Prelim 1, Solution

CS 2110, 13 March 2018, 7:30 PM

	1	**2**	**3**	**4**	**5**	**6**	**Total**
Question	Name	Short answer	Exception handling	Recursion	OO	Loop invariants	
Max	1	30	11	14	30	14	100
Score							
Grader							

The exam is closed book and closed notes. Do not begin until instructed.

You have **90 minutes**. Good luck!

Write your name and Cornell **NetID**, **legibly**, at the top of **every** page! There are 6 questions on 10 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your pages are still stapled together. If not, please use our stapler to reattach all your pages!

We have scrap paper available. If you do a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam so that we can make sense of what you handed in.

Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the space provided.

In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to make the problem more difficult. This does not mean that it's good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam. I will not talk about the exam with anyone in this course who has not yet taken prelim 1.

(signature)

1. Name (1 point)

Write your name and NetID, **legibly**, at the top of **every** page of this exam.

2. Short Answer (30 points)

(a) 6 points. Below are six expressions. To the right of each, write its value.

1. `'b' == (int)'b'` true char is numerical type
2. `(char)('A' + 1)` 'B' char is a numerical type
3. `new Integer(4) == new Integer(4)` false These are two separate objects.
4. `(new Double(1.5 + 2.0)).equals(new Double(2.5 + 1.0))` true
5. `"CS2110 is great".substring(4).substring(1, 5)` "0 is"
6. `k == -1 || 3 / (k + 1) != 4` (Note: k is of type int) true

(b) 5 points. State whether each of the following statements is true or false.

1. `String y= "Good Evening"; y= '6';` will compile. false. Assigns a char to a String.
2. The following declares a two-dimensional array of Strings: `String[2][3] myStrings;` false.
3. "Generic types" refers to the built-in Java types: byte, short, int, long, float, double, boolean, char. false these are "primitive types".
4. It is possible for a method to be both overridden and overloaded. true.
5. One purpose of wrapper class Double is to treat a double value as an object. true.

(c) 4 points.

Consider the following classes:

```
public class Dress1 {
  public abstract String getColor();
}
public class Dress2 extends Dress1 {
  public String getColor() {
    return "Gold and White";
  }
}
public class Dress3 extends Dress1 {
  public Dress1 makeSimilarDress() {
    Dress1 similar= new Dress1();
    return similar;
  }
}
public class Dress4 extends Dress1 {
  private String color;
  public Dress4(String c) {
    color= c;
  }
}
```

I. Which classes need to be abstract in order to compile?

II. Which classes will fail to compile even if they are abstract?

Dress1 must be abstract because it contains an abstract method. Assume it's been made abstract.

Dress2 compiles. It overrides abstract method getColor in Dress1.

Dress3 fails to compile even if it is made abstract because it contains "new Dress1()".

Dress4 must be abstract because it extends Dress1 but doesn't override method getColor.

(d) 8 points.

This function returns the sum of all the odd integers in an array:

```
/** Return the sum of the odd integers in array c (return 0 if c is null). */
public static int sumO(int[] c) {
  if (c == null) return 0;
  int sum= 0;
  for (int k= 0; k < c.length; k= k + 1) {
    if (c[k] % 2 != 0){
      sum= sum + c[i];
    }
  }
  return sum;
}
```

I. Write down, in English, at least 5 distinct test cases that you need to consider.

Solution: they must hit 5 of the categories below (multiple tests from one category count only as 1.)

0. sumO(null) is 0
1. sumO[]) is 0
2. sumO([some even integers]) is 0
3. sumO([some odd integers]) is sum of odd integers
4. sumO([some even integers and odd integers]) is sum of odd integers
5. negative odds count as odds, negative evens don't.

