15-214: Principles of Software Construction

Spring 2016 Sample Midterm Exam 1

Name: _____

Andrew ID: _____

Recitation section or TA name: _____

Instructions:

- Make sure that your exam has 10 pages (not including this cover sheet) and is not missing any sheets. Then write your full name, **Andrew ID**, and recitation section on this page (and all the others if you want to be safe).
- Write your answers in the space provided below the problem. Clearly indicate your answers.
- The exam has 6 questions with a maximum score of 107 points. The point value of each problem is indicated. Ideally you should take approximately one minute per point; pace yourself accordingly.
- This exam is CLOSED BOOK and CLOSED NOTES. You may not use a calculator, laptop or any other electronic or wireless device.
- If our questions are unclear, please make and state your assumptions.
- Good luck!

Question	Points	Score
Agree or disagree	24	
An employee payment system	22	
Java equality	11	
Pattern problems	8	
Roshambo	20	
A party invitation system	22	
Total:	107	

Question 1: Agree or disagree (24 points)

For each of the following statements, do you agree or disagree with the statement? Justify your answer with a short, 1-3 sentence explanation, using precise, technical terminology when possible. For some statements there is not necessarily a single correct answer; you earn credit by convincingly justifying your answer. Answers without justifications will not earn credit.

(a) (4 points) Using `instanceof` in Java indicates poor object-oriented design.

(b) (4 points) Delegation is better than inheritance for adding functionality to a Java class.

(c) (4 points) A unit test should only test the behavior of an individual method, not the interaction of multiple methods.

(d) (4 points) Formal specifications are superior to textual specifications.

(e) (4 points) Unit testing makes formal verification unnecessary.

(f) (4 points) You should make objects immutable if posssible.

Question 2: An employee payment system (22 points)

The `HourlyEmployee` class below tracks the wage and hours for an employee for one week of work; at the end of the week the system can calculate how much money the employee is owed for her work. A subclass, `TippedEmployee`, calculates the payment differently for employees who get tips at their job, such as servers at a restaurant. Tipped employees can be paid a lower hourly wage (below the normal minimum wage) if their wages plus tips exceed the normal minimum wage. Here is the code and specification:

```
class HourlyEmployee {
    //@invariant wage >= Wage.getMinimumWage()  // currently 700
    //@invariant hours >= 0
    final int wage; // cents per hour
    int hours = 0;

    //@requires w >= Wage.getMinimumWage()  // currently 700
    HourlyEmployee(int w) {
      wage = w;
    }

    //@requires h >= 0
    //@ensures  hours == \old(hours) + h
    void addHours(int h) { hours = hours + h; }

    //@ensures \result == hours * wage
    int computePaymentOwed() { return hours * wage; }
}

class TippedEmployee extends HourlyEmployee {
    //@invariant wage >= Wage.getTippedMinimumWage() // currently 300
    //@invariant hours >= 0
    //@invariant tips >= 0
    int tips = 0;

    //@requires w >= Wage.getTippedMinimumWage() // currently 300
    TippedEmployee(int w) {
      super(w);
    }

    //@requires t >= 0
    //@ensures  tips == \old(tips) + t
    void addTip(int t) { tips = tips + t; }

    //@ensures \result == max(hours * wage, hours * Wage.getMinimumWage() - tips)
    int computePaymentOwed() {
      return Math.max(hours * wage, hours * Wage.getMinimumWage() - tips);
    }
}
```

_____ / 0

(a) (8 points) Is `TippedEmployee` a behavioral subtype of `HourlyEmployee`? If not, which `TippedEmployee` specification line(s) violate the the behavioral subtyping rules and which rule(s) are they violating? Circle the specification line(s) above, and describe here which rule(s) they are violating.

(b) (8 points) Provide a good set of test cases for the `TippedEmployee.computePaymentOwed()` method. For the purpose of this question just test the behavior of `computePaymentOwed()` on valid data; i.e., describe hours, wages, and tips that satisfy the invariants of the class, and write the expected result of the `computePaymentOwed()` method for that data. Be sure that your set of test cases demonstrates an understanding of testing best practices; we believe a good test suite can be written with four or five well-considered test cases.

`hours`	`wage`	`tips`	Expected `computePaymentOwed()` result

(c) (6 points) Rewrite the `HourlyEmployee` specifications to ensure that no employee works more than 40 hours per week, describing here your changes to the invariants, preconditions, and/or postconditions. You might have to change more than one part of the specification. Ignore the `TippedEmployee` class for the purpose of this question.

