Spring 2011 CS151 Midterm II

Instructor: Dr. Kim

Maximum obtainable score: 43 points

41.5

4 **(4 pts)** Q1. Briefly explain the major role of abstract classes in object-oriented design.

Abstract classes allow both concrete and abstract methods as well as instance variables. They allow more flexibility than interfaces but still cannot be instantiated on their own.

7.5 **(8pts)** Q2. Consider a window in a windowing system. Suppose the MyWindow class represents a Window without any functionality for adding scrollbars and you want to allow scrolling of the window's content. (Note that this question is **not** about GUI programming.)

```
// the Window interface
interface Window
{   /** returns a string representation of this window */
    public String toString();
}

// Implementation of a Window without any scrollbars
class MyWindow implements Window
{
    public String toString()
    { return "Draw this window"; }
}
```

1) Which design pattern is suitable to solve the problem ?

Iterator Observer Strategy Composite (Decorator) Template Method

2) Write one class required to enhance a window object with a scrollbar so that the string representation of the enhanced window with a scrollbar becomes "Draw this window *with a scrollbar*". Follow the design pattern you chose in the part 1). Credit will be given only if you chose a correct answer for the part 1).

0.5

```
class ScrollWindow implements Window
{
  private MyWindow window;
  public ScrollWindow(MyWindow window){this.window = window;}
  public String toString(){return window.toString() + "with a Scrollbar";}
}
```

3) Write a test program that creates an instance of the enhanced window object and prints its string representation.

```
class WindowTester
{
  public static void main(String[] args)
  { ScrollWindow myScrollWindow = new ScrollWindow(
        new MyWindow());
    System.out.println(myScrollWindow);
  }
}
```

11.5

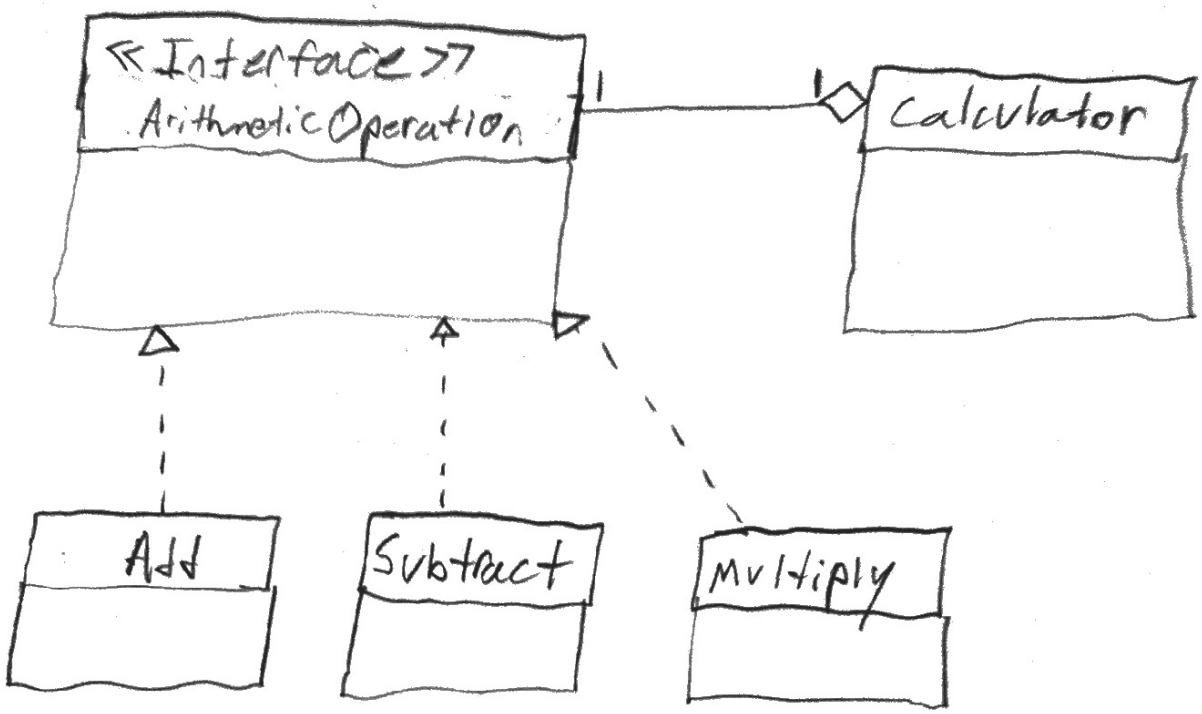
(7pts) Q3 Consider the following program to answer this question.

```
public class Tester {

    public static void main(String[] args) {

        Calculator calculator;

        calculator = new Calculator(new Add());
        int resultA = calculator.execute(3,4);

        calculator = new Calculator(new Subtract());
        int resultB = calculator.execute(3,4);

        calculator = new Calculator(new Multiply());
        int resultC = calculator.execute(3,4);

        System.out.println(resultA + " " + resultB + " " + resultC); //7 -1 12
    }
}

interface ArithmeticOperation { int evaluate(int a, int b); }

class Add implements ArithmeticOperation
{   public int evaluate(int a, int b)
    {    return a + b;  }
}

class Subtract implements ArithmeticOperation
{   public int evaluate(int a, int b)
    {    return a - b;  }
}
class Multiply implements ArithmeticOperation
{   public int evaluate(int a, int b)
    {    return a * b;  }
}
class Calculator {

    private ArithmeticOperation operation;

    public Calculator(ArithmeticOperation op)
    {  this.operation = op;}

    public int execute(int a, int b) {
        return operation.evaluate(a, b);
    }
}
```

