1.00/1.001
Introduction to Computers and Engineering Problem Solving
Spring 2012 - Quiz 2

You have 80 minutes to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary packages have already been imported. You may only add code inside the boxes. The code written outside the boxes may not be altered in any way. Good luck!

<table>
<thead>
<tr>
<th>Question 1</th>
<th>/ 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 2</td>
<td>/ 10</td>
</tr>
<tr>
<td>Question 3</td>
<td>/ 50</td>
</tr>
<tr>
<td>Question 4</td>
<td>/ 30</td>
</tr>
<tr>
<td>Total</td>
<td>/ 100</td>
</tr>
</tbody>
</table>
Question 1 – True/False (10 points)

Circle all answers that are correct.

1. A concrete class that extends an abstract class must implement all abstract methods.

   TRUE  FALSE

2. A class can extend multiple classes.

   TRUE  FALSE

3. A subclass can access _________ fields of its superclass in the same package.

   Public  Protected  Private  Package

4. An abstract class may have

   Public fields  Protected fields  Private fields

   Abstract methods  Concrete methods

5. A final method may only be declared in an abstract class.

   TRUE  FALSE
Question 2 - Recursion (10 points)

The greatest common divisor (GCD) is the largest positive integer that divides two numbers without a remainder. You will write a recursive method to find the GCD of two positive integers using Euclid’s algorithm.

Euclid’s algorithm is:
1. Let p be the larger of two positive integers p and q,
2. Divide p by q. Let integer r be the remainder. If r is zero, then q is the GCD.
3. Else, the GCD of p and q is the GCD of q and r.

```java
public static int gcd(int p, int q) {
    // Your implementation here
}
```
Question 3 - Inheritance (50 points)

In this question, you will model some celestial bodies, including planets and stars.

3.a The Planet constructor accepts a radius, which should set the corresponding data member. In addition to the constructor, the Planet class should have an abstract method calcAtmosphereThickness() that returns the thickness of the Planet's atmosphere (double). Write these two methods in the box below.

```java
public abstract class Planet {
    protected double radius;

} /*End of Planet Class*/
```

3.b There are two kinds of planet: RockyWorld and GasGiant. For modeling purposes, we assume that the atmospheric thickness on a RockyWorld is 1% of its planetary radius (while on a GasGiant it is 80% of its planetary radius). Complete the RockyWorld class below while inheriting as much as possible from its parent class and implementing any necessary methods.

```java
public class RockyWorld extends Planet{

} /*End of RockyWorld Class*/
```
3.c We note that some celestial bodies are gaseous, such as GasGiant planets and all stars. You are given the following Gaseous interface, which contains a method that determines the density of the gaseous material.

```java
public interface Gaseous {
    double getDensity();
}
```

Complete the GasGiant class below, inheriting as much as possible and implementing any necessary methods. The constructor should take two inputs: the Planet’s radius and density. Recall that the atmospheric thickness is 80% of the radius.

```java
public class GasGiant extends Planet implements Gaseous{
    private double density;
}
/*End of GasGiant Class*/
```
3.d  Complete the Star class below, inheriting as much as possible and implementing any additional methods as necessary. The class should have two data members: the Star's density and an array of Planets that revolve around the Star. The constructor should set these data members from arguments. You should also write a getter method for the array of Planets.

```java
public class Star implements Gaseous{

/*End of Star Class*/
```
3.e Add code in the TestUniverse class to model the following:

- Create three Planets, two Rocky and one Gaseous:
  - one called earth with radius = 6400,
  - one called mars with radius = 3400, and
  - one called jupiter with radius = 67000 and density = 1.3

- Place all of the Planets into an array.

- Create a new Star called sun that has density 1.4 and contains the array of Planets.

- Loop through sun's Planets and print the following to the console:
  - each Planet’s atmosphere thickness, and
  - each Planet’s density if it is Gaseous.

```java
public class TestUniverse {

    public static void main(String[] args) {

    }

} /*End of TestUniverse Class*/
```
Question 4 - Swing (30 points)

In this question you will complete a Java program to play Tic-Tac-Toe. Your design will use the model-view-controller pattern. Snapshots of the program running are as follows:

The board is composed of 9 JButton in a 3 by 3 GridLayout. The GridLayout lays out a container's components in a rectangular grid. The code snapshot on the left will result in the construction of the layout illustrated on the right.

```java
setLayout(new GridLayout(3,2));
add(new Button("1"));
add(new Button("2"));
add(new Button("3"));
add(new Button("4"));
add(new Button("5"));
add(new Button("6"));
```

Two players alternate taking turns. The first player uses marker "X" and the second player uses marker "O".
4.a Complete the Controller constructor below. The constructor:

- Initializes cells, the array of JButtons
- Initializes handle, the object that will listen for events. The ClickHandler constructor takes one argument, the Controller object.
- Adds the 9 JButtons to the JFrame, in a GridLayout. All buttons share the same ActionListener named handle, which you will write in part b.

```java
public class Controller extends JFrame{

private JButton[] cells;  // References to 9 JButtons

// When count is even, it is the first player’s turn
// When count is odd, it is the second player’s turn
private int count;

// ClickHandler implements ActionListener
private ClickHandler handle;

public Controller(){

/* End of Constructor */

public void addCount(){ count++; }

public int getCount(){ return count; }

public static void main(String[] args) { // Code not shown}
} /* End of Controller Class */
```
4.b Implement the method to handle button clicks within ClickHandler:

- If a button is already marked when clicked on, the method returns immediately without doing anything.
- Otherwise, a marker is put on the button.
- The first player uses String “X” as a marker and the second player uses String “O” as a marker.
- Use getCount() to tell you which player is making a move; remember to use addCount() after a valid move.
- There are methods void setText(String s) and String getText() in class JButton
- Last, call checkWin() to see if a player has won the game

```java
public class ClickHandler implements ActionListener {
    private Controller control;
    public ClickHandler(Controller c) {this.control = c;}

    private void checkWin() {
        // Code not shown. It checks if any player has won the game
        // A player has won the game if it has 3 markers in a row,
        // either horizontal, vertical, or diagonal. The method
        // pops up a dialog box if a player wins, e.g., “X wins!”
    }

    public void actionPerformed(ActionEvent e) {
    }
}
```

`/* End of ClickHandler class */`
1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.