Prelim 1 with Sample Solutions

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							

1. Short Answer (29 points.)

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

1. 'a' == (int)'a' true . Remember, char is a number type.
2. (char)('a' +2) 'c' . Remember, char is a number type.
3. new Integer(12345) == new Integer(12345) false . Two different objects are created.
4. ((Object)(new Integer(7))).equals(7) true . overriding function equals in class Integer is used
5. (double)(int) 5.3 5.0 .

(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. Object, C, D, I1, I2.

(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. false. A local variable cannot be declared static.

2. A local variable declared at the beginning of a method maintains its value from one call of the method to the next. false. It is allocated space in the stack frame for the call, which is thrown away when the call terminates.

3. Methods of a class `C` may access private variables of an inner class of `C`. true. This was done in assignment A3 on linked lists.

4. If a class implements an interface, its subclasses must not implement that interface. false. An example of a subclass implementing an interface that a superclass implements appears above in part (b).

5. During execution of a Java program, the call stack contains at most one frame for each method. false. It contains a frame for each call of the method that has not terminated.

6. Java always puts this constructor in any class `C` you write: `public C(){}` false. Java inserts this constructor ONLY if you do not declare a constructor in class 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);
    }
}
```

Here are important points to notice:
1. In m1, parameter p shadows static variable p. All references to p are to the parameter.
2. In m2, parameter q shadows local variable q. All references t0 q are to the parameter.

The output is:
4, 1, 4
6, 5, 2

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

2. Object-Oriented Programming (30 points)

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

1. Create (draw) an object of class C, with default values for fields.
2. Execute the constructor call C(args).
3. Yield as value of the expression the "name" of the new object, something like C@4fabc2.

(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) {
      super(n);
      contributions= c;
  }

  /** Return name of candidate,
    * a space, and the contributions. */
  public @Override String toString() {
      return super.toString() + " " +
                 contributions;
  }
}
```

(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.

```
String r= f.toString();          // we put in declarations to
int k= r.indexOf(" ");           // make it as clear as possible
String name= r.substring(0, k);
```

(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          // Note: it would be wrong
  * ob has the same name as this Candidate. */     // to try to get the name
public @Override boolean equals(Object ob) {      // using ob.toString().
   if (!(ob instanceof Candidate)) return false;  // If ob is of class Federal,
   return name.equals(((Candidate)ob).name);    // it calls Federal's toString
}
```

If parameter ob was declared Candidate instead of Object, the method would not override method toString in class Object. Because of the annotation @Overide, the program would not compile.

(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.

1. Make a class abstract so it can't be instantiated.
2. Make a method abstract so it must be overridden in subclasses.
3. A class can extend only one other class, but it can implement many interfaces.

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 below.

```
public class C {
  public static void m(int q) {
     System.out.println("1: ");
     int x= q / (q + 1);
     try {
        System.out.println("2: ");
        if (q != 1) throw new RuntimeException();
        System.out.println("3: ");
        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: ");
     }
     System.out.println("8: ");
  }
}
```

```
CONSOLE FOR C.m(-1);
1:
```

```
CONSOLE FOR C.m(0);
1:
2:
7:
8:
```

```
CONSOLE FOR C.m(1);
 1:
 2:
 3:
 5:
```

4. Recursion (15 Points)

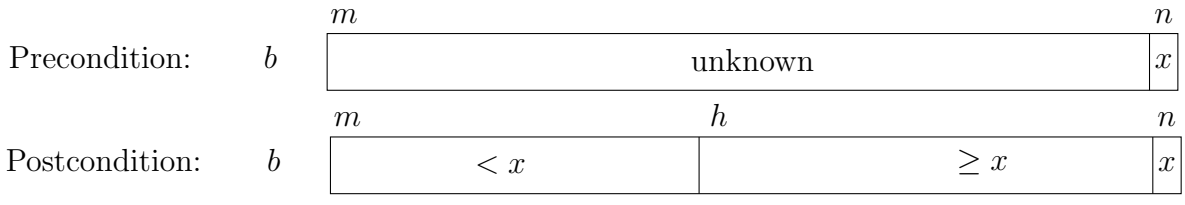
We want to compress long strings that contain many adjacent equal characters (but no digits). For example, the compression of "aaaaaaaaaaaabaaaaaazzzz" would be "a12b1a6z4". Thus, the compression of a string that contains no digits is a copy of the string but in which each sequence of adjacent equal 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.

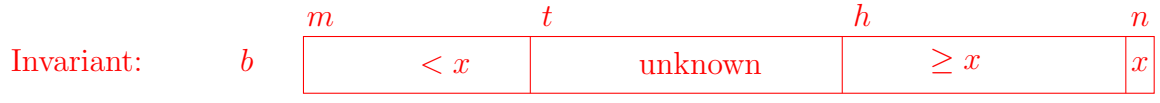
```
/** = if s is "", then 0; otherwise the number of times
  * the first char of s appears at the beginning of s.
  * E.g. for s = "xxxy#xxxxz", the answer is 3. */
public static int numberOfFirst(String s) {
    if (s.length() <= 1) return s.length();
    if (s.charAt(0) != s.charAt(1)) return 1;
    return 1 + numberOfFirst(s.substring(1));
}

/** = the compression of s (as explained above)
  * Precondition: s does not contain a digit in '0'.. '9'. */
public static String compress(String s) {
    if (s.length() == 0) return "";
    int n= numberOfFirst(s);
    return s.substring(0,1) + n + compress(s.substring(n));
}
```

5. Loop Invariants (15 points)

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



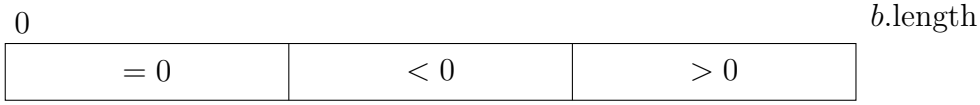
Generalize the above array diagrams, completing the invariant below. Your generalization should introduce a new variable. Be sure to place your variables carefully; ambiguous answers will be considered incorrect. Here's one answer:



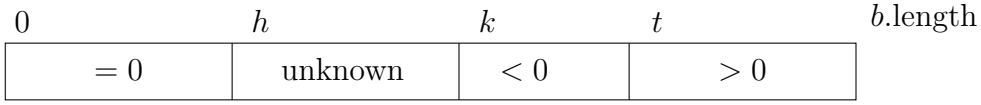
(b) 9 points Consider the following precondition, postcondition, and invariant.

Precondition:

Postcondition:



Invariant:



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.
h= 0; k= b.length; t= b.length;
while (h != k) {
     if (b[k-1] = 0) { Swap b[k-1] and b[h]; h= h+1; }
     else if (b[k-1] < 0) { k= k-1; }
     else { k= k-1; t= t-1; Swap b[k] and b[t]; }
}
```