(1) Which design pattern is used to design this application ? (Circle one.)

Iterator Observer (Strategy) Composite Decorator Template Method

(2) Draw a class diagram to depict the design pattern you selected for this question. You are required to specify class names, their relationships, and multiplicities for any aggregation. **Do not copy the general class diagram from the book. The class diagram should be specific to this application.**



7

(7pts) Q4. The Game Interface defines common operations to several games in which players play against the others, but only one is playing at a given time. Chess and Monopoly can be concrete classes that implement the Game interface. Suppose the implementations of initializeGame, makePlay, and endOfGame methods are specific to a concrete. However, the algorithm to perform playOneGame is applicable for any game and will be invariant. The Game interface and algorithm are as follows:

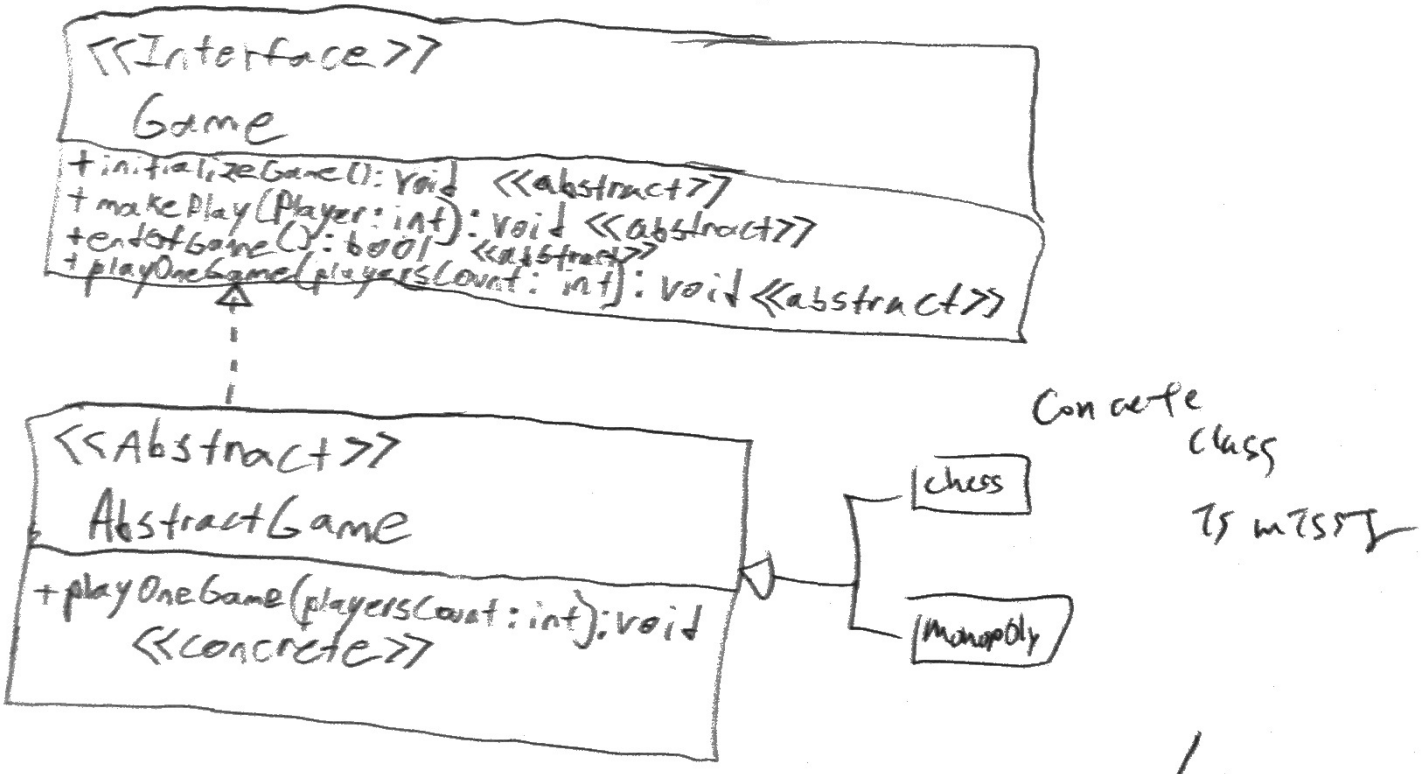
```
public interface Game
{
  void initializeGame();
  void makePlay(int player);
  boolean endOfGame();
  void playOneGame(int playersCount);
}
```

```
void playOneGame(int playersCount)
{   initializeGame();
    int j = 0;
    while( ! endOfGame() )
    {   makePlay( j );
        j  = (j + 1) % playersCount;
    }
}
```

