CSE 331 18wi Final Exam 3/12/18 Sample Solution

Question 1. (8 points, 2 each) Equality. Recall that there are several different notions of object equality that we've encountered this quarter. In particular, we have the following three:

- Reference equality: two objects are reference equivalent (==) if they are the same object.
- Behavioral equality: two objects are behaviorally equivalent if there is no sequence of operations (other than ==) that can distinguish them.
- Observational equality: two objects are observationally equivalent if there is no sequence of *observer* operations that can distinguish them (other than ==)

Now suppose we have a class `Thing` that implements *immutable* objects with appropriate `equals` and `hashCode` methods. Consider the following code fragment. We would like to know, for each of the assertions P, Q, R, and S, what we can conclude at that point in the program.

```
final Thing x = ...;
final Thing y = ...;
if (x.hashCode() == y.hashCode()) {
   { P }
} else {
   { Q }
}
if (x.equals(y)) {
   { R }
} else {
   { S }
}
```

For each of the assertions P, Q, R, and S, indicate, by writing in the blank, the roman numeral of the *strongest* assertion what we can conclude at that point in the program:

i. x is equal to y (by reference)
ii. x is equivalent to y (observationally or behaviorally or both)
iii. x is not equal to y (by reference)
iv. x is not equivalent to y (observationally or behaviorally or both)
v. true (the weakest possible assertion)

P ___v___ Q ___iv___ R ___ii___ S ___iv___

Note: For Q and S we awarded partial credit if the given answer was iii. It is true if two objects are not equivalent as reported by equals then they are not the same object (not ==), but that is a weaker assertion.

Question 2. (8 points, 4 each) Debugging. Use the terms defect, error, and failure to answer the following questions in 1 or 2 sentences.

(a) Define *debugging* in terms of these concepts.

Discover the defect that led to a failure.

(b) What is the purpose of adding an `assert` statement to a program?

Turn an error into a failure as quickly as possible.

Question 3. (8 points, 4 each) More debugging. Also answer in 1 or 2 sentences each.

(a) Suppose we have a program that exhibits a failure only when assertions are disabled and executes correctly if assertions are enabled. What is the most likely reason this could happen?

Most likely is that the expression in some `assert` statement has a side effect or performs a computation that is required for correct functioning of the program.

(b) A first step in debugging is to create a small test that demonstrates the failure. Why should we add that test case to the regression-test suite once we've fixed the bug? After all, the bug is fixed and all the tests pass now.

A defect is caused by some error or misunderstanding. Adding the test to the permanent suite guarantees that if the defect is re-introduced in the future for a similar or different reason it will be caught. It also increases the quality and comprehensiveness of the test suite.

The next several questions concern the following classes.

Please remove this page from the exam and use it to answer questions on the next few pages. Do not include this page in your submitted exam. It will not be graded.

Consider the following classes:

```
abstract class Bird {
  public abstract void speak();
  public void move() { System.out.println("flap flap!"); }
  public void move(int n) { move(); speak(); }
}

class Canary extends Bird {
  public void speak() { System.out.println("chirp!"); }
  public void move(int n) { speak(); speak(); }
}

class Duck extends Bird {
  public void speak() { System.out.println("quack!"); }
}

class RubberDuck extends Duck {
  public void speak() { System.out.println("squeak!"); }
  public void move() { speak(); swim(); }
  public void swim() { System.out.println("paddle!"); }
}
```

Please remove this page from the exam and use it to answer questions on the next few pages. Do not include this page in your submitted exam. It will not be graded.

Question 4. (12 points, 2 points each). Here are several groups of statements that might be found in a program that uses the classes on the previous page. For each group, if the statements compile and execute successfully without any errors, write the output that is produced. If there is an error, explain in a sentence what is wrong.

(a)
```
Bird b = new Bird();
b.move();
```

Compile error: cannot create instances of an abstract class.

(b)
```
Bird b = new Canary();
b.move(17);
```

```
chirp!
chirp!
```

(c)
```
Bird b = new Duck();
b.move(42);
```

```
flap flap!
quack!
```

(d)
```
Bird b = new RubberDuck();
b.move(3);
```

```
squeak!
paddle!
squeak!
```

(e)
```
Duck donald = new RubberDuck();
donald.swim();
```

Compile error: no `swim` method in class `Duck`

(f)
```
Duck donald = new RubberDuck();
donald.move();
```

```
squeak!
paddle!
```

Question 5. (8 points) More Birds. We'd now like to create a class to hold a collection of `Birds` (an aviary). Here are some basic parts of the class definition:

```
// a collection of Birds
public class Aviary {
  private List<Bird> birds;
  private static Random r = new Random();

  // construct an empty Aviary
  public Aviary() {
    birds = new ArrayList<Bird>();
  }

  // add one Bird to the Aviary
  public void add(Bird b) {
    birds.add(b);
  }
  // return a random Bird from the Aviary

  public Bird get() {
    return birds.get(r.nextInt(birds.size()));
  }
}
```