II. Write down, in Java, the code for these test cases. Assume that function sumO is declared in class U. Hint: to declare an array with the values 1, 2, and 3, you can use:
`new int[]{1, 2, 3}`

```
@Test
public void testSumO() {
```

assertEquals(0, U.sumO(null));
assertEquals(0, U.sumO(new int[0]));
assertEquals(0, U.sumO(new int[]{2, 4, 6}));
assertEquals(9, U.sumO(new int[]{1, 3, 5}));
assertEquals(4, U.sumO(new int[]{1, 2, 3, 4}));
assertEquals(-1, U.sumO(new int[]{-1, -2}));

```
}
```

(e) 4 points. Write the algorithm for evaluating the new-expression `new CA(4)`.
1. Create a new object of class CA.
2. Execute the constructor call new CA(4).
3. Use as value of the new-expression the name of (pointer to) the new object —the stuff that was written in the tab of the new object.

(f) 3 points.

In A3, the doubly-linked list class DLLList had these fields:

```
private Node first; // first node of linked list (null if size is 0)
private Node last;  // last node of linked list (null if size is 0)
private int size;   // Number of values in the linked list.
```

while inner class Node had these fields:

```
private Node prev; // Previous node on list (null if this is first node)
private E val;     // The value of this element
private Node next; // Next node on list. (null if this is last node)
```

Change the class invariants above so that the doubly linked list is a circular doubly linked list.
The most important point is to change the definitions of prev and next. We change them to:

Previous node on list (pointer to node last if this is node first)
Next node on list (pointer to node first if this is node last)

Now it is truly a circular doubly linked list. We did not look at whether you changed the defs of first and last because they were not thaeimportant point. However, one can quibble with the current definitions of first and last, because there is no longer the concept of a "first" node and a "last" node. One can change this in several ways.

1. One could say that the so-called "first" node could be changed at any time to any node, since the list is circular.
2. One could change the def of first to "any node of the list" and change last to say it contained first.prev (null if first is null).
3. One could change the def of first to "any node of the list" and delete field last completely! One needs only one pointer into the list.

3. Exception handling (11 Points)

(a) 8 points. What-input-is-needed-to-get-output. Using the given class and procedure, answer the questions to the right, providing an appropriate procedure call as needed. Write "none" if no procedure call will give the desired output.

```
public class R {
  public static void b(int k, String s) {
    int x= 0;
    int y= 0;
    try {
      System.out.println("1");
      y= s.length();
      System.out.println("2");
      x= k / y;
      System.out.println("3");
    } catch(NullPointerException npe) {
      System.out.println("4");
      x= k / (k-3);
      System.out.println("5");
    } catch(RuntimeException re) {
      System.out.println("6");
      try {
        int z= k / (y-4);
        System.out.println("7");
      } catch(RuntimeException r) {
        System.out.println("8");
      }
    }
    System.out.println("9");
    int z= k / (k-6);
    System.out.println("10");
  }
}
```

2 points per option (all-or-nothing)

Give one call of procedure b that will print the following:

```
1
4
5
9
10
```

b(0, null);

What call on b does not print `"10"`?

b(6, null);

What call on b prints this:

```
1
2
3
9
10
```

b(1, "x");

What call on b will result in a thrown exception?

b(3, null);

(b) 3 points. Executing a try-statement. Write the algorithm (in English) for executing the following `try`-statement, which is not within another try-block.

`try { S3 } catch (ArithmeticException e) { S4 } .`

Execute S3. If S3 throws an ArithmeticException, execute S4. If S3 throws any other exception, throw it out to the call of the method that contains this try-statement. (The previous sentence is necessary. Without it, one does not know what happens if S3 throws any other exception.)

4. Recursion (14 Points)

(a) 6 points Execute the three calls birrellS(1); birrellS(5); and birrellS(8); and write the return value of the calls in the places provided below.

```
public static int birrellS(int n) {
  if (n <= 1) return 1;
  if (n % 2 != 0) {
    return n * birrellS(n - 1);
  }
  return birrellS(n - 1);

}
```

Return value for `birrellS(1)`: 1
Return value for `birrellS(5)`: 15
Return value for `birrellS(8)`: 105

