Prelim 1, Solution

CS 2110, 27 September 2018, 7:30 PM

	1	2	3	4	5	6	Total
Question	Name	Short answer	Exception handling	Recursion	OO	Loop invariants	
Max	1	34	8	16	31	10	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 (34 points)

(a) 6 points. Each of the expressions below either (1) does not compile, (2) throws an exception at runtime, or (3) can be evaluated to produce a value. To the right of each one, state which case it is, and for (3), write down its value.

1. `(char)('0'+ 8)` '8' char is numerical type.
2. `1 + null` Error Syntax error, does not compile. + not defined for operand null.
3. `"CS" + 2110 + '1'` "CS21101" "CS" and '1' will be cast to String to perform a legal catenation.
4. `"Alan.M.Turing".substring(2).charAt(5)` `'T'`. `"Alan.M.Turing".substring(2)` evaluates to `"an.M.Turing"`, and `charAt(5)` then produces `'T'`.
5. For a variable `s` declared as `String s= new String();`,
   `s != s.substring(0);` false `s.substring(0)` returns a pointer to s, since s is immutable.
6. `11/2.0` 5.5 `1` will be cast to a double for division. No truncation is done.

(b) 5 points. Circle T or F in the table below.

(a)	T	F	Static fields cannot be accessed from non-static methods. false The inside-out rule tells you that this is possible.
(b)	T	F	Procedure calls must end with the class invariant true but function calls don't have to. false All methods should end with the class invariant true.
(c)	T	F	In a subclass constructor, you must explicitly call the superclass constructor. false You can use this(...), and if you leave that out, Java inserts one for you.
(d)	T	F	At runtime, downward type casts must be done explicitly. true Upward casts will be done automatically, not downward.
(e)	T	F	Given a class `C` that implements the `Comparable` interface, if the programmer doesn't define a `compareTo` method, Java will use the memory locations to compare two objects of `C`. false The programmer must define a compareTo method since `B` implements `Comparable`, and if not there will be a compilation error.

(c) 7 points. Use Classes Instrument and Drum below to answer the following questions.

```
public abstract class Instrument {
    public Instrument() {
        System.out.println("I am an instrument!");
    }
    public void makeSound() {
        System.out.println("curses!");
    }
}

public class Drum extends Instrument {
    public Drum() {
        super();
        System.out.println("I am a drum!");
    }

    public void makeSound(){
        System.out.println("BOOM!");
    }
}
```

Write down what will be printed in the terminal when the following code is executed.

```
Instrument i= new Drum();
i.makeSound();
```

```
I am an instrument!
I am a drum!
BOOM!
```

(d) 6 points. Implement function `differ` whose specification is given below. Do not create any new String or array objects.

```
/** Return true if the list of characters in s differs from the
  * list of characters in c. Case-sensitive. Lists are ordered.
  * Throw an IllegalArgumentException if s or c is null.
  Examples:
  char[] array1=  {'b', 'r', 'a', 'c', 'y'};
  char[] array2=  {'r', 'b', 'a', 'c', 'y'};
  differ("bracy", array1) is false
  differ("Bracy", array1) is true
  differ("bracyyy", array1) is true
  differ("bracy", array2) is true */
public static boolean differ(String s, char[] c) {


  if (s == null || c == null)
        throw new IllegalArgumentException();

    if (s.length() != c.length) return true;
    for (int i= 0; i < c.length; i++)
        if (s.charAt(i) != c[i]) return true;

    return false;

}
```

(e) 6 points. Write 4 test cases based upon the specification of `differ` (black box testing). Do not write assertEquals(...) and all that. Just give the two arguments of a call on `differ` for the test case. Answers do not have to compile, could be partly in English.

Here are 6 test cases, based on the specification.

1. c is null and s is not null
2. s is null and c is not null
3. Lengths of c and s are different
4. At some index i, c[i] and s[i] are different
5. c and s contain the same nonempty list of chars
6. c is {'A'} and s is "a" (check case sensitivity)

(f) 4 points. Write the three steps in evaluating the new-expression call `new D(v)` .

