NAME_____

Final Exam
CSCI 2600 Principles of Software
December 12, 2014

- **DO NOT OPEN THIS EXAM UNTIL TOLD TO DO SO!**
- **READ THROUGH THE ENTIRE EXAM BEFORE STARTING TO WORK.**
- **YOU ARE ALLOWED ONLY 4 "CHEAT" PAGES. NO OTHER MATERIAL IS ALLOWED.**

This exam is worth 260 points.

Make sure you have 20 pages counting this one. There are 6 parts, each including multiple 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. _____/35
2. _____/40
3. _____/45
4. _____/30
5. _____/45
6. _____/65

TOTAL:_____/260

Name: _____

Part I. Reasoning About Code

1. (10pts, 2pts each) True/False questions.

a) (TRUE/FALSE) The loop invariant must hold before and after every statement in the loop.

b) (TRUE/FALSE) If **P** implies **Q** and **{ P } code { R }** is true, then **{ Q } code { R }** is true.

c) (TRUE/FALSE) if **P** implies **wp(code,Q)** then **{ P } code { Q }** is true.

d) (TRUE/FALSE) There may exist two logically distinct loop invariants LI_1 and LI_2 where each enables proving partial correctness. (Logically distinct means that LI_1 and LI_2 are not just different ways for writing the same logical formula.)

e) (TRUE/FALSE) If the loop terminates then the decrementing function is 0.

2. (10pts) Compute the weakest precondition with backwards reasoning. Show the intermediate conditions. Show the final solution on the number line.

Let **`x`** be a **`double`**.

```
_____________________________________
if (x < 5) {
   ____________________________________
   x = x*x;
   ____________________________________
}
else {
   ____________________________________
   x = x+5;
   ____________________________________
}
{ x ≥ 16 ∧ x ≤ 25 }
```

3. (15pts) This question asks you to prove that the following implementation is correct.

```
// Precondition: n >= 0
   i = 0;
   r = 1;
   while (i != n) {
      i = i+1;
      r = r*i;
   }
// Postcondition: r = n!
```

a) (7pts) Prove that the loop terminates. You don't need to be overly formal. However, you do need to find a suitable function D, show that it decrements, and then show that D=0 implies `¬ (i!=n)`.

b) (8pts) Prove that the loop computes the correct answer if it terminates. You have to find a suitable loop invariant and argue partial correctness.

Part II. Specifications, Subtyping and the LSP

1. (10pts, 2pts each) True/false questions.

a) (TRUE/FALSE) When a client calls a method without meeting the preconditions, the method must throw an exception.

b) (TRUE/FALSE) If S1 is a stronger specification than S2, then any implementation that satisfies S1 must also satisfy S2.

c) (TRUE/FALSE) A stronger specification is harder to implement than a weaker one.

d) (TRUE/FALSE) A weaker specification is easier to use than a stronger one.

e) (TRUE/FALSE) Adding more preconditions to a method sometimes strengthens and sometimes weakens the specification.

Name: _____

2. (10pts) Fill in the missing part of the specification. If a clause is empty, explicitly state "none". (Don't worry about what looks like a mismatch in the use of **`int`** and **`Integer`**. As we saw in class, this is type correct because Java autoboxes **`int`** into **`Integer`**.)

```
// A set of integers.
class IntSet {

   private List<Integer> set = new LinkedList<Integer>();

   // Inserts the argument in the IntSet.
   public void add(int x) { … }

   /**
    * returns true if and only if the given x is in this IntSet.
    *
    *@requires: _____________________________________________

    *@modifies: _____________________________________________

    *@effects: _____________________________________________

    *@returns: _____________________________________________

    */
    public boolean contains(int x) {
        int index = set.indexOf(x);
        if (index != -1) {
           set.remove(index);
           set.add(0,x);
        }
        return index != -1;
    }
}
```

3. (12pts, 2pts each) Consider the four specifications for **`double ln(double x)`** which returns the natural logarithm of **`x`**.

