Prelim 1

CS 2110, 3 October 2019, 7:30 PM

	1	2	3	4	5	6	7	Total
Question	Name	Short answer	OO	Recursion	Loop invariants	Exception handling	new-exp	
Max	1	42	20	14	8	12	3	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 the first page, and your Cornell ID Number (7 digits) at the top of pages 2-7! There are 6 questions on 7 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 page 1. Write your Student ID Number (the 7 digits on your student ID) at the top of pages 2-7 (so each page has identification).

2. Short Answer (42 points)

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

1. `8 + "abc" + (1 + 2)`
2. `"abcd".substring(1,3).indexOf('a')`
3. `new Double(9000.0) == new Double(9000.0)`

(b) 8 points. Circle T or F in the table below.

(a)	T	F	This statement is syntactically correct: `int x= 'a';`.
(b)	T	F	It's possible to overload methods by swapping parameter names, e.g. `foo(int x, char y)` and `foo(int y, char x)`.
(c)	T	F	`b[3]` refers to the third item in array `b`.
(d)	T	F	Since `Comparable` is an interface, you can't have a variable of type `Comparable`.
(e)	T	F	Any call of this method will run without errors: `static boolean hasA(String x) { return x.contains("A"); }`
(f)	T	F	If no objects of a class exist, a non-static method of the class cannot be called.
(g)	T	F	Access modifier `public` allows superclasses to access the variable or method, but not unrelated classes.
(h)	T	F	When writing a class, you must write a constructor in it.

(c) 3 points. Does the code to the right compile? If not, explain why. Give your answer directly below. Specifications have been removed to make it easier to see the code.

```
public abstract class Dog {
  public abstract void fetch();

  public void bark() {
    System.out.println("Woof!");
  }
}

public class Corgi extends Dog {
  @Override public void bark() {
    System.out.println("Arf!");
  }
}
```

(d) 3 points To the right, class `S` has one field and a constructor. Consider this new-expression:

```
new S()
```

Below, write what evaluation of this new-expression prints.

```
public class S {
  private int a= 1;

  public S() {
     System.out.println(a);
     int a= 2;
     a= this.a + 4;
     System.out.println(a);
     System.out.println(a + this.a);
  }
}
```

(e) 8 points. Implement function `isNotReverse` according to its specification below. Do not use recursion. Do not create any more Strings.

```
/** Return true if b is NOT the reverse of c and false otherwise.
  * Precondition: b and c are not null */
public static boolean isNotReverse(String b, String c)
```

(f) 6 points. Write five (5) distinct test cases based on the specification of `isNotReverse` given in part (e) above (black box testing). Do not write formal `assertEquals` calls. Just write what `b` and `c` are in each case and state what the function should do/return in each case (exception, true, or false).

(g) 8 points. Consider the declarations to the right. These statements are syntactically correct and compile:

```
B b= new B();    I2 i2= b;
```

Suppose these two statements are followed by the four statements below. For each, circle yes if it compiles and no if it doesn't, and also blot out completely the uncircled word (this may help in grading)

no yes `I1 k= (I2) i2;`

no yes `I1 k2= b;`

no yes `I1 k3= i2;`

no yes `String s= i2.toString();`

```
interface I1 { }

interface I2 { }

class A implements I2 {}

class B extends A
         implements I1, I2 {}
```

3. Object-Oriented Programming (20 points)

(a) 10 points. To the right is the beginning of class `Vehicle`, with two fields and method `visit` (which you don't have to write). Below, complete the constructor and method equals.
Note: Use `new LinkedList<>()` to create a new `LinkedList` object.

```
class Vehicle {
  private Color color;
  /** list of places to go, not null */
  private LinkedList<String> places;

  /** Add p to the places to go. */
  public void visit(String p){...}
```

```
/** Constructor: instance with color color and no places to go. */
public Vehicle(Color color){




 }

/** Return true if this and ob are objects of the same
  * class and are the same color. */
public boolean equals(Object ob){




}
```

(b) 10 points. To the right is the beginning of class `Submarine`, which extends class `Vehicle` of part (a). It has one field. Below complete the constructor, method `visit`, and method `equals`, all of which go in class `Submarine`.

```
class Submarine extends Vehicle {
  /** how deep it can go */
  private int maxDepth;
```