1. Create (draw) a new object of class D, with default values for the fields.
2. Execute the constructor call D(v).
3. Yield as value of the new-expression the name of (or pointer to) the new object.

3. Exception handling (8 Points)

```
public static void oof(int x, int y) {
    public static void oof(int x, int y) {
        System.out.println("A");
        int res = x / (y * 10);
        try {
            if (y == 0 || y == 1)
                throw new IllegalArgumentException();
            System.out.println("C");
            res = 1 / (y+1);
            System.out.println("D");
        }
        catch (ArithmeticException e) {
            System.out.println("E");
            if (y == -1)
                throw new RuntimeException();
            System.out.println("F");
        }
        catch (Exception e) {
            System.out.println("G");
        }

        System.out.println("H");
    }
```

What would be the output to the console for the following three statements?

1. oof(-1,-1);
   A
   C
   E
   RuntimeException

2. oof(0,0);
   A
   ArithmeticException: Division by zero

3. oof(1,1);
   A
   G
   H

4. Recursion (16 Points)

(a) 8 points Consider the following recursive function:

```
public static void sevenUp(int n) {
  if (n > 0) {
      System.out.println(n);
      if (n != 7) {
          if (n % 2 == 0) // n is even, add 2 and halve it
              sevenUp((n + 2) / 2);
          else // n is odd, multiply by 3 and subtract 3
              sevenUp(3*n - 3);
      }
  }
}
```

Execute the following calls and write down what is printed for each, but on each, stop after 4 println statements have been executed.

sevenUp(1):	sevenUp(22):	sevenUp(26):
1	22	26
	12	14
	7	8
		5

(b) 8 points Write the body of recursive procedure `spacify`. **You must use recursion; do not use a loop!** You can use function `f`, given below.

```
/** Return n as a string with a blank (i.e. a space) inserted every 3 digits.
  * Precondition: n >= 0.
  * E.g. for n = 3152463, return the string "3 152 463".  */
public static String spacify(int n) {
    if (n < 1000) return "" + n;
    return spacify(n/1000) + " " + f(n % 1000);
}

/** = m but with leading 0's, if necessary, to have at least 3 chars.
      Precondition: 0 <= m */
public static String f(int m) {
   if (m >= 1000) return "" + m;
   if (m >= 100) return "0" + m;
   return "00" + m;
}
```

5. Object-Oriented Programming (31 points)

Below is abstract class `Student`, which is used throughout this problem:

```
public abstract class Student implements Comparable                    {
   private String name; // Name of student}

   /** Constructor: a student with name n.  */
   public Student(String n) {name= n;}

   /** Return true if student passed all courses, false otherwise */
   public boolean passed() {... Assume implemented ...}

   /** Return a representation of this Student: their name. */
   public String toString() {... Assume implemented ...}

   /** Return true only if student took CS2110 and passed all courses */
   public abstract boolean canGraduate();

   public abstract void graduate();

   /** Return a negative int, 0, or positive int depending on whether
     * this students's name comes before, is the same as, or comes after
     * ob's name, in dictionary order. Throw a ClassCastException
     * if ob is not a Student.  */
   public int compareTo(Object ob) {
      Student obs= (Student) ob;
      return name.compareTo(obs.name);
   }

 }
```

(a) 5 points Recall that interface `Comparable` declares abstract function `compareTo(Object ob)`. This function is declared in class `Student`. Complete its body and make one other change in class `Student` so that it implements interface `Comparable`.

Here are two hints. (1) Casting `ob` to `Student` throws a `ClassCastException` if `ob` is not a `Student`. (2) Class `String` implements function `compareTo(String)` with essentially the same specification.

(b) 12 pts Class `Undergrad` (below) extends `Student`.

(1) Complete the body of the constructor.

(2) Complete the return statement in function `canGraduate`, assuming the boolean variable is assigned properly (you do not have to do this).