(b) 8 points Consider the following class representing Rhinos. Write the body of recursive procedure `numCountry`. **You must use recursion; do not use a loop!**

```
public class Rhino {
    private String co; // The country in which this Rhino was born. Not null
    private Rhino parent; // null if this Rhino has no known parent

    /** Constructor: an instance born in country c with parent p.
      * Precondition: c is not null. */
    public Rhino(String c, Rhino p) {
        co= c;  parent= p;
    }

    /** Return the number of Rhinos in this Rhino's family that were
      * born in country c. This Rhino's family consists of this Rhino, its
      * parent, its parent's parent, its parent's parent's parent, etc.  */
    public int numCountry(String c) {


        int k= co.equals(c) ? 1 : 0;
        return parent == null ? k : k + parent.numCountry(c);
    OR
         if (parent == null) { return co.equals(c) ? 1 : 0; }
         return (co.equals(c) ? 1 : 0) + parent.numCountry(c);
    YOU CAN ALSO WRITE IT WITHOUT USING A CONDITIONAL EXPRESSION


    }
}
```

5. Object-Oriented Programming (30 points)

Below is class Event, which you will be using throughout this problem:

```
public class Event {
  public String name;
  public String location;

  /** Constructor: Event with name n and location loc. */
  public Event(String n, String loc) {
    name= n; location= loc;
  }
}
```

(a) 4 points Complete the body of the constructor in class Athlete:

```
/** An Olympic athlete */
public class Athlete {
  private String name;
  private String country; // Country represented, null if none
  private int athleteID;


  /** Constructor: Athlete with name n, country country,
   *  and athlete ID id. */
   public Athlete(String n, String country, int id) {
     name= n;
     this.country= country; "this." is necessary
     athleteID= id;
  }

  /** Return true if this athlete represents a country. */
  public boolean isEligible() {
    return country != null;
  }
}
```

(b) 10 points Complete the body of the constructor and function isEligible in class RegisteredAthlete:

```
/** An Olympic althlete who is currently registered to compete. */
public class RegisteredAthlete
            extends Athlete implements Registered {
  private int maxE;           // Max number of events to register at one time.
  private Event[] events;     //  Athlete is competing in events
  private int numE;           // events[0..numE-1]
```

```
   /** Constructor: Newly registered athlete registered in 0 events, with
    *  name n, country country, athlete ID id. Can compete in at most max
    *  events. */
  public RegisteredAthlete(String n, String country, int id, int max) {
    super(n, country, id);
    maxE= max;
    events= new Event[maxE]; //new Event[max] also works.
    numE= 0;
  }

   /** Return true if this athlete represents a country and has registered
     * at least 1 event. */
  public @Override boolean isEligible() {
    return super.isEligible() && numE >= 1;
  }

  /** If athlete is competing in the max number of events allowed, return false.
   *  Otherwise, add e to the athletes's events and return true. */
  public boolean addEvent(Event e) {
    if (numE >= maxE) { // numE == maxE is OK
      return false;
    }
    events[numE]= e;
    numE= numE + 1;
    return true;
  }

  /** Return the registered athlete's events. */
  public Event[] getEvents() {
    return events;
  }
}
```

(c) 16 points Below is a declaration of interface Enrolled. Above, do whatever is necessary in class RegisteredAthlete to have it implement Registered.

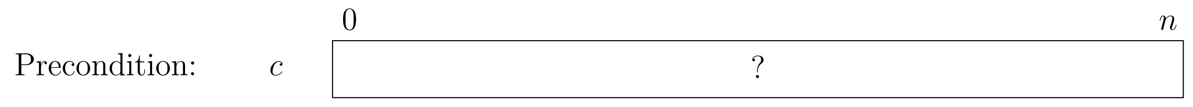
```
public interface Registered {
  /** If athlete is competing in the max number of events allowed, return false.
    * Otherwise, add e to the athlete's events and return true. */
  public boolean addEvent(Event e);

  /** Return the registered athlete's events. */
  public Event[] getEvents();
}
```

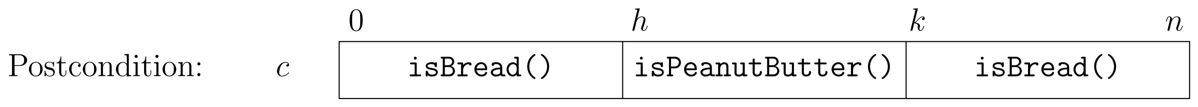
6. Loop Invariants (14 points)