A
```
requires: x > 0
returns: y such that |e^y - x| <= 0.01
```

B
```
requires: x > 0
returns: y such that |e^y - x| <= 0.001
```

C
```
returns: y such that |e^y - x| <= 0.001
throws: IllegalArgumentException if x <= 0
```

D
```
returns: y such that |e^y - x| <= 0.0001 if x > 0
         and Double.NEGATIVE_INFINITY if x <= 0
```

Name: _____

For each pair below circle the **stronger** specification, or neither if the two are incomparable.

a) A B neither

b) A C neither

c) A D neither

d) B C neither

e) B D neither

f) C D neither

4. (8pts) Willy Wazoo wants to design a new Java-like programming language that disallows overloading and changes overriding so that an overriding method can have different parameters from the method it overrides. Willy type checks overriding methods using the following rule:

method **`Y' B.m(X' x)`** overrides method **`Y A.m(X x)`** correctly if **`X'`** is a *subclass* of **`X`** and **`Y'`** is a *subclass* of **`Y`**.

Willy's language supports subtype polymorphism just like Java: if **`B`** is a subclass of **`A`**, then **`B`** can be used where an **`A`** is expected and the overriding **`B.m`** must work where **`A.m`** is expected.

Explain why Willy's rule is wrong and correct his rule.

Name: _____

Part III. ADTs, Abstraction Functions and Rep Invariants

1. **(15pts, 3pts each)** Short answer.

a) Give three distinct reasons why you may **not** want to check the rep invariant at the entry and/or exit of a given method. Consider only instance methods that are not constructors.

b) Shall we worry about representation exposure in class **`Movie`**? Explain.

```
class Movie {
    // The rep
    private String title;
    …
    public String getTitle() { return title; }
}
```

c) How about class **`Duration`**? Shall we worry about representation exposure?

```
class Duration {
    // The rep
    private Date start;
    private Date end;
    …
    public Date getStart() { return start; }
    public Date getEnd() { return end; }
}
```

d) The abstraction function maps _____ to _____ . Is the abstraction function invertible?

e) Which is more important for a client of an ADT: the rep invariant, the abstraction function or the two are equally important? Explain.

Consider the **IntMap** interface and unrelated **IntStack** and **IntQueue** below. Several questions below are based on this code.

```
/** An IntMap is a mapping from integers to integers.
* IntMap can be thought of as a set of key-value pairs:
*
* @specfield pairs == { <k1, v1>, <k2, v2>, <k3, v3>, ... }
*/
interface IntMap {
/** Associates the specified value with the specified key in this map. */
bool put(int key, int val);
/** Removes the mapping for the key from this map if it is present. */
int remove(int key);
/** Returns true if this map contains a mapping for the specified key. */
bool containsKey(int key);
/** Returns the value to which specified key is mapped, or 0 if this
* map contains no mapping for the key. */
int get(int key);
}

/**
* An IntStack represents a stack of ints.
* IntStack can be thought of as an ordered list of ints:
*
* @specfield stack : List<int>
*
* stack == [a_0, a_1, a_2, ..., a_k]
*/
interface IntStack {
/** Pushes an item onto the top of this stack.
* If stack_pre == [a_0, a_1, a_2, ..., a_(k-1), a_k]
* then stack_post == [a_0, a_1, a_2, ..., a_(k-1), a_k, val].
*/
void push(int val);
/**
* Removes the int at the top of this stack and returns that int.
* If stack_pre == [a_0, a_1, a_2, ..., a_(k-1), a_k]
* then stack_post == [a_0, a_1, a_2, ..., a_(k-1)]
* and the return value is a_k.
*/
int pop();
}
```

Name: _____

2. (20pts) Willy Wazoo wants to write his own implementation for **`IntStack`**, but the only data structure he knows how to use is an **`IntMap`**! So he started out like this before he got stuck:

```
class WillysIntStack implements IntStack {
     // Represents the IntStack
     private IntMap theRep;
     // Size of the stack
     private int size;
}
```

Help Willy write a rep invariant and abstraction function for his implementation. Don't change his representation. You'll later implement **`IntStack`** using the rep invariant and abstraction function.

a) (5pts) Rep invariant

b) (5pts) Abstraction function

c) (10pts) Help Willy implement **`push`** and **`pop`** and prove (informally, as we did in class) that they preserve the rep invariant.