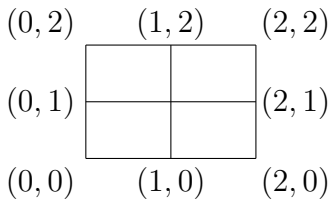
```
    /** Constructor: submarine with color c, no places to go, maxDepth d */
    public Submarine(Color c, int d) {




    }

    /** Add p to the places to go only if it is "ocean". */
    @Override public void visit(String p) {




     }

   /** Return true if this and ob are objects of the same class
       * and have the same color and max depth. */
      @Override public boolean equals(Object ob){




   }
}
```

4. Recursion (14 Points)

(a) 4 points Method `L` to the right calculates numbers in the Lucas sequence. Below, write the calls made in evaluating the call `L(3)` in the order they are called, starting with `L(3)`.

```
/** Return Lucas number n.
  *  Precondition: 0 <= n. */
public static int L(int n) {
    if (n == 0) return 2;
    if (n == 1) return 1;
    return L(n-2) + L(n-1);
}
```

(b) 10 points. To the right, we show the lower left part of a grid, but of course it extends further to the right and up to any point (x, y) with $0 \leq x$ and $0 \leq y$. A bee at any point (x, y) can fly Right to $(x + 1, y)$ or Up to $(x, y + 1)$. Suppose the bee starts at $(0, 0)$. How many different ways can the bee fly to (x, y)?

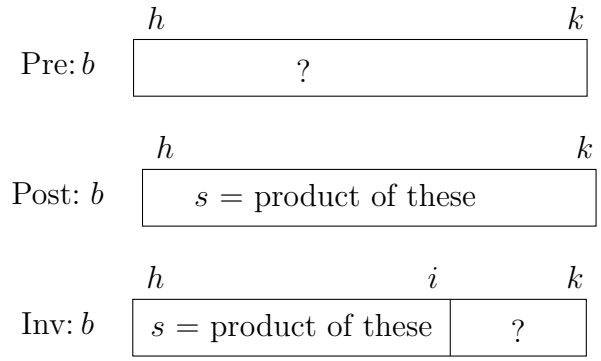


Example, for $(x, y) = (2, 1)$ there are three ways:
1: (Up, Right, Right), 2: (Right, Up, Right), 3: (Right, Right, Up).
Complete method `numWays`, below. *Use recursion and no loops.*

```
/** Return the number of ways that a bee starting at (0, 0) can
  * fly to reach (x, y), flying one step to the Right or Up as a time.
  * Precondition: 0 <= x, 0 <= y.
  * Example, if x = y = 0, return 1. If (x, y) = (2, 1), return 3. */
 public static int numWays(int x, int y) {




}
```

5. Loop Invariants (8 points)

To the right are the precondition and post-condition of a loop (with initialization) that stores in s the product of $b[h..k]$.



(a) 2 points Below, write initialization that truthifies the loop invariant. You need not declare variables.

(b) 2 points Write the loop condition B so that the loop terminates properly when B is false. Do *not* write a while loop; just write the loop condition.

(c) 4 points Write the repetend so that it makes progress toward termination and keeps the invariant true. Do *not* write a while loop; just write the repetend.

6. Exception handling (12 Points)

Consider method `mystery`. Function `s.toString()` returns the value of `s`. To the right of the method are four calls on `mystery`. Under each write (1) the output printed by the call (on one line is OK), including any exception that is *not* caught, and (2) the value returned by the call (if there is one).

```
public static int mystery(String s) {
    try {
        s= s.toString();
        System.out.println("P");
        int k= Integer.parseInt(s);
        System.out.println("Q");
        int[] b= { 6, 2, 1, 9 };
        return b[k];
    } catch (NumberFormatException e) {
        System.out.println("R");
        throw new RuntimeException();
    } catch (NullPointerException e) {
        System.out.println("S");
        return 4;
    } catch (Throwable e) {
        System.out.println("T");
    }
    return 3;
}
```

(a) 3 points `mystery("2");`
Output: Return:

(b) 3 points `mystery(null);`
Output: Return:

(c) 3 points `mystery("7");`
Output: Return:

(d) 3 points `mystery("cs2110");`
Output: Return:

7. New-expression (3 Points)

Write the 3-step algorithm for evaluating the new-expression `new Thing(0)` .