Prelim 1

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'`
2. `(char)('A' + 1)`
3. `new Integer(4) == new Integer(4)`
4. `(new Double(1.5 + 2.0)).equals(new Double(2.5 + 1.0))`
5. `"CS2110 is great".substring(4).substring(1, 5)`
6. `k == -1 || 3 / (k + 1) != 4` (Note: k is of type int)

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

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

(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?

(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.

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() {









}
```

(e) 4 points. Write the algorithm for evaluating the new-expression `new CA(4)`.

(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.

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
```

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

What call on b prints this:

```
1
2
3
9
10
```

What call on b will result in a thrown exception?

(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 }
```

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)`:

Return value for `birrellS(5)`:

Return value for `birrellS(8)`:

(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) {









    }
}
```

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) {




  /** 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` on the next page

```
   /** An Olympic althlete who is currently registered to compete. */
public class RegisteredAthlete
            extends Athlete                                    {
  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) {




 }

/** Return true if this athlete represents a country and has registered
     * at least 1 event. */
  public @Override boolean isEligible() {


 }
```

(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();
}
```

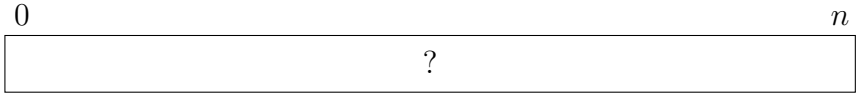
6. Loop Invariants (14 points)

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.

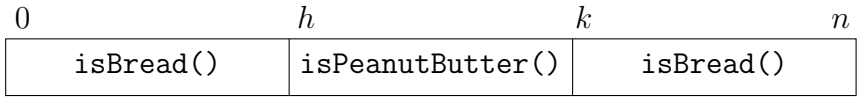
```
class Food {
   private boolean bread;
   public Food (boolean b) { bread= b; }
   public boolean isBread() { return bread; }
   public boolean isPeanutButter() { return !bread; }
}
```

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

Precondition: c



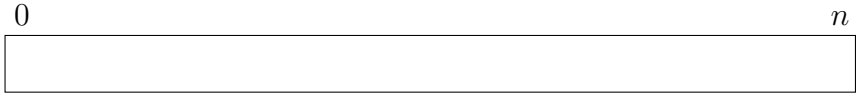
Postcondition: c



`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.

Invariant: c



(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* $c[j]$.)

```
initialization:

while (                    ) {









}
```