15-214: Principles of Software Construction
8th March 2012

Name: 
Recitation Section (or Time):

Instructions:
• Make sure that your exam is not missing any sheets (it should contain 13 pages).
• Write your full name on this page and Andrew ID on the header of all.
• Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
• The problems are of varying difficulty. The point value of each problem is indicated.
• Ideally you should take one minute per point. So pace yourself accordingly.
• This exam is CLOSED BOOK. You may not use a calculator, laptop or any other electronic or wireless device.
• Write concise and focused answers (long ramblings will hurt your grade).
• If anything is unclear, just make and state your (reasonable) assumptions.
• Make sure your handwriting is READABLE.

You have 80 minutes to complete this Exam. Good luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Unit Testing</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Method Dispatch</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Design Patterns</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td></td>
<td><strong>79</strong></td>
</tr>
</tbody>
</table>
Java [15 points]

Q1. [5 points] Fill in the blanks:
   a) [interface] is the feature that allows different implementations of the same interface to behave differently.
   b) [polymorphism] is the capability of a class to manifest the properties and methods of another class while adding its own functionality.
   c) Java permits a class to replace the implementation of a method that it has inherited. It is called [override] keyword assigned to a class member to hide that member from all other classes.
   d) The main method is defined to be [static] because it is not a property of an instance but of the class.

Q2. [2 points] Print out the result:
   ```java
   String a = new String("4");
   String b = new String("4");
   System.out.print( a.equals(b) );
   System.out.print( a == b );
   ```

Q3. [4 points] Find two things that cause this code not to compile? Mark the errors.
   ```java
   public final abstract class Animal{
   public void speak(){
      System.out.println("My name is " + getName() + ".");
   }
   public String getName();
   }

   public class Dog extends Animal{
   public String getName(){
      return "Tommy";
   }
   public static void main(String[] args){
      Animal d = new Dog();
      d.speak();
      d.getName();
   }
   }
   ```

Q4. [4 points] Given below is a portion of the inheritance hierarchy of the various errors and exceptions. Please use this to answer the following questions.
   ```java
   class java.lang.Throwable (implements java.io.Serializable)
   class java.lang.Error
   class java.lang.AssertionError
   class java.lang.LinkageError
   class java.lang.ClassCircularityError
   class java.lang.ClassFormatError
   class java.lang.ClassInitializationError
   class java.lang.IncompatibleClassChangeError
   class java.lang.AbstractMethodError
   class java.lang.IllegalAccessError
   class java.lang.InstantiationException
   class java.lang.NoSuchFieldError
   class java.lang.NoSuchMethodError
   ```
class java.lang.NoClassDefFoundError
class java.lang.UnsatisfiedLinkError
class java.lang.VerifyError
class java.lang.ThreadDeath
class java.lang.VirtualMachineError
class java.lang.InternalError
class java.lang.OutOfMemoryError
class java.lang.StackOverflowError
class java.lang.UnknownError
class java.lang.Extension
class java.lang.ClassNotFoundException
class java.lang.CloneNotSupportedException
class java.lang.IllegalAccessException
class java.lang.InstantiationException
class java.lang.InterruptedException
class java.lang.NoSuchFieldException
class java.lang.NoSuchMethodException
class java.lang.RuntimeException

class java.lang.ArithmeticException
class java.lang.ArrayStoreException
class java.lang.ClassCastException
class java.lang.IllegalArgumentException
class java.lang.IllegalThreadStateException
class java.lang.NumberFormatException
class java.lang.IllegalMonitorStateException
class java.lang.IllegalStateException
class java.lang.IndexOutOfBoundsException
class java.lang.ArrayIndexOutOfBoundsException
class java.lang.StringIndexOutOfBoundsException
class java.lang.NegativeArraySizeException
class java.lang.NullPointerException
class java.lang.SecurityException
class java.lang.UnsupportedOperationException

a) What does the following program print?

```java
public static void main(){
    try{
        String s = "";
        Char b = s.charAt(5);
    } catch ( Exception e ){
        System.out.println("caught an exception");
    } catch ( Error e ){
        System.out.println("caught an error");
    }
}
```

b) What does the following program output?

```java
public static void main(){
    try{
        assertTrue(false);
    } catch (Exception e){
        System.out.println("caught an exception");
    } catch (Error e){
        System.out.println("caught an error");
    }
}
```
Unit Testing [10 points]

You are to develop a set of unit test cases for the following code block.

Note that the code below may contain some faults. The test cases you write should catch these faults and any others that might arise as the code evolves.

```java
public class AccumValue {

    private int value;

    public AccumValue(int init){ value = init; }

    /**
     * Computes a new accumulation value from the values in vs up to limit
     * @returns true if value changed, false otherwise.
     */
    public boolean calcAccum ( int limit, AccumValue[] vs ){
        int[] array = new int[limit];
        if( array.length < limit )
            return false;

        buildArray(array,vs);

        int tmp=0;
        for( int s : array )
            tmp += s;

        int old = value;
        value = tmp;
        return old != value;
    }

    protected void buildArray( int[] array, AccumValue[] vs ){
        for( int i=0 ; i<array.length ; ++i )
            array[i] = vs[i].getValue();
    }

    public int getValue(){ return value; }
}
```
Q1. [10 points] Fill in the blanks to produce several test cases that check the code shown above. 
If a test case should throw an exception you should write it as:

```java
@Test(expected = NameOfException.class)
```

