Name: NetID:

Prelim 1, Solution

CS 2110, 3 October 2019, 5:30 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. `2 + 3 + "abc" + 1 + 4` "5abc14" Operators are evaluated left to right
2. `"abc".substring(0,1).indexOf('b')` -1 The substring is "a" so there is no 'b' in the string.
3. `new Integer(10000) == new Integer(10000)` false These are two separate objects.

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

(a)	T	F	Because of automatic conversion, `int x= 1.0;` is valid Java. false A double can't be converted to an int without an explicit cast.
(b)	T	F	It's possible to overload methods by swapping arguments of different types, e.g. `foo(int x, char y)` and `foo(char y, int x)`. true The order of arguments is part of the method signature.
(c)	T	F	`a[a.length]` refers to the last item in the array `a`. false The last index is `a.length-1`.
(d)	T	F	An object whose class implements `Comparable` can be stored in a variable of type `Comparable`. true Polymorphism in Java allows this.
(e)	T	F	The method `int getLength(String x) { return x.length(); }` can throw a `NullPointerException`. true if `x` is `null`, calling `length()` on it will do so.
(f)	T	F	A static method `m` can be called only from other static methods. false Static method `m` can be called from a non-static method using `ClassName.m()`.
(g)	T	F	Access modifier `private` denies access to unrelated classes but allows superclasses to access the variable or method. false `private` denies access to all other classes.
(h)	T	F	A class cannot have more than one constructor. false You can overload the constructor with different arguments.

(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.

No. Class Porsche doesn't implement abstract method makeNoise.

```
public abstract class Vehicle {
  public abstract void makeNoise();

  public void numDoors() {
    System.out.println("I have 4 doors");
  }
}

public class Porsche extends Vehicle {
  @Override public void numDoors() {
    System.out.println("I have 2 doors");
  }
}
```

(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.

2 3 5

```
public class S {
  private int a= 2;

  public S() {
     System.out.println(a);
     int a= 5;
     a= this.a + 1;
     System.out.println(a);
     System.out.println(a + this.a);
  }
}
```

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

```
/** Return true if b is the reverse of c and false otherwise.
  * Precondition: b and c are not null */
public static boolean isReverse(String b, String c) {
    if (b.length() != c.length()) return false;

    for (int i= 0; i < b.length(); i++) {
        if (b.charAt(i) != c.charAt(c.length() - 1 - i)) return false;
    }
    return true;


}
```

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

Here are 6 possible test cases:

1. b is null and c is not null (exception)
2. c is null and b is not null (exception)
3. b and c have different lengths (false)
4. b and c are both empty (true)
5. b and c are both non-empty, and b is the reverse of c (true)
6. b and c are both non-empty and the same length, but b is not the reverse of c (false)

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

```
B b= new B();    I1 i1= 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)

```
interface I1 { }

interface I2 { }

class A implements I1 {}

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

no [✓ yes]	`String s= i1.toString();`
no [✓ yes]	`I2 k= b;`
[✓ no] yes	`I2 k2= i1;`
[✓ no] yes	`I2 k3= (I1) i1;`

3. Object-Oriented Programming (20 points)

(a) 10 points. To the right is the beginning of class `Animal`, with two fields and method `addToDiet` (which you don't have to write). Below, complete the constructor and method equals.

```
class Animal {
  private String name;
  // list of things it eats, not null
  private ArrayList<String> diet;

  /** Add nF to this Animals diet. */
  public void addToDiet(String nF){...}
```

```
/** Constructor: instance with name name and empty diet. */
public Animal(String name){
     this.name= name;
     diet= new ArrayList<>();




 }

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

    if (ob == null || getClass() != ob.getClass()) return false;
    Animal obA= (Animal) ob;
    return obA.name.equals(name);


}
```