Question 3: Java equality (11 points)

Consider the following code that uses a `Child` class, storing a child's name and age:

```
Child p = new Child("Peter", 12);
Child q = p;
Child r = new Child("Rebecca", 14);
Child s = new Child("Peter", 12);
```

Intuitively, two `Child`s are equal if and only if they share the same name and age.

(a) (3 points) If the `Child` class is implemented correctly to satisfy the above intuition, does each of the following expressions evaluate to `true` or `false`?

i. `p == q` _____

ii. `p == r` _____

iii. `p == s` _____

iv. `p.equals(q)` _____

v. `p.equals(r)` _____

vi. `p.equals(s)` _____

(b) (8 points) Complete the `Child` class, including all code needed for equality-checking to work as described above and also satisfy the `Object` method contracts.

```
public class Child {
    private final String name;
    private final int age;
    public Child(String name, int age) {
        if (name == null) throw new NullPointerException();
        this.name = name;
        this.age = age;
    }
```

Question 4: Pattern problems (8 points)

An enthusiastic youngster in your company has grand plans to use design patterns within the company. The first part of the plan is to make a list of all the design patterns. The second part of the plan is to develop a library that implements each pattern in the list, so that the pattern code can be reused.

The enthusiastic youngster has two significant misconceptions about patterns. What are they and why are they misconceptions?

- Misconception #1:
- Why a misconception:
- Misconception #2:
- Why a misconception:

Question 5: Roshambo (20 points)

Roshambo, or Rock-paper-scissors, is a two player game in which each player simultaneously forms one of three shapes—rock, paper, or scissors—with an outstretched hand. The game has only three outcomes other than a tie: rock beats scissors, scissors beats paper, and paper beats rock.

(a) Write a Java enum to represent a move (rock, paper, or scissors) in Roshambo. The enum should contain an instance method to compare this move to another move, to determine which of the two moves wins. This instance method should be similar to the `Comparable.compareTo()` method that takes another move as an argument, returning a negative, zero, or positive value to indicate that this move is beaten, ties, or beats the other move.

(b) Java enums already contain a final `compareTo` method, but pretend it is not final for the purposes of this question. One solution to part (a) might be to override `compareTo(...)` to return the requisite value. In 2-3 sentences explain why this would be a poor design choice.

(c) Write a `main` method in your `Roshambo` class to actually play Roshambo. The sole command line argument is the number of games to play. Play the required number of games. In each game, repeatedly generate a random play for player A and another for player B and print them out until a winner has been determined. The output should look like this:

```
java Roshambo 3
A plays SCISSORS, B plays ROCK
B wins.

A plays ROCK, B plays ROCK
A plays PAPER, B plays PAPER
A plays SCISSORS, B plays PAPER
A wins.

A plays PAPER, B plays PAPER
A plays ROCK, B plays ROCK
A plays PAPER, B plays ROCK
A wins.
```

When generating random Roshambo turns, you may find it useful to use this function, which returns a random element from an array:

```
private static Random rnd = new Random();

static <E> E randomElement(E[] a) {
    return a[rnd.nextInt(a.length)];
}
```

_____ / 0

Question 6: A party invitation system (22 points)

In this problem you will design the core of an online party invitation system. There are one-time parties, repeated weekly parties, and repeated annual parties; other types of parties might be added in the future. Each party has a host, a guest list, a name, a description, a location, and a date and time.

Hosts must have an account for the party invitation system. Guests may be invited by hosts and thus might not have an account; they can sign up for an account (by selecting a username and password) in the future. A host can use the system to notify all party guests of a message, and a guest can use the system to notify the host of her intention to attend (or not attend) the party. The system initially supports SMS and email notifications (which require a phone number and email address, respectively), and more notification mechanisms might be added in the future. Hosts and guests with accounts can set their preferred notification mechanism.

(a) (10 points) Based on the description above, create a domain model for the party invitation system, using a UML class diagram and standard UML notation where possible; you may explain any non-standard notation you use. Please try to draw neatly.

(b) (8 points) Consider the following scenario:

> Amelia registers for an account on the party system, then uses the system to create a party with herself as the host. She then adds her friends to the guest list (one by one), including each guest's email address or phone number for later notifications.

Create a system sequence diagram for the above scenario, including any interactions between the overall system and its external users. Use a UML sequence diagram, using standard UML notation where possible; you may explain any non-standard notation you use.

(c) (4 points) Write a reasonable system-level behavioral contract for the interaction that allows a user to create a party with himself/herself as the host. Be sure that your answer demonstrates an understanding of the parts and purpose of system-level behavioral contracts.