(3) Complete the body of procedure `graduate`.

```
public class Undergrad extends Student {
   private String[] courses; // list of courses taken

   /** Constructor:  an instance with name n and courses taken.*/
   public Undergrad(String n, String[] courses) {
      super(n);
      this.courses= courses;
   }

  /** Return true if (1) the Undergrad passed all courses and */
    * (2) the Undergrad took CS2110.  */
  public boolean canGraduate() {
      boolean tookCS2110= ... code to set true if CS2110 was taken;
      return passed() && tookCS2110;
  }

  /** If able to graduate, print "graduating"; otherwise throw
    * a RuntimeException with message "cannot graduate".  */
  public void graduate() {
      if (canGraduate()) System.out.println("graduating");
      else throw new RuntimeException("cannot graduate");
  }
```

(c) 5 points The following function `toString` is to be added to class `Undergrad`. Complete its body.

```
/** Return name of undergrad and number of courses taken.  */
public String toString() {
   return super.toString() + " " + courses.length;
}
```

(d) 9 points Class `CSstudent` (below) is a subclass of `Undergrad`. Each of the three methods has some kind of syntax error, so it doesn't compile. Identify the three syntax errors.

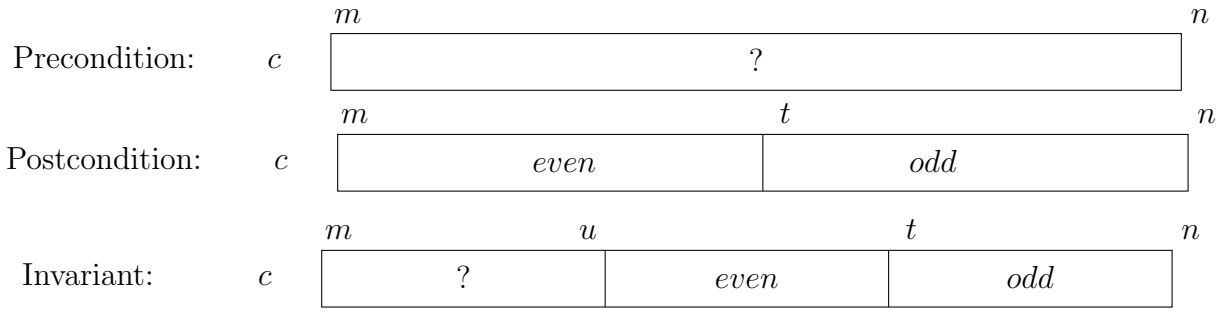
```
public class CSstudent extends Undergrad {
 /** CS student with name n and courses taken cses.  */
 public CSstudent(String n, String[] cses) {
    // Error:  no body, so super(); is inserted and there is no constructor with
no parameters in the superclass.
 }

 @Override // Error:  this is overloading, not overriding.
 public String toString(String s) {
    return s;
 }

 public static String getEm() {
    return toString("em"); //Error:  static method calls non-static method.
 }
}
```

6. Loop Invariants (10 points)

Below are the precondition, postcondition, and invariant of a loop that swaps the odd values in $c[m..n-1]$ to the end of $c[m..n-1]$. Note that m is not necessarily 0, n is not necessarily $c.length$, and both m and n should not be changed. You do not have to be concerned with declaring variables.



(a) 2 pts Given that the Precondition is true, write an initialization that truthifies the Invariant:
u= n-1; t= n;

(b) 3 points Write a while-loop condition. Make sure that if the while-loop condition is false, the Invariant implies the Postcondition. m <= u or m != u+1 or u != m-1

(c) 5 points Given that segment $c[m..u]$ is not empty, write a loop body that keeps the Invariant true and makes progress toward termination (by decreasing the size of $c[m..u]$).
Note: Use a procedure call $swap(c, i, j)$ to swap array elements $c[i]$ and $c[j]$.

```
if (c[u]%2 == 0)  u= u-1
else {t= t-1;  swap(c, u, t);  u= u-1;}
```

OR

```
if (c[u]%2 == 1) {t= t-1; swap(c, u, t)};
u= u-1;
```