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

```
class Rabbit extends Animal {
  // how high it jumps
  private int height;
```

```
    /** Constructor: rabbit with name name, empty diet, jumps h */
    public Rabbit(String name, int h) {
        super(name);
        height= h;

   }

    /** Add nF to the diet only if it is "carrot". */
    @Override public void addToDiet(String nF) {
         if (nF.equals("carrot")) addToDiet(nF);


     }

   /** Return true if this and ob are objects of the same class
     * and have the same name and height. */
   @Override public boolean equals(Object ob){
         if (!super.equals(ob)) return false;
         Rabbit obR= (Rabbit) ob;
         return obR.height == height;



   }
}
```

4. Recursion (14 Points)

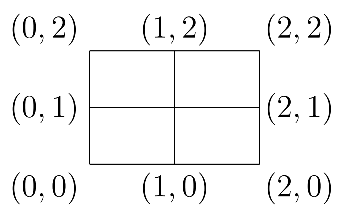
(a) 4 points Method `F` to the right calculates Fibonacci numbers. Below, write the calls made in evaluating the call `F(3)` in the order they are called, starting with `F(3)`

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

`F(3)`   `F(2)`   `F(1)`   `F(0)`   `F(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 $0 \leq y$. An ant standing at any point (x, y) can take a step Right to $(x + 1, y)$ or Up to $(x, y + 1)$. Suppose the ant starts at $(0, 0)$. How many different ways can the ant travel to (x, y)?

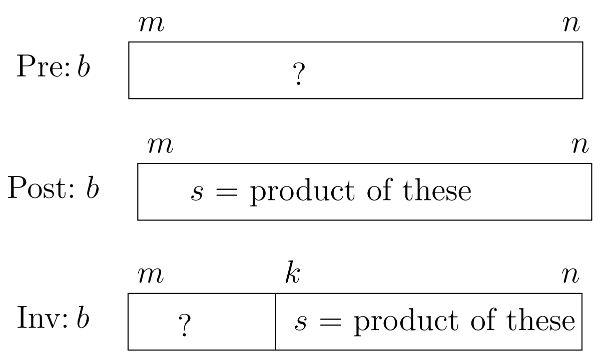
Example, for $(x, y) = (2, 1)$ there are three paths:
1: (Up, Right, Right), 2: (Right, Up, Right), 3: (Right, Right, Up).
Complete method `numPaths`, below. *Use recursion and no loops.*



```
/** Return the number of paths that an ant starting at (0, 0)
  * can take to reach (x, y), taking steps to the Right or Up.
  * Precondition: 0 <= x, 0 <= y.
  * Example, if x = y = 0, return 1. If (x, y) = (2, 1), return 3. */
 public static int numPaths(int x, int y) {
   if (x * y == 0) return 1;  // can go only R or only U or not at all
   return numPaths(x-1, y) + numPaths(x, y-1);







}
```

5. Loop Invariants (8 points)

To the right are the precondition and postcondition of a loop (with initialization) that stores in s the product of $b[m..n]$.



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

k= n+1; s= 1;

(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.

m < k OR k - m > 0 OR something similar

(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.

k= k - 1; s= s * b[k]; OR s= s * b[k-1]; k= k - 1;

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("B");
        int k= Integer.parseInt(s);
        System.out.println("C");
        int[] b= {50, 49, 49, 48};
        return b[k];
    } catch (NumberFormatException e) {
        System.out.println("D");
        throw new RuntimeException();
    } catch (NullPointerException e) {
        System.out.println("E");
        return -1;
    } catch (Throwable e) {
        System.out.println("F");
    }
    return -2;
}
```

(a) 3 points `mystery("3");`
Output: B C
Return: 48

(b) 3 points `mystery(null);`
Output: E
Return: -1

(c) 3 points `mystery("10");`
Output: B C F
Return: -2

(d) 3 points `mystery("mystery");`
Output: B D RuntimeException
Return:

7. New-expression (3 Points)

Write the 3-step algorithm for evaluating the new-expression `new ABC(5)` .

1. Create a new object of class `ABC`, with default values for fields that are not assigned in their declaration.
2. Execute the constructor call `ABC(5)`.
3. Use as value of the expression a pointer to the new object (i.e. the name that was written in the tab of the new object).