(1) Which design pattern is suitable to solve this problem? (Circle one.)

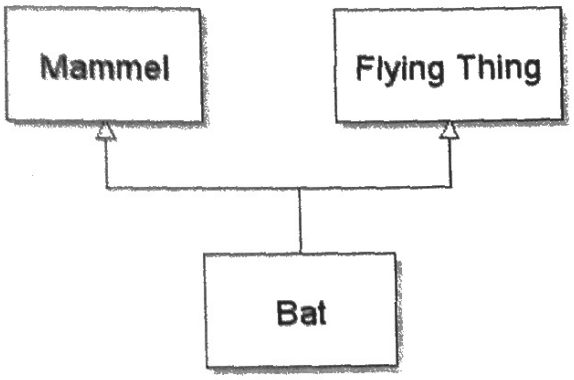
Iterator Observer Strategy Composite Decorator (TemplateMethod)

(2) Draw a class diagram to depict the design pattern you selected in the question. In the class diagram, indicate if the rectangle of the diagram represents an interface, an abstract class, or a concrete class. Also, specify all methods defined in an interface or a class, and indicate they are abstract or concrete. **Do not just copy the class diagram from the book. The class diagram should be specific to this application.**



6.0

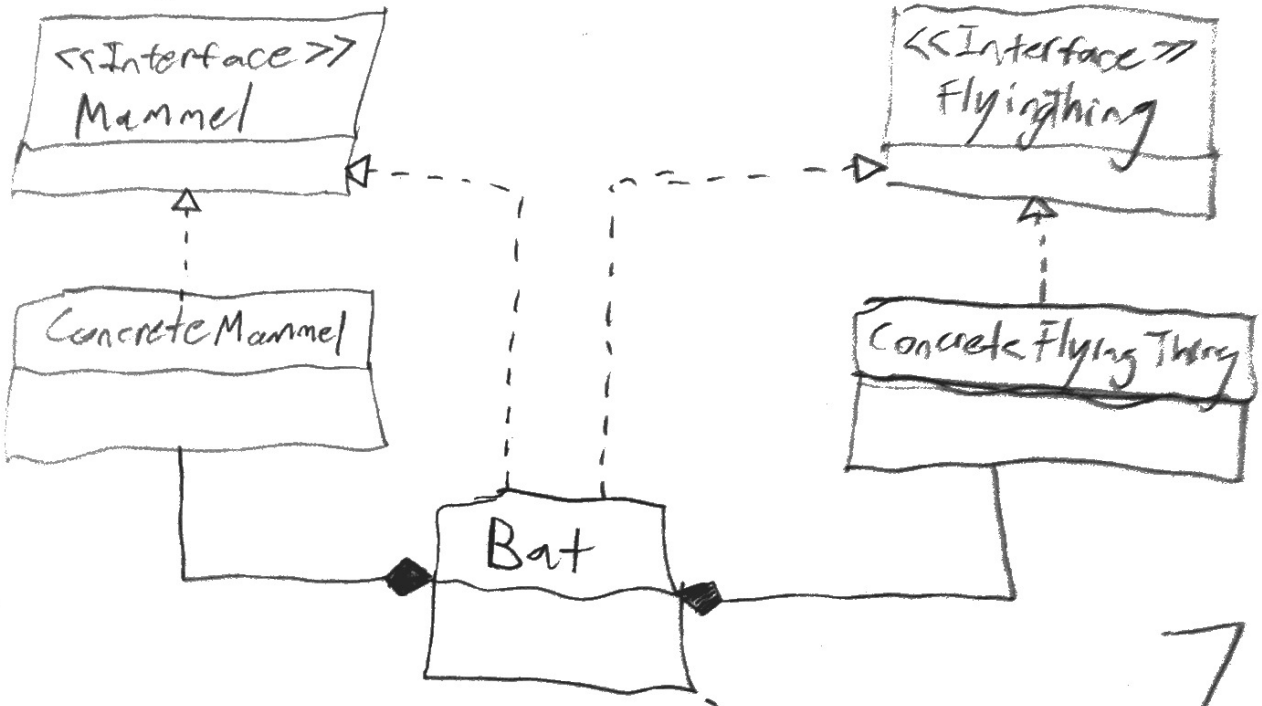
(7pts) Q5. Java does not allow "multiple inheritance". Therefore, the is-a relationships depicted in the following class diagram cannot be implemented in Java.



