**Question 1.** (3 points) The terms *coupling* and *cohesion* are used to talk about the design quality of a collection of classes.

(a) To improve the design of a collection of classes, should we increase or decrease these qualities? Circle the correct answer for each quality:

Coupling: increase (decrease)

Cohesion: (increase) decrease

(b) A good rule of thumb in design is that each thing (class, object, method, etc.) should do one thing well, instead of several unrelated things. Which of these expresses that idea best? (circle)

coupling (cohesion)

**Question 2.** (2 points) JavaDoc comments contain funny strings like `@param` and `@return`, as in the following:

```
/**
 * Return the next larger integer
 * @param n the original integer
 * @return the value n+1
 */
public int increment(int n) {
   return n+1;
}
```

What is the purpose of these funny strings? What would change if we removed the "`@`" symbols and wrote `param` and `return` instead of `@param` and `@return`?

**The @ tags control the formatting of the resulting html JavaDoc pages. Without the @ symbols, the generated documentation wouldn't have special formatting for the parameters and return values. Instead, all of the text (including the words param and return) would be run together in the description of the method.**

**Question 3.** (10 points) In the following table, there are separate columns for interfaces, abstract classes, and ordinary (concrete) classes, and there are separate rows describing various properties that some or all of these items can have. Mark an "X" in each box where the corresponding interface, abstract class, or ordinary class can have the property described in that row.

| Interfaces can… | Abstract Classes can… | Regular (not abstract) Classes can… | Mark **all** column-row intersections for which the complete statement is **true**. |
|---|---|---|---|
| X | X | | contain method specifications without implementations. |
| | X | X | contain method implementations. |
| | X | X | contain constructors. |
| | X | X | contain instance variables. |
| X | X | X | contain `static final` variables. |
| | X | X | implement an interface. |
| | X | X | inherit from an abstract class. |
| | X | X | inherit from a regular class. |
| X | X | X | be used as the declared (static) type of a variable. |
| | | X | have instances created using `new` |

**Question 4.** (14 points) Consider the following class definitions. You can remove this page from the exam to look at while answering the questions on the next page.

```
interface Bespin {
  void gas();
}

public abstract class CloudCity {
  public abstract void noise();
  public void hail() {
    System.out.println("hailing " + me());
  }
  public String me() { return "cloud city"; }
}

public class Lando extends CloudCity {
  public void noise()       { System.out.println("holler"); }
  public void credit()      { System.out.println("one credit"); }
  public void credit(int k) { System.out.println( k + " credits"); }
}

public class CarbonFreeze extends CloudCity implements Bespin {
  public void gas()    { System.out.println("tibana"); }
  public void noise()  { System.out.println("whoosh"); }
}

public class Lobot extends Lando {
  public String me()  { return "lobot"; }
  public void noise() {
    super.noise();
    System.out.println("(whisper)");
  }
  public void credit() {
    credit(5);
    System.out.println("another credit");
  }
}
```

**Question 4.** (cont.) For each of the following, indicate what output is produced when the code is executed. If some sort of error occurs instead, give a brief explanation of the problem.

(a)
```
Lando b = new Lando();
b.noise();
```
**holler**

(b)
```
Lando b = new Lando();
b.hail();
```
**hailing cloud city**

(c)
```
CloudCity c = new CloudCity();
c.noise();
```
**error – cannot instantiate abstract class**

(d)
```
Lando b = new Lobot();
b.credit();
```
**5 credits**
**another credit**

(e)
```
Lobot t = new Lobot();
CloudCity c = t;
c.hail();
```
**hailing lobot**

(f)
```
CloudCity c = new Lobot();
Lobot t = c;
t.hail();
```
**error – c does not have type Lobot**

(g)
```
Bespin b = new CarbonFreeze();
b.gas();
b.noise();
```
**tibana**
**error – no noise() in Bespin**

**Question 5.** (1 point) True or false: Suppose we have two packages `package1` and `package2`, both of which contain a class named `Mumble`. If we try to import both classes by writing

```
import package1.*;
import package2.*;
```

at the top of a Java source file that doesn't otherwise refer to `Mumble`, the code won't compile because both packages contain a class with this name.

true (false)

**Question 6.** (2 points) If we want to implement a class that listens for mouse clicks, we could do it either by implementing the `MouseListener` interface or by extending the `MouseAdapter` class. Give one (short) reason why one might chose to extend `MouseAdapter` instead of implementing `MouseListener`.

**If we extend `MouseAdapter`, we only have to override the methods we want to implement and don't have to implement the rest of the methods in `MouseListener`. Instead, we inherit the empty implementations in `MouseAdapter` for the methods we don't implement in the new class.**

**Question 7.** (5 points) In a Model-View-Controller architecture, each of those components has various responsibilities. For each of the following, identify which component has that responsibility by writing "M", "V", or "C" in the blank.

__**M**__ Notify all viewers when something has changed.

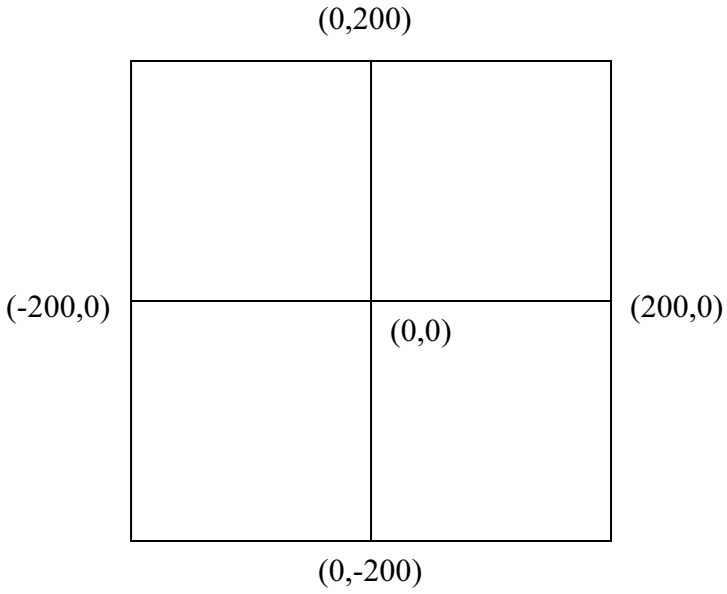
__**C**__ Handle input events from the user.

__**M**__ Maintain the state of all objects in the computation.

__**V**__ Display the current status of the system on the screen for the user.

__**M**__ There is only one instance of this component even though there may be multiple instances of the other two.

**Question 8.** (5 points) Suppose we are simulating a game that is played on a field that measures 400 feet by 400 feet. Horizontal (x) and vertical (y) coordinates are measured from the center (0,0) of the field, and each have a range from -200 to + 200. The location of a point on the field is given by its (x,y) coordinate pair.




We want to display this (x,y) field on a 300 by 300 pixel `JPanel` window. To do this we have to convert the (x,y) game coordinates to (h,v) horizontal and vertical pixel coordinates in the window.

(a) Write down one or more Java statements needed to convert a y coordinate on the 400 by 400 foot field to the corresponding vertical screen coordinate v in the 300 by 300 pixel graphics window. Your equation can either round or truncate the calculated value to the nearest pixel.

```
v = (int) ( (200.0 - y) * 300 / 400.0 )
```

**Equivalent formulas are also fine.**

(b) Assuming we are using the Model-View-Controller architecture for the program that simulates this game, where in the code should the coordinate conversion described in part (a) be placed? (circle)

model (view) controller