Prelim 1

CS 2110, September 29, 2016, 5:30 PM

	0	**1**	**2**	**3**	**4**	**5**	**Total**
Question	Name	Short answer	OO	Exception handling	Recursion	Loop invariants	
Max	1	29	30	10	15	15	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** at the top of **every** page! There are 5 questions on 8 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)

0. Name (1 point)

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

1. Short Answer (29 points.)

(a) 5 points. Below are five expressions. To the right of each, write its value.

1. 'a' == (int)'a'
2. (char)('a' +2)
3. new Integer(12345) == new Integer(12345)
4. ((Object)(new Integer(7))).equals(7)
5. (double)(int) 5.3

(b) 5 points. Consider these declarations of classes and interfaces:

```
public interface I1 { ... }
public interface I2 { ... }
public class C implements I1 { ... }
public class D extends C implements I1, I2 { ... }
```

Consider the statement:

```
C var= new D( ... );
```

Write down a list of all things to which variable `var` can be cast.

(c) 6 points. Put a check mark before each of the following sentences that is correct and an X before each that is incorrect.

1. A local variable can be declared static provided the method in which it is declared is static.
2. A local variable declared at the beginning of a method maintains its value from one call of the method to the next.
3. Methods of a class `C` may access private variables of an inner class of `C`.
4. If a class implements an interface, its subclasses must not implement that interface.
5. During execution of a Java program, the call stack contains at most one frame for each method.
6. Java always puts this constructor in any class `C` you write: `public C(){}`

(d) 8 points. To the right or below class C1, write the output printed by a call on method main of class C1 below. Please be extremely careful.

```
public class C1 {
    private static int p= 1;
    private int q= 2;
    private int m1(int p) { p= q+1; q= q+2; return q; }
    private int m2(int q) { p= q+1; q= q+2; return q; }

    public static void main() {
        C1 c= new C1();
        int x= c.m1(4);
        System.out.println(x + ", " + p + ", " + c.q);
        c.q= 2; p= 1;
        x= c.m2(4);
        System.out.println(x + ", " + p + ", " + c.q);
    }
}
```

(e) 5 points. Below, write an enum that has the constants AM and PM. Name the enum anything you want.

2. Object-Oriented Programming (30 points)

(a) 5 points Write down the steps in evaluating a new-expression `new C(args)` .

(b) 10 points Below are two class declarations. Complete the bodies of the constructor and function `toString` in class `Federal`. Be careful; pay attention to access modifiers.

```
public class Candidate {
   private String name;

   /** A candidate named n.
     * Precond.: no space in n */
   public Candidate(String n) {
         name= n;
   }

  /** Return name of candidate */
  public @Override String toString() {
      return name;
  }
}
```

```
public class Federal extends Candidate {
  private double contributions;

  /** Constructor: instance with name n
    *             and contributions c
    * Precond.: no space in n */
  public Federal(String n, double c) {




  }

  /** Return name of candidate,
    * a space, and the contributions. */
  public @Override String toString() {




  }
}
```

(c) 5 points Suppose the following assignment has been executed, where ... is some string that is a person's name.

```
Federal f= new Federal(..., 5.0);
```

Write a sequence of statements to extract the name of the person in object `f` and store it in String variable `v`. You don't have to declare `v` or any other variables. It doesn't matter whether you wrote methods in part (b) correctly; we go by the method specifications.

(d) 5 points Method equals, shown below, is to be placed in class `Candidate`. Complete the method body. Also, after the method body, write what happens if the type of parameter `ob` is changed to `Candidate`.

```
/** Return true iff ob is a Candidate and
  * ob has the same name as this Candidate. */
public @Override boolean equals(Object ob) {









}
```

(e) 5 points In one sentence each: (1) Explain why one makes a class abstract, (2) Explain why one makes a method in an abstract class abstract, and (3) Explain why one would use an interface instead of an abstract class with all methods abstract.

