Exam 2
CSCI 2600 Principles of Software
November 3, 2015

- DO NOT OPEN THIS EXAM UNTIL TOLD TO DO SO!
- READ THROUGH THE ENTIRE EXAM BEFORE STARTING TO WORK.
- YOU ARE ALLOWED ONLY 2 “CHEAT” PAGES. NO OTHER MATERIAL IS ALLOWED.

This exam is worth 150 points.

Make sure you have 12 pages counting this one. There are 3 parts, each including multiple questions for a total of 19 questions. If you need more room for an answer than is provided, please use the back of the page and indicate that you have done so. If you re-do a question, please make clear what is your final answer.

Be clear and brief in your explanations—rambling and lengthy answers will be penalized. All questions have short answers.

The following is for the use of graders

1. ________/30
2. ________/80
3. ________/40

TOTAL: ________/150
Part I. Exceptions, Proving Rep Invariants

Question 1. (6 pts, 2 pts each) TRUE/FALSE
   
   a) (TRUE/FALSE) An exception always indicates an unrecoverable failure in the program.
   
   b) (TRUE/FALSE) IllegalArgumentException is a checked exception.
   
   c) (TRUE/FALSE) Sometimes, it is desirable to catch one exception and throw a different exception.

Question 2. (10 pts, 2pts each) In these situations, which is better, a checked or unchecked exception?

   a) Map.getExisting when the key is not in the map ______CHECKED______

   b) Rational.divide when the argument is zero ____CHECKED________

   c) Object.clone when the system is out of memory ___UNCHECKED________

   d) File.load when the file does not exist ____CHECKED________

   e) File.load when there is a disk failure ___UNCHECKED________

Question 3. (9 pts) The following class represents an interval of time between two Dates.

```java
public class Interval {
    private Date start;
    private Date stop;
    private long duration;
    // Rep invariant: duration = stop.getTime() - start.getTime()

    public Interval(Date start, Date stop) {
        this.start = start;
        this.stop = stop;
        duration = stop.getTime() - start.getTime();
    }

    public Date getStart() { return start; }

    public Date getStop() { return stop; }

    public long getDuration() { return duration; }
}
```

Willy Wazoo argues that the rep invariant of Interval always holds, because duration is initialized once in the constructor, to the correct value. Give all reasons why Willy is wrong.
Question 4. (5 pts) Here are 3 specifications for function `public double sqrt(double x)`.

Spec A: `@requires x ≥ 0
   @returns y such that |y*y - x| < 0.0001`

Spec B: `@returns y such that |y*y - x| < 0.0001 if x ≥ 0, and 0.0 if x < 0`

Spec C: `@returns y such that |y*y - x| < 0.0001 if x ≥ 0
   @throws IllegalArgumentException if x < 0`

Order the specifications from best choice to worst choice. Explain your answer.

Answer: _, _, _
Name: __________________________

Part II. True Subtyping, Equality, Java Subtyping, Overloading and Java Generics

Question 5. (10 pts, 2 pts each) TRUE/FALSE

a) (TRUE/FALSE) A true subtype is always a Java subtype.

b) (TRUE/FALSE) In Java, an overriding method can declare a new exception, as long as the new exception is a subtype of one declared in the overridden method.

c) (TRUE/FALSE) The consistency property of hashCode states that for every non-null x and y, x.equals(y) implies x.hashCode() == y.hashCode().

d) (TRUE/FALSE) In Java, which method family is called, is determined at runtime.

e) (TRUE/FALSE) An important similarity between interface and abstract class is that neither can be instantiated.

Question 6. (18 pts) Consider the code.

class X {
    void m(X a) { System.out.println("XX"); }
    void m(Y a) { System.out.println("XY"); }
    void m(Z a) { System.out.println("XZ"); }
}
class Y extends X {
    void m(X a) { System.out.println("YX"); }
    void m(Y a) { System.out.println("YY"); }
    void m(Z a) { System.out.println("YZ"); }
}
class Z extends Y {
    void m(X a) { System.out.println("ZX"); }
    void m(Y a) { System.out.println("ZY"); }
    void m(Z a) { System.out.println("ZZ"); }
}

X x1 = new X();
X x2 = new Y();
X x3 = new Z();
Y y1 = new Y();
Y y2 = new Z();
Z z1 = new Z();

