Exam 2
CSCI 2600 Principles of Software
November 4th 2014

- DO NOT OPEN THIS EXAM UNTIL TOLD TO DO SO!
- READ THROUGH THE ENTIRE EXAM BEFORE STARTING TO WORK.
- YOU ARE ALLOWED ONLY 1 “CHEAT” PAGE. NO OTHER MATERIAL IS ALLOWED.

This exam is worth 100 points.

Make sure you have 11 pages counting this one. There are 4 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. __________/20
2. __________/18
3. __________/46
4. __________/16

TOTAL: __________/100
Part I. Abstraction Functions and Rep Invariants

1) Short answers.

a) (2pts) The rep invariant maps ______________ to ____________ .

b) (2pts) The abstraction function maps ______________ to ____________ .

c) (4pts) What is the benefit of using a rep invariant that restricts concrete values so that there is one concrete value (or as few as possible), mapping to a particular abstract value? For example, RatNum in your homework had a rep invariant that restricted concrete values to rational numbers in reduced form (i.e., 2/4 was not a valid rep).

d) (4pts) Must the rep invariant of an ADT implementation hold before and after every statement in every method of the implementation? Explain.

2) (8pts) Consider the LineSegment class, partially implemented below. Write a rep invariant and an abstraction function and complete the implementation of equals.

```java
/** This helper class represents a 2D point with integer *
 * coordinates.
 */
class Point {
    int x;
    int y;
    Point(int x, int y) { // constructor
        this.x = x;
        this.y = y;
    }
}
```
/** This class represents the mathematical concept of a line segment. 
 * Specification fields:
 * start-point : point // the starting point of the line
 * end-point : point // the end point of the line
 */
class LineSegment {
    Point start;
    Point end;

    // Rep invariant:

    // Abstraction function:

    public LineSegment(Point start, Point end) {
        this.start = start;
        this.end = end;
    }

    /** Returns true if and only if the two objects represent the same line segment. */
    public boolean equals(}
Part II. Testing

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

   a) (True/False) 100% statement coverage implies 100% branch coverage.
   b) (True/False) 100% branch coverage implies 100% statement coverage.
   c) (True/False) When a JUnit test fails, the programmer must fix the program being tested.
   d) (True/False) Suppose you have two correct implementations, $I_1$ and $I_2$ of the same spec. A correct white-box test written for $I_1$ will pass when run on $I_2$.
   e) (True/False) Suppose you have two correct implementations, $I_1$ and $I_2$ of the same spec. A correct black-box test written for $I_1$ will pass when run on $I_2$.

The questions below use this code:

```c
int f(int y) {
    int x = 0;
    int s = 0;
    while (x < y) {
        x = x+3;
        y = y+2;
        if (x+y < 10) {
            s = s+x+y;
        } else {
            s = s+x-y;
        }
    }
    return s;
}
```

2) (4pts) Draw the control-flow graph (CFG) for $f$. Use the space above, next to the code. You are given the entry and exit nodes of the CFG.

3) (4pts) Write a minimal test suite that achieves 100% branch coverage. Note: a minimal test suite contains fewer test cases than any other test suite.
Part III. LSP, Java Subtyping and Equality

1) (12pts, 2pts each) True/False questions.
   a) (True/False) An overriding method in Java must have return type, which is the same or a subtype of the return type of the method it overrides.
   b) (True/False) An overriding method must have parameters that are the exact same type as the parameters of the method it overrides.
   c) (True/False) A Java subtype is not necessarily a true subtype.
   d) (True/False) Java subtyping guarantees that at call a.m() a is non-null.
   e) (True/False) String f(Object) is a subtype of Object f(String).
   f) (True/False) int f(String) is a subtype of int f(Object).

2) (8pts, 4pts each) Short answers.
   a) It is dangerous and confusing when a Java subtype is not substitutable for its supertype. Why?

   b) Is Triangle a true subtype of Polygon? Explain your answer. Note: numVertices is a derived specification field.

```java
class Polygon {
   // effects: adds Point p to vertices of this Polygon:
   //          this_old.numVertices + 1 = this_new.numVertices
   //          if p is not already in
   void addVertex(Point p);
   // returns: number of vertices in this Polygon
   int numVertices();
}

class Triangle extends Polygon {
   // effects: does nothing; in particular:
   //          this_old.numVertices = this_new.numVertices
   void addVertex(Point p) {}
}
```

CSCI 2600 Exam 2, Fall 2014
3) (4pts) Consider the following Java hierarchies and the clients below. At the designated places, write which method gets called, or indicate compile-time or runtime error.

```java
class Y extends X { ... }
class A {
    X m(Object o) { ... }
    X m(Z z) { ... }
}
class B extends A {
    X m(Z z) { ... }
}
class C extends B {
    Y m(Z z) { ... }
}
A a = new B();
Object o = new Z();
X x = a.m(o); // Which m is called here?

A a = new C();
Z z = new Z();
X x = a.m(z); // Which m is called here?
```