3. Exception handling (10 Points)

Execute the three calls C.m(-1); C.m(0); and C.m(1); on procedure m shown below. You know that calls on println print on the Console; place the output of the calls on println in the places provided on the right below.

```
import java.io.*;

public class C {
  public static void m(int q) {                             CONSOLE FOR C.m(-1);
     System.out.println("1: ");
     int x= q / (q + 1);
     try {
        System.out.println("2: ");
        if (q != 1) throw new RuntimeException();
        System.out.println("3: ");                          CONSOLE FOR C.m(0);
        x= q / 0;
        System.out.println("4: ");
     } catch (ArithmeticException e) {
        System.out.println("5: ");
        if (q == q) throw new RuntimeException();
        System.out.println("6: ");
     }
     catch (RuntimeException e) {
        System.out.println("7: ");                          CONSOLE FOR C.m(1);
     }
     System.out.println("8: ");
  }
}
```

4. Recursion (15 Points)

We want to compress long strings that contain many adjacent equal characters (but no digits). For example, the compression of "aaaaaaaaaaaabaaaaaazzzz" would be "a12b1a6z4". Thus, the compression of a string that contains no digits is a copy of the string but in which each sequence of adjacent equal characters (e.g. "****") is replaced by that character followed by the number of times it occurs (e.g. "*4").

Write the following two functions. Use recursion. Do not use loops. The first function is far better written using iteration, but we want to see how well you do with recursion. Remember our principle of using already written functions as much as possible in order to reduce work. You can assume that parameter `s` is not null. Do not write assert statements for Preconditions.

```
/** = if s is "", then 0; otherwise the number of times
  * the first char of s appears at the beginning of s.
  * E.g. for s = "xxxy#xxxxz", the answer is 3. */
public static int numberOfFirst(String s) {









}

/** = the compression of s (as explained above)
  * Precondition: s does not contain a digit in '0'.. '9'. */
public static String compress(String s) {









}
```

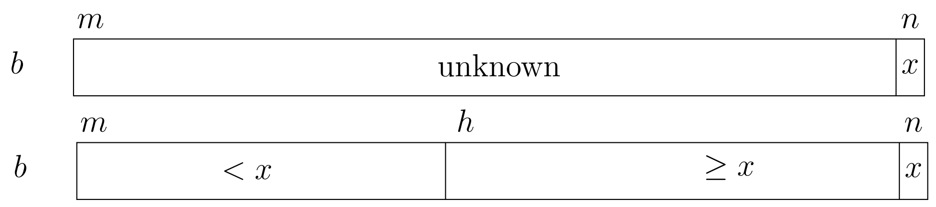
Name: NetID:

5. Loop Invariants (15 points)

(a) 6 points Consider the following precondition and postcondition.

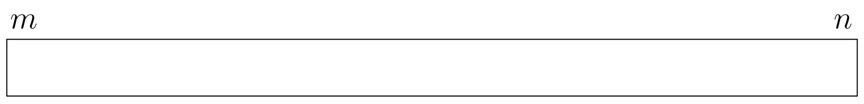
Precondition: b

Postcondition: b



Generalize the above array diagrams, completing the invariant below. Your generalization should introduce a new variable. Be sure to place your variables carefully; ambiguous answers will be considered incorrect.

Invariant: b

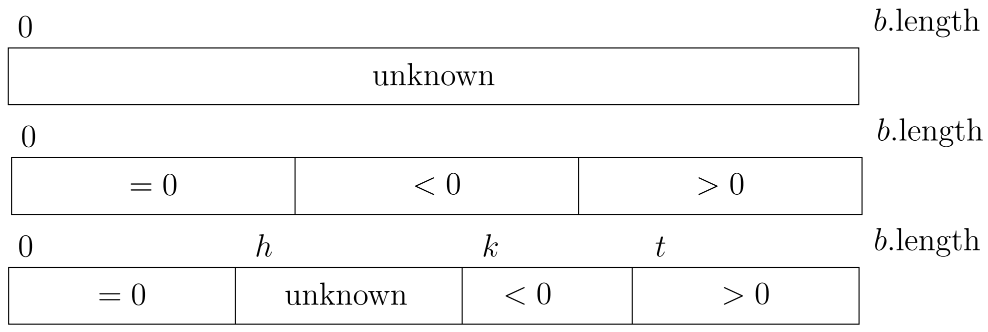


(b) 9 points Consider the following precondition, postcondition, and invariant.

Precondition: b

Postcondition: b

Invariant: b



Below, write a loop with initialization that uses the invariant given above to implement the comment given below. Assume that array b is already initialized. You do not have to declare variables, but you do have to assign appropriate values to h, k, and t wherever necessary. To swap b[] and b[j], just say, "Swap b[i] and b[j]." Your grade depends only on how well you use the four loopy questions to write the code.

```
// Given the Precondition as shown above, swap values of array b
// so that the Postcondition holds.
```