
4	7	
5	7	
6	7	
7	7	
8	7	
9	7	
10	7	
11	7	
12	7	
13	7	
14	7	
Total	100	

Question 1

What is the output of the following Java code?

```
interface I {
  String world();
}

abstract class C implements I {
  public String hello() { return "Hello "; }
  abstract public String world();
}

class D extends C {
  public String hello() { return "World "; }
  public String world() { return super.hello() + hello(); }
}

class E extends D {
  public String hello() { return "Mundo "; }
}

public class Main {
  public static void main(String[] args) {
    I x = new D(); System.out.println(x.world());
    I y = new E(); System.out.println(y.world());
  }
}
```

Question 2

What is the output of the following Java code?

Briefly justify your answer.

```
class MyBool {
  private boolean _v;
  public MyBool(boolean v) { set(v); }
  public void set(boolean v) { _v = v; }
  public boolean get() { return _v; }
  public boolean equals(Object that) {
    return (that instanceof MyBool)
        && (this.get() == ((MyBool) that).get());
  }
}

public class Main {
  public static void main(String[] args) {
    MyBool x = new MyBool(false);
    MyBool y = x;
    System.out.println( (x.equals(y)) + "," + (x == y) );
    y.set(true);
    System.out.println( (x.equals(y)) + "," + (x == y) );

    MyBool u = new MyBool(false);
    MyBool v = new MyBool(false);
    System.out.println( (u.equals(v)) + "," + (u == v) );
    v.set(true);
    System.out.println( (u.equals(v)) + "," + (u == v) );
  }
}
```

Question 3

Consult the notes starting on page 16.

Write correct implementations of the `equals` and `clone` methods in `Circle`.

```
class Circle implements Shape { // ...
  public boolean equals(Object o) {









  }
}
```

```
class Circle implements Shape { // ...
  public Object clone() {









  }
}
```

Question 4

`ComplexShape` is used to represent aggregate shapes, which are made up from other shapes.

To achieve this, `ComplexShape` uses an `ArrayList` called `_shapes`. You may wish to consult the interface of `ArrayList` described on page 18.

Write correct implementations of the `equals` and `clone` methods in `ComplexShape`.

```
class ComplexShape implements Shape { // ...
  public boolean equals(Object o) {









  }
}
```

```
class ComplexShape implements Shape { // ...
  public Object clone() {









  }
}
```

Question 5
Write correct implementations of the `addShape` and `draw` methods in `ComplexShape`.

```
class ComplexShape implements Shape { // ...
  /** add a shape to the composite */
  public void addShape(Shape s) {









  }
}
```

```
class ComplexShape implements Shape { // ...
  /** draw all shapes in the composite */
  public void draw(Color c) {









  }
}
```

Question 6
Write a fragment of Java code — using the classes `Circle`, `Rectangle` and `ComplexShape` — to create a complex shape object named "`myShape`" that contains:

- A circle of radius 4 with center at (0,0).
- A circle of radius 6 with center at (0,0)
- A square of side 10 with lower left corner at (0,0)

```
Shape myShape =
```

Question 7
This question has two parts.

(a) Draw a UML class diagram showing the relationships among `Shape`, `Circle`, `Rectangle` and `ComplexShape`.

(b) Name the design pattern most closely associated with these classes.

Question 8

We now focus on creating shapes at locations other than the origin. The class `Translate` below is intended to help in describing such objects. For example, consider the object consisting of:

- A circle of radius 3 at center (0,0)
- A circle of radius 5 at center (2,3)

Such an object might be constructed using the following code:

```
Circle c1 = new Circle();
c1.setRadius(3);

Circle c2 = new Circle();
c2.setRadius(5);

Translate t = new Translate();
t.setXY(2,3);
t.setShape(c2);

ComplexShape o = new ComplexShape();
o.addShape(c1);
o.addShape(t);
```

Write a correct implementation of the `draw` method in `Translate`. Hint: Move cursor by amount `(_x,_y)`, then draw, then restore the cursor to its original position

```
class Translate implements Shape { // ...
  public void draw(Color c) {









  }
}
```

Question 9

Write a fragment of Java code — using the classes `Circle`, `Rectangle`, `Translate` and `ComplexShape` — to create a complex shape object named "`myShape`" that contains:

- A circle of radius 3 with center at (0,0).
- A circle of radius 5 with center at (3,4).
- A complex shape that consists of a circle of radius 3 at the (8,9) and a circle of radius 5 with center (10,12)

```
Shape myShape =
```

Question 10
This question has two parts.

(a) Draw a UML class diagram showing the relationships among `Shape`, `Circle`, `Rectangle` and `Translate`.

(b) Name the design pattern most closely associated with these classes.

Question 11

Write the code for a `PrintVisitor` that will print out a `Shape`. For example, if we take `o` from the example given in Question 8, then

```
o.accept(new PrintVisitor());
```

should output:

```
ComplexShape(Circle(3), Translate(2, 3, Circle(3)))
```

In your answer, use `System.out.print(String)`.

```
class PrintVisitor implements ShapeVisitor {









}
```

Question 12
In this question, we add a `ShapeFactory` to isolate the creation of new `Shape` objects. For example:

```
Shape c = ShapeFactory.newShape("circle");       \\ creates a circle
Shape r = ShapeFactory.newShape("rectangle");    \\ creates a rectangle
Shape t = ShapeFactory.newShape("translate");    \\ creates a translate
Shape o = ShapeFactory.newShape("complexShape"); \\ creates a complexShape
```

Write the `newShape` method in the `ShapeFactory` class. You can assume that the string parameter is one of `"circle"`, `"rectangle"`, `"translate"` or `"complexshape"`.

```
class ShapeFactory {
  static Shape newShape(String shapeKind) {









}
```

Question 13
This question has two parts.