(a) (2 points) We would like to add a method to this class to be able to add any collection of `Birds` to an `Aviary`. Fill in the most general type possible for the `flock` parameter of the `addAll` method below so it will accept any Java Collection containing `Birds` or objects that are from any subclasses of `Birds`.

```
  // add a Collection of Birds to the Aviary

  public void addAll( Collection <? extends Bird> flock) {
    birds.addAll(flock);
  }
```

(continued next page)

Question 5. (cont.) For some applications we'd like to create a collection of `Birds` that can hold only `Ducks` or subtypes of `Ducks`. Suppose we create the following class:

```
class Pond extends Aviary {
  public void add(Duck d) {
    super.add(d);
  }
}
```

(b) (2 points) Does this new add method override or overload the add method inherited from `Aviary`, or does it cause some sort of error when we try to compile this new class?

Overload.

(`Pond` contains an `add(Duck)` method as well as the inherited `add(Bird)`.)

(c) (2 points) Is class `Pond` a Java subtype of `Aviary`? Explain your answer in 1 or 2 sentences.

Yes. `Pond` is declared to extend `Aviary`, and the resulting subclass compiles without errors.

(d) (2 points) Is class `Pond` a true subtype of `Aviary`? Explain your answer in 1 or 2 sentences.

Yes, but probably not for the reasons the author of `Pond` intended. An instance of `Pond` can be substituted for an instance of `Aviary` and any client code that expects an `Aviary` object will see exactly the same behavior when it is using a `Pond`.

Question 6. (10 points, 1 each) Generics and containers. Recall our `Bird` class hierarchy:

```
class Bird
class Canary extends Bird
class Duck extends Bird
class RubberDuck extends Duck
```

Now suppose we have the following variables:

```
Object o; Bird b; Canary c; Duck d; RubberDuck r;

List<? extends Bird> leb;
List<? extends Duck> led;
List<? super Duck> lsd;
```

For each of the following, circle OK if the statement has correct Java types and will compile without type-checking errors; circle ERROR if there is some sort of type error.

OK [✓ ERROR] `led.add(d);`

OK [✓ ERROR] `leb.add(o);`

[✓ OK] ERROR `lsd.add(r);`

OK [✓ ERROR] `lsd.add(o);`

[✓ OK] ERROR `leb.add(null);`

OK [✓ ERROR] `d = lsd.get(1);`

[✓ OK] ERROR `d = led.get(1);`

[✓ OK] ERROR `b = leb.get(1);`

[✓ OK] ERROR `o = led.get(1);`

OK [✓ ERROR] `b = lsd.get(1);`

Question 7. (8 points, 2 each) Comparing specifications. Recall that a prime number *n* is a natural number (integer) that is greater than 1 and that cannot be written as the product of two natural numbers that are both smaller than *n*. Consider the following specifications. All values (parameter and return values) are integers.

```
S1: @param x  input number
    @requires x > 0
    @return the largest prime number that is less than x
    @throws NoSuchPrimeNumberException if there is no prime number that is less
                than x

S2: @param x  input number
    @requires x > 2
    @return the largest prime number that is less than x

S3: @param x  input number
    @requires x > 0
    @return the largest prime number that is less than or equal to x
    @throws NoSuchPrimeNumberException if there is no prime number that is less
                than or equal to x

S4: @param x  input number
    @return the largest prime number that is less than x
    @throws NoSuchPrimeNumberException if there is no prime number that is less
                than x
```

In the answers below, you do not need to include each specification in the list of ones that are stronger or equivalent to itself. Just list the other specifications that are stronger or equivalent (if any). If there are no other specifications in an answer, write "none".

(a) List all of the specification that are stronger than or equivalent to S1. ___**S4**________

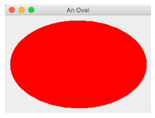
(b) List all of the specification that are stronger than or equivalent to S2. ___**S1, S4**_____

(c) List all of the specification that are stronger than or equivalent to S3. ___**none**______

(d) List all of the specification that are stronger than or equivalent to S4. ___**none**______

Note: No partial credit was awarded if a wrong answer was included in addition to the ones given above. We did allow 1 point partial credit on (b) if one of the two correct answers was omitted.

Question 8. (10 points) Java graphics. One of the new interns has been trying to learn Swing so they can work on an old application program that uses it. The intern has come up with this program, which is supposed to draw a red oval that almost fills a window. Unfortunately, when the program is run nothing appears to happen. We've rechecked the method calls that compute the size of the oval and we're sure that those are right – the size is computed correctly, and the width and height parameters are in the right order. Something else is causing the problem.