Name: _____

3. (10pts) Willy's on a roll! He now wants to implement an **`IntQueue`** (interface right below).

```
/**
* An IntQueue represents a queue of ints.
* IntQueue can be thought of as an ordered list of ints:
*
* @specfield queue : List<int>
*
* queue == [a_0, a_1, a_2, ..., a_k]
*/
interface IntQueue {
/** Inserts an item at the back of this queue.
* If queue_pre == [a_0, a_1, a_2, ..., a_(k-1), a_k]
* then queue_post == [a_0, a_1, a_2, ..., a_(k-1), a_k, val].
*/
void enqueue(int val);
/**
* Removes the int at the front of this queue and returns that int.
* If queue_pre == [a_0, a_1, a_2, ..., a_(k-1), a_k]
* then queue_post == [a_1, a_2, ..., a_(k-1), a_k]
* and the return value is a_0.
*/
int dequeue();
}
```

Willy started out like this and got stuck again.

```
class WillysIntQueue implements IntQueue {
     // Represents the IntQueue
     private IntMap theRep;
     private int first;
     private int last;
}
```

Help Willy write a rep invariant and an abstraction function. Do not change Willy's representation. Don't worry about overflow and underflow. You don't have to write **`enqueue`** and **`dequeue`** but they should be clear from your rep invariant and abstraction function.

a) (5pts) Rep invariant

b) (5pts) Abstraction function

Name: _____

Part IV. Testing

1. (10pts) Consider the specification for **`binarySearch`**. Several questions below concern **`binarySearch`**.

```
/**
 * 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)
```

Write 4 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):

```
@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.

Name: _____

Now, let's look at the implementation of **`binarySearch`**

```
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;
}
```

2. (8pts) Draw the control-flow graph (CFG) for the **`binarySearch`** routine.

3. (6pts) What is the % branch coverage that the 5 test cases from part 1 achieve? What is the % statement coverage?

4. (6pts) Explain why one cannot write a single test case that achieves 100% statement coverage using an array of size 4. How about with size 5?

Name: _____

Part V. Java

1. (17pts) Short answers and TRUE/FALSE answers.

a) Write the code that sets a boolean variable **`flag`** to true if your Java program is running with assertions turned on. The code sets **`flag`** to false otherwise.

b) (TRUE/FALSE) If **`B`** is a Java subtype of **`A`** then **`B[][]`** is a Java subtype of **`A[][]`**.

c) (TRUE/FALSE) If **`B`** is a Java subtype of **`A`** then **`C<B>`** is a Java subtype of **`C<A>`**.

d) (TRUE/FALSE) If **`a.equals(b)`** then **`a.hashCode()`** must equal **`b.hashCode()`**.

e) (TRUE/FALSE) If **`a.hashCode()`** equals **`b.hashCode()`** then **`a.equals(b)`** must be true.

f) Is overloading resolved at compile time or at run time? That is, when is the decision on "what method family" made, at compile time or at run time?

g) When is overriding resolved, at compile time or at run time?

Name: _____

2. (8pts) Consider the following Java hierarchy and the clients below. At the designated places, write which method gets called, or indicate compile-time or runtime error.

```
class OneDPoint {
   public boolean equals(OneDPoint other) { … }
}
class TwoDPoint extends OneDPoint {
   public boolean equals(TwoDPoint other) { … }
}

Object p1 = new TwoDPoint();
Object p2 = new TwoDPoint();
boolean b = p1.equals(p2); // Which equals is called here?

OneDPoint p1 = new TwoDPoint();
Object p2 = new TwoDPoint();
boolean b = p1.equals(p2); // Which equals is called here?

OneDPoint p1 = new TwoDPoint();
OneDPoint p2 = new TwoDPoint();
boolean b = p1.equals(p2); // Which equals is called here?