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>y1</th>
<th>y2</th>
<th>z1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
<td>Xy</td>
<td>Xy</td>
<td>Xz</td>
</tr>
<tr>
<td>x2</td>
<td>Yx</td>
<td>Yx</td>
<td>Yx</td>
<td>Yy</td>
<td>Yy</td>
<td>Yz</td>
</tr>
<tr>
<td>x3</td>
<td>Zx</td>
<td>Zx</td>
<td>Zx</td>
<td>Zy</td>
<td>Zy</td>
<td>ZZ</td>
</tr>
<tr>
<td>y1</td>
<td>Yx</td>
<td>Yx</td>
<td>Yx</td>
<td>Yy</td>
<td>Yy</td>
<td>Yz</td>
</tr>
<tr>
<td>y2</td>
<td>Zx</td>
<td>Zx</td>
<td>Zx</td>
<td>Zy</td>
<td>Zy</td>
<td>ZZ</td>
</tr>
<tr>
<td>z1</td>
<td>Zx</td>
<td>Zx</td>
<td>Zx</td>
<td>Zy</td>
<td>Zy</td>
<td>ZZ</td>
</tr>
</tbody>
</table>

Fill in each box in the above table with the output of the corresponding invocation. For example, fill in the cell for row headed by y1 and column headed by x2 with the output of y1.m(x2).
Questions 7-9 below are based on the following code for a 2D point with integer coordinates. Note: be careful, the question has changed from the practice test.

```java
public class Point {
    private int x;
    private int y;
    public boolean equals(Object o) {
        if (o instanceof Point) {
            Point p = (Point) o;
            return p.x == x && p.y == y;
        }
        else
            return false;
    }
}

Question 7. (6 pts) Let’s implement a 3D point:

```java
public class ThreeDPoint extends Point {
    private int z;
    public boolean equals(Object o) {
        if (o instanceof ThreeDPoint) {
            ThreeDPoint p = (ThreeDPoint) o;
            return super.equals(o) && p.z == z;
        }
        else
            return false;
    }
}
```

Indicate which of the following properties holds for the `equals` method.

(a) Is `equals` reflexive? If not, give a counterexample below.

(b) Is `equals` symmetric? If not, give a counterexample.

(c) Is `equals` transitive? Again if not, give a counterexample.
Question 8. (6 pts) Another implementation of ThreeDPoint.equals:

```java
public boolean equals(Object o) {
    if (o instanceof Point) {
        return o.equals(this);
    }
    else if (o instanceof ThreeDPoint) {
        ThreeDPoint p = (ThreeDPoint) o;
        return super.equals(o) && p.z == z;
    }
    else
        return false;
}
```

Again, indicate which of the following properties holds for the equals method.

(a) Is equals reflexive? If not, give a counterexample below.

(b) Is equals symmetric? If not, give a counterexample.

(c) Is equals transitive? Again if not, give a counterexample.

Question 9. (6 pts) Yet another implementation of ThreeDPoint.equals:

```java
public boolean equals(Object o) {
    if (o instanceof ThreeDPoint) {
        ThreeDPoint p = (ThreeDPoint) o;
        return super.equals(o) && p.z == z;
    }
    else if (o instanceof Point) {
        return super.equals(o);
    }
    else
        return false;
}
```

(a) Is equals reflexive? If not, give a counterexample below.

(b) Is equals symmetric? If not, give a counterexample.

(c) Is equals transitive? Again if not, give a counterexample.
Questions 10-12 below use this code.

// Digit represents a single digit, from 0 to 9.
// @specfield value: The value of the digit (0 through 9 inclusive).
public class Digit {

    private int v;

    private static Digit[] instances = new Digit[10];

    // Constructs a Digit representing the given character,
    // such that digit.value = i.
    // @param i the value of the returned digit
    // @requires 0 <= i <= 9
    public Digit(int i) {
        if (i < 0 || i > 9)
            throw new IllegalArgumentException();
        this.v = i;
    }

    // Returns a Digit representing the given digit.
    // @requires 0 <= i <= 9
    // @param i the value of the returned digit
    // @returns a digit such that digit.value = i
    public static Digit factory(int i) {
        if (instances[i] == null) {
            instances[i] = new Digit(i);
        }
        return instances[i];
    }

    public int getValue() {
        return this.v;
    }

    // Counts the number of unique digits in the given number.
    // For example, the number 2012 has three unique digits: 0, 1, and 2.
    // @requires number is not null, and contains no null elements
    // @param number a number, represented as a list of Digits
    // @returns the number of unique digits in the given number;
    // the result is >= 0 and <= 10.
    public static int numDigits(List<Digit> number) {
        Set<Digit> s = new HashSet<Digit>;
        for (Digit d : number) {
            s.add(d);
        }
        return s.size();
    }
}
Question 10. (6 pts) A client calls `numDigits`. The client is surprised when `numDigits` returns 11. Explain how this failure is possible.

Question 11. (2pts) Write the smallest JUnit test that you can that exposes this problem in the implementation.

Question 12. (6pts) Give two distinct ways the `Digit` class can be modified to prevent this failure, without modifying the specification or implementation of the `numDigits` method. Be specific. You can use a small amount of code if you want, but you can get full credit without doing so.

a)

b)
Question 13 (5 pts) Consider the specifications of write-only NumberSet and IntegerSet. Integer is a Java and true subtype of Number.

// A mutable set of Numbers
class NumberSet {

    // @effects makes a new empty NumberSet
    public NumberSet();

    // @returns n ∈ this
    public boolean contains(Number n);

    // @modifies this
    // @effects thispost = thispre ∪ { n }
    public void add(Number n);
}

// A mutable set of Integers
class IntegerSet {

    // @effects makes a new empty IntegerSet
    public IntegerSet();

    // @requires n is an Integer
    // @returns n ∈ this
    public boolean contains(Number n);

    // @requires n is an Integer
    // @modifies this
    // @effects thispost = thispre ∪ { n }
    public void add(Number n);
}

Circle the correct statement. Explain your answer.

a) NumberSet is a true subtype of IntegerSet
b) IntegerSet is a true subtype of NumberSet
c) Neither is a true subtype of the other
Question 14. (15pts, 1.5pts each) Suppose we have the following classes:

class Animal { ... }
class Cat extends Animal { ... }
class Rat extends Animal { ... }

Suppose we have a program that contains the following objects and list:

Object o;
Animal a;
Cat c;
Rat r;
List<? extends Animal> lea;

For each of the following, circle Error if there is a type error (at compile time) or circle OK if the statement passes the type checker.

OK Error lea.add(a);
OK Error lea.add(c);
OK Error c = lea.get(0);
OK Error a = lea.get(0);
OK Error o = lea.get(0);

Now, let us declare another list

List<? super Animal> lsa;

As before, circle Error if there is a type error. Circle OK if the statement is correct.

OK Error lsa.add(a);
OK Error lsa.add(c);
OK Error c = lsa.get(0);
OK Error a = lsa.get(0);
OK Error o = lsa.get(0);
Part III. Testing

Question 15. (10 pts, 2pts each) TRUE/FALSE

a) (TRUE/FALSE) It is always possible to cover all def-use pairs in a function.

b) (TRUE/FALSE) All-uses coverage implies statement coverage.

c) (TRUE/FALSE) A test written against a stronger spec will work with a weaker spec.

d) (TRUE/FALSE) Specification tests and black-box tests are different names for the same concept.

e) (TRUE/FALSE) A test suite that detects every bug in an implementation, has 100% statement coverage.

Consider the specification for `binarySearch`. Questions 16-19 below concern `binarySearch`.

```java
/**
 * @requires: a is non-null, non-empty, and sorted in increasing order
 * @requires: val is an element in a
 * @returns: the index of val in a
 */
public static int binarySearch(int[] a, int val)
```

Question 16. (10 pts) Write 3 black-box JUnit tests making use of black-box heuristics, and give a brief description of what you are testing.

Here is an example test (you cannot reuse this one):

```java
@Test
public void testFoundKeyNearMiddle() {
    int[] a = {-3, -2, -1, 3, 7};
    assertEquals(2, SortedSearch.binarySearch(a,-1));
}
```

Description: This tests the class of output when the value is in the middle of the array.
Now, let's look at the implementation of `binarySearch`

```java
int binarySearch(int[] a, int val) {
    int min = 0;
    int max = a.length - 1
    while (min < max) {
        int mid = (min + max) / 2;
        if (val == a[mid]) {
            min = mid;
            max = mid;
        }
        else if (val > a[mid]) {
            min = mid + 1;
        }
        else { // val < a[mid]
            max = mid - 1;
        }
    }
    return max;
}
```

17. (10 pts) Draw the control-flow graph (CFG) for the `binarySearch` routine.

18. (5 pts) What is the % branch coverage that the 4 test cases from Question 16 achieve? What is the % statement coverage?

19. (5 pts) What is the smallest array that can achieve 100% branch coverage?