We want to make a Peanut Butter sandwich. Class `Food`, below, has two methods `isBread()` and `isPeanutButter()` to determine whether an `Ingredient` is bread or peanut butter. You will use the four loopy questions to develop a single loop (with initialization) that modifies a `Food` array c to make sure all the peanut butter is on the inside of the sandwich and the bread is on the outside.

```
class Food {
   private boolean bread;
   public Food (boolean b) { bread= b; }
   public boolean isBread() { return bread; }
   public boolean isPeanutButter() { return !bread; }
}
```

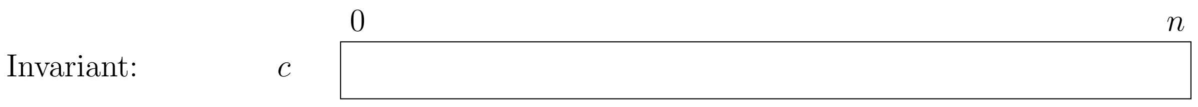
(a) 6 points Consider this precondition and postcondition for an array c of `Food` objects.





`Sizes of c[0..h-1] and c[k..n] differ by at most 1`

Complete the invariant below to generalize the above array diagrams. You will have to introduce a new variable. Place your variables carefully; ambiguous answers will be considered incorrect. Note: Several different invariants can be drawn; draw any one of them.



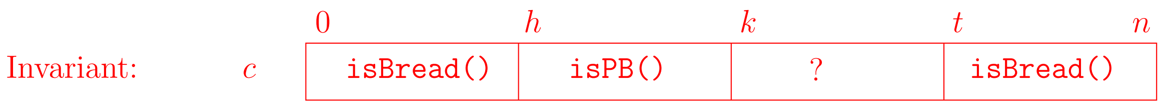
Sample Answer: This problem is easier than the Dutch National Flag in that the first and last sections have the same property (elements are Bread). It is harder in that the sizes of Bread sections differ by at most 1.

Many did not copy the part of the invariant that the sizes of the two Bread sections differ by at most 1. Some complained, saying that the question did not tell them to do this. But think this way. One loopy question asks you to prove this:

invariant AND falsity of loop condition IMPLY postcondition.

How can you prove this if that part of the postcondition is not in the invariant?

Some drew a variable right above a vertical line, creating ambiguity. Please don't do that! Below, note how 0 is to the right of a line and all others to the left of a line.



`Sizes of c[0..h-1] and c[t..n] differ by at most 1`

Note that there are several other correct invariants; alternative correct solutions might have the ? section in a different place or might have the new index t on the other side of the line (or called something else). A different invariant will result in different answers to 5(b), (c), and (d).

(b) 1 point Write the initialization that truthifies the invariant.

(c) 2 points Write a while-loop condition and state which segment has to get smaller to make progress toward termination. (*Hint: make sure that when the loop condition is false, the invariant implies the postcondition.*)

(d) 5 points Write a loop body that keeps the invariant true and makes progress toward temination (*Note: use procedure* `swap(c,i,j)` *to swap array elements* $c[i]$ *and* $c[j]$.)

It is wrong to use variables in the loop (except for local variables of the repetend) that are not defined in the invariant.

```
h= 0; k= 0; t= n+1;
while (k < t) {
     if (c[k].isBread()) {
        if (h < n+1-t) { swap(c, k, h); h= h+1; k= k+1;}
        else { t= t-1; swap(c, k, t); }
     } else {
          k= k+1;
     }
}
```

The attempt given below has two successive if-statements each of which processes a ? value c[k]. It is incorrect. Suppose the ? section has ONE value in it, Bread. The first if-statement removes it from the ? section, putting it into a Bread section. Thus, the ? section now has 0 values, and the attempt in the second if-statement to process a value in it is erroneous —it may even result in an array-index-out-of-bounds exception. Thus, an else is needed, as shown above.

```
h= 0; k= 0; t= n+1;
while (k < t) {
     if (c[k].isBread()) {
        if (h < n+1-t) { swap(c, k, h); h= h+1; k= k+1;}
        else { t= t-1; swap(c, k, t); }
     }
     if (c[k].isPB()) {
         k= k+1;
    }
}
```