TwoDPoint p1 = new TwoDPoint();
TwoDPoint p2 = new TwoDPoint();
boolean b = p1.equals(p2); // Which equals is called here?
```

Name: _____

Several questions 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 {
```

int

```
  private char 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();
  }
}
```

Name: _____

3. (6pts) A client calls **`numDigits`**. The client is surprised when **`numDigits`** returns 11. Explain how this failure is possible.

4. (6pts) Write the smallest JUnit test that you can that exposes this problem in the implementation.

5. (8pts) 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)

Name: _____

Part VI. Design Patterns and Refactoring

1. (40pts, 2pts each) Very short (2-3 word) answers.

a) What pattern forces a class to have a single instance?

b) What patterns allow for creation of objects that are subtypes of a given type?

c) What pattern helps reuse existing objects?

d) Can interning be applied to mutable types?

e) Can a mutable class be a Singleton?

f) What design pattern represents complex whole-part objects?

g) What design pattern changes the interface of a class without changing its functionality?

h) What design pattern adds small pieces of functionality without changing the interface?

i) Decorator uses which design pattern?

j) What pattern helps restrict access to an object?

k) What is the difference between an object adapter and a class adapter?

l) Which one is more efficient, an object adapter or a class adapter?

m) What pattern hides a large and complex library and promotes low coupling between the library and the client?

n) What patterns help traverse composite objects?

o) What pattern groups unrelated traversal operations into classes in the composite hierarchy?

p) What patterns group all related traversal operations into separate classes?

q) If you anticipate the composite hierarchy to change while the set of traversal operations stays the same, what pattern would you rather use, Interpreter or Visitor?

r) Conversely, if you anticipate no changes in the composite hierarchy (e.g., BooleanExp doesn't change), but you expect addition of traversal operations, what pattern would you use, Interpreter or Visitor?

s) What pattern allows for an object to maintain multiple views that must be updated when the object changes?

t) Give an example of usage of the Composite pattern in the Java GUI library

Name: _____

Several questions below use the following code.

```
/** Exp represents, in preorder, arithmetic expressions over integers
 •  and binary operators + and -. E.g., - + 1 2 3 (is 1 + 2 – 3
 •  in inorder). Exp uses the Composite and Visitor patterns.
 */
abstract class Exp {
    abstract public void accept(Visitor v);
}
class PlusExp extends Exp {
    private Exp left;
    private Exp right;
    public PlusExp(Exp left, Exp right) {
        this.left = left;
        this.right = right;
    }
    public void accept(Visitor v) {
        left.accept(v);
        right.accept(v);
        v.visit(this);
    }
}
class MinusExp extends Exp {
    private Exp left;
    private Exp right;
    public MinusExp(Exp left, Exp right) {
        this.left = left;
        this.right = right;
    }
    public void accept(Visitor v) {
        left.accept(v);
        right.accept(v);
        v.visit(this);
    }
}
class IntExp extends Exp {
    int value;
    public IntExp(int val) {
        this.value = val;
    }
    public int getValue() {
        return value;
    }
    public void accept(Visitor v) {
        v.visit(this);
    }
}
```

Name: _____

2. (10pts) Complete the `Evaluate` visitor, which evaluates an arithmetic expression. Fill in the rep, `getResult`, `visit(PlusExp e)`, `visit(MinusExp e)` and `visit(IntExp e)`.

```
interface Visitor {
   public void visit(PlusExp e);
   public void visit(MinusExp e);
   public void visit(IntExp e);
}

class Evaluate implements Visitor {
    // Add necessary rep structure here

    // returns: the result of this evaluation
    public boolean getResult() {


    }
    public void visit(PlusExp e) {



    }
    public void visit(MinusExp e) {



    }
    public void visit(IntExp e) {



    }
}
```

int

3. (5pts) Write a JUnit test case that creates expression 5 - 10 + 2, evaluates the expression using the visitor, and asserts that the result is correct.

4. (10pts) We cannot help but notice that `Exp` has lots of duplicate code! Suggest refactorings that eliminate as much of the duplicate code as possible, without breaking the test case from question 3.