where NameOfException is a reasonable exception that the code should throw for that situation. If the test case should not throw any exception then either cross out the blank box or leave it empty.

```java
public class AccumValueTester {

    @Test
    public void testConstructor() {
        assertEquals(214, new AccumValue(214).getValue());
    }

    @Test
    public void testArgumentsSanityCheck1() {
        new AccumValue(0).calcAccum(1, 2);
    }

    @Test
    public void testArgumentsSanityCheck2() {
        new AccumValue(0).calcAccum(1, 2);
    }

    @Test
    public void testArgumentsSanityCheck3() {
        new AccumValue(0).calcAccum(1, 2);
    }

    @Test
    public void testCorrectResult() {
        AccumValue c = new AccumValue(0);
        AccumValue[] x = { new AccumValue(2), new AccumValue(3) };
        assertTrue(true);
        assertEquals(5, c.getValue());
    }
}
```
Method Dispatch [16 Points]

Consider the following code:

```java
package b;
abstract class Bird {
    protected int a = 1;
    public abstract void identify();
    protected void shout(){
        System.out.println("Kaw-Kaw");
    }
    public void action(){
        System.out.println("Fear me "+ this.a +" times");
    }
    public void flyAround(){
        System.out.println("Wooooosh, I am a ");
        identify();
        shout();
        System.out.println("You should ");
        action();
    }
    protected int getA(){ return a; }
    protected void addToA(){ a++; }
}

package b;
public class Penguin extends Bird{
    private int a;
    public Penguin(){ a = 2; }
    @Override
    public void identify(){
        System.out.println("Penguin "+ a +" reporting for duty!");
    }
    public void shout(){
        System.out.println("Shouting is uncivilized...");
    }
    public void action(int times){
        System.out.println("Swim "+ (times + getA()) +" times!");
    }
    public void flyAround(){
        System.out.println("Sometimes I dream I can fly like this: ");
        super.flyAround();
    }
}

package a;
public class Program{
    public static void act1(){
        Bird b = new Bird();
        b.flyAround();
    }
}
```
public static void act2(){
    Bird b = new Penguin();
    act3(b);
}

public static void act3(Penguin p){
    Bird b = new Penguin();
    b.shout();
}

public static void act4(){
    Bird b = new Penguin();
    b.action(15);
}

public static void act5(){
    Bird b = new Penguin();
    b.flyAround();
}

public static void act6(){
    Bird b = new Penguin();
    b.identify();
}

public static void act7(){
    Bird b = new Penguin();
    b.addToA();
    ((Penguin)b).action(4);
}

Q1. [16 points] For each of the following methods that refer to the code above, if there is a compilation error, explain what it is and how to fix it. If there is no compilation error in a particular act method, say what the method prints.

a) Program.act1()

b) Program.act2()
c) Program.act3(new Penguin())

d) Program.act4()

e) Program.act5()

f) Program.act6()

g) Program.act7()
The world-renowned hacker, "NotYour214TA", has some tricks up his sleeve. In order to protect his code from being understood by other he changes around all of the names so that other hackers can't read it. Little does he know at Carnegie Mellon, we laugh at such simple problems!
Q1. [12 points] There are 3 key design patterns that are used in the code above. State these patterns and describe which classes play which roles in those patterns.

*Hint 1*: CHOOSE FROM THESE PATTERNS: singleton, facade, adapter, strategy, proxy, composite, observer, factory method, template, decorator.

**PATTERNS THAT ARE NOT IN THIS LIST WILL NOT RECIEVE CREDIT**

*Hint 2*: A single class may be used in multiple patterns.

<table>
<thead>
<tr>
<th>Pattern #1:</th>
<th>Role of each relevant class in pattern #1:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Pattern #2:</th>
<th>Role of each relevant class in pattern #2:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Pattern #3:</th>
<th>Role of each relevant class in pattern #3:</th>
</tr>
</thead>
</table>
Q2. [4 points] Name the most appropriate pattern for each purpose:

*Hint:* CHOOSE FROM THESE PATTERNS: singleton, facade, adapter, strategy, proxy, composite, observer, factory method, template method, decorator.

a) add functionality to individual objects dynamically and transparently

b) expose the functionality of an object through another interface

c) fix the structure of an algorithm but allow portions of that algorithm to vary

d) load an object from a database on demand
Verification [22 points]

Q1. [8 points] Compute the weakest precondition.
Assume integer operations. Simplify the result as much as possible.
Important note: Assume all variables and operations range over integers.

a) \( \text{wp} ( \texttt{r} = \texttt{x}; \texttt{n} = \texttt{n-1} ', \texttt{ans} = \texttt{x^n} \times \texttt{r}) \)

b) \( \text{wp} ( \texttt{x} = \texttt{x}; \texttt{n} /\!\= 2 ', \texttt{ans} = \texttt{x^n} \times \texttt{r}) \)

Q2. [6 points] Always (all models), Never (no models), or Sometimes (some models)

a) True => False
b) False => False
c) False => True
d) False => x=1
e) x=1 => False
f) x=1 => True
Q3. [8 points] The weak and the strong. Indicate your answer by circling an assertion or by notating "None of these."

a) Which assertion is weakest?
   - x=1
   - Odd(x)
   - True
   - None of these

b) Which assertion is strongest?
   - x=1
   - Odd(x)
   - True
   - None of these

C) Which assertion is strongest?
   - x=1
   - Odd(x)
   - False
   - y=1
   - None of these

d) Which assertion is weakest?
   - x=1
   - Odd(x)
   - False
   - y=1
   - None of these