4) (4pts) Now consider these slightly different Java hierarchies and the clients below:

```java
class Y extends X { ... }
class W extends Z { ... }
class A {
    void m(Z z, Y y) { ... }
}
class B extends A {
    void m(W w, X x) { ... }
}
class C extends B {
    void m(W w, X x) { ... }
}
A a = new B();
Z z = new W();
Y y = new Y();
a.m(z, y); // Which m gets called here?

B b = new C();
W w = new W();
X x = new Y();
b.m(w, x); // Which m gets called here?

B b = new C();
W w = new W();
Y y = new Y();
b.m(w, y); // Which m gets called here?
```
Several questions below are based on the following code for Timestamp:

```java
class Timestamp {  
    int min;  
    int sec;  
    public boolean equals(Object o) {  
        if (o instanceof Timestamp) {  
            Timestamp t = (Timestamp) o;  
            return t.min == min && t.sec == sec;  
        }  
        else  
            return false;  
    }  
}  
```

5) (4pts) NanoTimestamp extends Timestamp:

```java
class NanoTimestamp extends Timestamp {  
    int nano;  
    public boolean equals(Object o) {  
        if (o instanceof NanoTimestamp) {  
            NanoTimestamp n = (NanoTimestamp) o;  
            return super.equals(n) && n.nano == nano;  
        }  
        else  
            return false;  
    }  
}  
```

Indicate which of the following properties holds for the equals method. You must consider the method family equals, not just NanoTimestamp.equals.

(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.
6) (4pts) Another implementation of \texttt{NanoTimestamp.equals}:

\begin{verbatim}
public boolean equals(Object o) {
    if (o instanceof Timestamp) {
        return super.equals(o);
    }
    else if (o instanceof NanoTimestamp) {
        NanoTimestamp n = (NanoTimestamp) o;
        return super.equals(n) && n.nano == nano;
    }
    else
    return false;
}
\end{verbatim}

Again, indicate which of the following properties holds for the \texttt{equals} method.

(a) Is \texttt{equals} reflexive? If not, give a counterexample below.

(b) Is \texttt{equals} symmetric? If not, give a counterexample.

(c) Is \texttt{equals} transitive? Again if not, give a counterexample.

7) (4pts) Yet another implementation of \texttt{NanoTimestamp.equals}:

\begin{verbatim}
public boolean equals(Object o) {
    if (o instanceof NanoTimestamp) {
        NanoTimestamp n = (NanoTimestamp) o;
        return super.equals(n) && n.nano == nano;
    }
    else if (o instanceof Timestamp) {
        return super.equals(o);
    }
    else
    return false;
}
\end{verbatim}

(a) Is \texttt{equals} reflexive? If not, give a counterexample below.

(b) Is \texttt{equals} symmetric? If not, give a counterexample.

(c) Is \texttt{equals} transitive? Again if not, give a counterexample.
8) (6pts) Consider the following implementations of `Timestamp.hashCode`:

(a) int hashCode() { return super.hashCode(); }  
(b) int hashCode() { return 42; }  
(c) int hashCode() { return min+sec; }  
(d) int hashCode() { return min*min; }  
(e) int hashCode() { return 31*min+sec; }  

Which implementations of `hashCode` are inconsistent with `Timestamp.equals` (given earlier, on page 7)?  

Rank the consistent implementations from best to worst:  

Ranking: __, __, __, __, __  

Part IV. Design Patterns  

1) Short answers.  

a) (4pts) This code attempts to implement a Singleton `ServiceFactory` class, but it is doing something wrong. Explain what is wrong.  

```java  
class ServiceFactory {  
   IAccountingService accountingAdapter;  
   private ServiceFactory() {  
      // initializes adapter using dependency injection  
   }  
   private static ServiceFactory instance;  
   public static ServiceFactory getInstance() {  
      if (instance == null)  
         instance = new ServiceFactory();  
      else  
         return instance;  
   }  
   public Object clone() {  
      ServiceFactory o = new ServiceFactory();  
      o.accountingAdapter = accountingAdapter.clone();  
      return o;  
   }  
}  
```
b) (4pts) REMOVED, was a bad question.

c) (2pts) What pattern(s) allows a client to treat collection objects and unit objects uniformly, i.e., through a common interface?

d) (2pts) What pattern changes the interface of an existing class to an expected, different interface?

e) (4pts) We cannot use HashSet (as attempted in the erroneous code below) to implement String interning. Why? Fix the code by providing the correct data structure.

```java
static HashSet<String> names = ...  
String canonicalName(String n) {
    if (names.contains(n))
        return n;
    else {
        names.add(n);
        return n;
    }
}
```