(a) Draw a UML class diagram showing the relationships among `Shape`, `Circle`, `Rectangle` and `ShapeFactory`.

(b) Name the design pattern most closely associated with these classes.

Question 14
Write an implementation of the `hashCode` method for the classes `Circle` and `ComplexShape`. For your convenience, here is the contract for `hashCode`, copied from the `java.lang.Object` specification:

> Whenever it is invoked on the same object more than once during an execution of an application, the `hashCode` method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
>
> If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
>
> It is *not* required that if two objects are unequal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

Here is the (relevant portion of the) recipe to construct good hashcodes from Bloch's book.

- Store some constant nonzero value, say 17, in an `int` variable called `result`.
- For each significant field `f` in your object (each field taken into account by the equals method, that is), do the following:
  - Compute an `int` hash code `c` for the field as follows:
    - If the field `f` is a base type, compute `(int)f`.
    - If the field `f` is an object reference, recursively invoke `f.hashCode()`. If the value of the field is `null`, return `0`.
    - If the field `f` is an array, treat it as if each element were a separate field. That is, compute a hash code for each significant element by applying these rules recursively, and combine these values as described below.
  - Combine the hash code `c` computed in the previous step into result as follows:

    ```
    result = 37*result + c;
    ```
- Return `result`.

Write your answers on the following page.

```
class Circle implements Shape { // ...
  public int hashCode() {









  }
}

class ComplexShape implements Shape { // ...
  public int hashCode() {









  }
}
```

Most of this exam concerns the construction of a simple 2-D graphics library, which uses the following utility class.

```
class BasicGLib {
  /** draw a circle of color c with center at current cursor position
   *  the radius of the circle is given by radius
   */
  public static void drawCircle(Color c, int radius) {/*...*/}

  /** draw a rectangle of Color c
   *  with lower left corner at current cursor position
   *  the length of the rectangle along the x axis is given by xlength
   *  the length of the rectangle along the y axis is given by ylength
   */
  public static void drawRect(Color c, int xlength, int ylength) {/*...*/}

  /** move the cursor by coordinate (xcoord,ycoord) */
  public static void moveCursor(int xcoord, int ycoord) {/*...*/}

  /** clear the entire screen and set cursor position to (0,0) */
  public static void clear() {/*...*/}
}
```

For example:

```
  BasicGLib.clear();                    // initialize
  BasicGLib.drawCircle(Color.red, 3);   // a red circle: radius 3, center (0,0)
  BasicGLib.drawRect(Color.blue, 3, 5); // a blue rectangle: (0,0),(3,0),(3,5),(0,5)
  BasicGLib.moveCursor(2, 2);           // move cursor
  BasicGLib.drawCircle(Color.green, 3); // a green circle: radius 3, center (2,2)
  BasicGLib.drawRect(Color.pink, 3, 5); // a pink rectangle: (2,2),(5,2),(5,7),(2,7)
  BasicGLib.moveCursor(-2, -2);         // move cursor back to (0,0)
```

In this exam, you will write code to build and manipulate complex `Shape` objects built out of circles and rectangles. An outline of the code is given on the following page.

```
import java.awt.Color; import java.util.ArrayList;
interface Shape extends Cloneable {
  public void draw(Color c);
  public void accept(ShapeVisitor v);
  public Object clone();
}
interface ShapeVisitor {
  public void visitCircle(int r);
  public void visitRectangle(int x, int y);
  public void visitComplexShape(ArrayList shapes);
  public void visitTranslate(int x, int y, Shape s);
}
class Circle implements Shape {
  private int _r;
  public void setRadius(int r)    { _r = r; }
  public int getRadius()          { return _r; }
  public void draw(Color c)       { BasicGLib.drawCircle(c, _r); }
  public Object clone()           { return null; }
  public boolean equals(Object o) { return false; }
  public void accept(ShapeVisitor v) { v.visitCircle(_r); }
}
class Rectangle implements Shape {
  private int _x, _y;
  public void setXY(int x, int y) { _x = x; _y = y; }
  public int getX()               { return _x; }
  public int getY()               { return _y; }
  public void draw(Color c)       { BasicGLib.drawRect(c, _x, _y); }
  public Object clone()           { return null; }
  public boolean equals(Object o) { return false; }
  public void accept(ShapeVisitor v) { v.visitRectangle(_x,_y); }
}
class ComplexShape implements Shape {
  private ArrayList _shapes = new ArrayList();
  public void addShape(Shape s)   { }
  public void draw(Color c)       { }
  public Object clone()           { return null; }
  public boolean equals(Object o) { return false; }
  public void accept(ShapeVisitor v) { v.visitComplexShape(_shapes); }
}
class Translate implements Shape{
  private int _x, _y;
  private Shape _s;
  public void setXY(int x, int y) { _x = x; _y = y; }
  public void setShape(Shape s)   { _s = s; }
  public int getX()               { return _x; }
  public int getY()               { return _y; }
  public Shape getShape()         { return _s; }
  public void draw(Color c)       { }
  public Object clone()           { return null; }
  public boolean equals(Object o) { return false; }
  public void accept(ShapeVisitor v) { v.visitTranslate(_x,_y,_s); }
}
```

`ArrayList` is defined in part:

```
public class ArrayList { /*...*/
  public ArrayList() {/*.../*}
  public int size() {/*.../*}
  public boolean isEmpty() {/*.../*}
  public Object clone() {/*.../*}
  public boolean equals(Object o) {/*.../*}
  public Object get(int index) {/*.../*}
  public boolean add(Object o) {/*.../*}
  public void clear() {/*.../*}
}
```