(a) Briefly explain why Java does not allow multiple inheritance.

In the interest of simplicity in dealing with the Diamond Problem in which a function call is ambiguous because two parents have a function of the same name, Java does not allow multiple inheritance.

(b) Redraw the above class diagram in a way that it depicts a design that removes the problem of multiple inheritance but still expresses multiple is-a relationships. The suggested design should be not have "version control problem" either. (You are expected to know the meaning of version control problem.) You may need to add more rectangle(s) and add/change relationship(s) to the above class diagram.



(10pts) Q6. The following program ShapeSelection draws one circle on the screen. When the user selects the circle by pressing the mouse on the circle, the selection of the circle is toggled. The program fills the circle when it is selected. Only outline is drawn for the unselected shape. If the user presses outside of the circle, nothing is changed. **Write the ShapeComponent class** to complete this application. Assume all required library classes are already imported. You don't have to write any import statement.

API and more

1) Create the circle as follows. Note that Ellipse2D.Double is a Shape.

```
new Ellipse2D.Double(10,10, 50,50);
```

2) To add a mouse listener to this ShapeComponent

```
this.addMouseListener(
            new MouseAdapter()
            {    public void mousePressed(MouseEvent event)
                 {          }
             });
```

Note that addMouseListener is supposed to be inherited from JComponent.

3) `Shape: boolean contains(Point2D p)`
Tests if a specified Point2D is inside the boundary of the Shape. (Point extends Point2D.)

4) Graphics2D

- `public abstract void draw(Shape s)` strokes the outline of a Shape using the settings of the current Graphics2D context.
- `public abstract void fill(Shape s)` fills the interior of a Shape using the settings of the Graphics2D context.

5) MouseEvent

`public Point getPoint()` returns the x,y position of the event relative to the source component.

6) Header of paintComponent: `public void paintComponent(Graphics g)`

```
public class ShapeSelection
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      ShapeComponent scene = new ShapeComponent();

      frame.add(scene, BorderLayout.CENTER);
      frame.setSize(200, 100);
      frame.setVisible(true);
   }
}
// Your answer goes in the next page.
```

```
public class ShapeComponent extends JComponent
{ private final boolean toggleSelected;
  private Ellipse2D.Double ellipse;

  public ShapeComponent()
  {
    toggleSelected = false;
    ellipse = new Ellipse2D.Double(10, 10, 50, 50);
    this.addMouseListener(
        new MouseAdapter()
        {
            public void MousePressed(MouseEvent event)
            {
              if (contains(event.getPoint()))
                toggleSelected = !toggleSelected;
            }
        });
  }
                    could use ellipse.contains
  public boolean Contains(Point2D point)
  {  return .(Math.pow(point.getX() - 35, 2)
            + Math.pow(point.getY() - 35, 2)
            <= Math.pow(25, 2));
  }
  @Override
  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    if(toggleSelected)
      g2.fill(ellipse);
    else
      g2.draw(ellipse);
  }
}
```

7

10