Your job is to modify, delete, or add the necessary code to fix the program so the output will look like the drawing to the left (i.e., the expected picture). There might be multiple bugs.

```
import java.awt.*;
import javax.swing.*;
public class DrawOval {
  public static void main(String[] args) {
    JFrame frame = new JFrame("An Oval");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new Oval();
    panel.setPreferredSize(new Dimension(300,200));
//  ~~panel.paintComponent(new Graphics2D());~~
    frame.add(panel, BorderLayout.CENTER);
    frame.pack();
    frame.setVisible(true);
  }
}
public class Oval extends JPanel {
  @Override
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    int height = getHeight();
    int width  = getWidth();
    // draw a red oval
    g2.setColor(Color.red);
    g2.fillOval(10,10,width-20,height-20);
  }
}
```

Corrections are shown in the code above, and all are in method `main`. The program should not create a new `Graphics2D` object and should not call `paintComponent`. It does need to add the `Oval` to the frame, pack the frame, and make it visible. `frame.add(panel)` would also work without the `CENTER` parameter, and the code would work if the panel is added elsewhere in the frame.

We did not penalize additional changes to the code if the resulting program still produced the correct results.

Question 9. (10 points, 2 each) Design patterns. Here is an alphabetical list some design patterns we discussed this quarter. Note that some of these may be more specific instances of other patterns.

Adapter, Builder, Composite, Decorator, Dependency Injection, Factory, Iterator, Intern, Interpreter, Model-View-Controller (MVC), Observer, Procedural, Prototype, Proxy, Singleton, Visitor

For each statement below, list *all* of the design patterns from the list above that meet the description. Each answer might include one or more design patterns, and for full credit you must list all of them. There is at least one design pattern that fits each of these descriptions.

(a) The pattern involves a class where most of the functionality is provided by one other class.

Adapter, Decorator, Proxy

(b) The pattern is a way to implement the Procedural pattern without using `instanceof` tests.

Visitor

(c) The pattern should only be used only for immutable classes.

Intern

(d) The pattern is fundamental to the way that Java's Graphical User Interface (GUI) libraries are organized.

Observer

(e) It is necessary to make constructors private if we want to *require* that clients use this pattern.

Builder, Factory, Prototype, Singleton, Intern

Note: For part (e), we did award partial credit if one or two of the listed patterns were missing and no incorrect ones were supplied, but if more than a couple of the correct ones were missing then that was not sufficient for partial credit.

Question 10. (5 points) Build tools. All modern development environments (IDEs) like Eclipse, Intellij, Visual Studio, and so forth have built-in tools to automatically build and rebuild executable programs as code is added or changed in a project. Yet almost all projects use an external build tool like ant or make to do the actual building, bypassing the provided tools in the IDE. Why? Why not just simplify things by using the IDE build tools? (Be brief – a couple of sentences ought to be enough.)

It is important that all members of a project team use the same build strategy and tools so the build is consistent and reproducible for all team members, regardless of their preference in development environments or systems.

Question 11. (6 points) Version control. A major reason for using version control systems like git is to allow many programmers to collaborate on a project. But even if you are working on a project by yourself, using a version control system can be useful. Describe two distinct benefits of using a version control system for a project even if you are working by yourself. One or two sentences each should be enough.

Here are several:

(i) Provide backup storage for files.

(ii) Provide a history log to see when changes were made and, assuming useful log messages, why.

(iii) Provide the ability to retrieve past versions of files.

(iv) Make it easy to keep files synchronized and up-to-date when working on a project in multiple locations. (This is very similar to (i), but we did allow credit for it as a different reason.)

$$\begin{tikzcd}
 & & & A \arrow[dll] \arrow[d] \arrow[drr] & & \\
 & B \arrow[dr] & & C \arrow[dl] \arrow[dr] & & D \arrow[dd] \\
 & & E & & F \arrow[dr] & \\
 & & & & & G
\end{tikzcd}$$

Question 12. (5 points, 1 point each) System integration. When building a large system, there are two common strategies for the order in which to implement and test the different parts of the system: top-down and bottom-up. These two strategies have different characteristics and strengths.

For each of the following, circle "top-down" or "bottom-up" if that strategy is the best match to the description. If both strategies are a good match or are effective at solving the problem, circle "both". If neither strategy matches the description or neither is particularly effective at solving the problem, circle "neither".

(a) Best at catching major design or usability errors early

[✓ top-down] bottom-up both neither

(b) When a new module is added, the number of modules it interacts with and potential number of places to look for an error is larger

top-down [✓ bottom-up] both neither

(c) Requires building stubs, sometimes also known as "mock objects"

[✓ top-down] bottom-up both neither

(d) Best at showing visible or tangible progress to clients and other team members

[✓ top-down] bottom-up both neither

(e) Best at uncovering infrastructure efficiency issues early

top-down [✓ bottom-up] both neither

Question 13. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

Draw a picture of something that you plan to do during your spring break!

(Ideas for better pictures to include in the permanent copy of the sample solution would be appreciated.)

Congratulations from the CSE 331 staff!
Have a great spring break!!