Name: NetID:

Prelim 1, Solution

CS 2110, 28 September 2017, 5:30 PM

	1	2	3	4	5	6	Total
Question	Name	Short answer	Exception handling	Recursion	OO	Loop invariants	
Max	1	31	12	16	30	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 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)

1. Name (1 point)

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

2. Short Answer (31 points.)

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

1. `'a' - 'b' == 'c' - 'd'` true char is numerical type
2. `new Integer(6000) == new Integer(6000)` false Objects are different
3. `((Object)(new Integer(3))).equals(new Integer(1) + 2)` true autounboxing, equals
4. `'a' < "bcd".charAt(1) ? "yes" : "no"` "yes"
5. `"oddsAndEnds".substring(3).charAt(1)` 'A'

(b) 5 points. For each of the following statements, state whether it is true or false.

1. Private constructors are not allowed in Java. false.
2. If no constructor is declared in a class, a default constructor is inserted. true
3. String objects are immutable, so class String has no procedures for changing the value in a String object. true
4. The space for a local variable that is declared in the body of the loop is allocated whenever the loop body is executed and deallocated when the body finishes. false
5. The expression `i < 5` is evaluated exactly 3 times during execution of this loop: false

```
for (int i= 0; i < 5; i= i + 1)  i= i + 1;
```

(c) 4 points. Write the algorithm for executing the procedure call `p(1+3, 2)`.
1. Push a frame for the call onto the call stack.
2. Assign the argument values to the parameters (in the frame for the call).
3. Execute the method body (using the frame for the call for the local vars and pars).
4. Pop the frame for the call from the call stack.

(d) 6 points. Implement method `countRepeats`, below. Do not use recursion.

```
   /** Return the number of elems of b that are the same as the preceding elem.
  * Precondition: b is not null.
  * Example: countRepeats([1, 0, 0, 1])  is  1.
  * countRepeats([1, 2, 1, 1, 3, 3, 3])   is   3. */
public static int countRepeats(int[] b) {
    int count= 0;
    for (int i= 1; i < b.length; i= i+1) {
        if (b[i] == b[i - 1]) count= count + 1;
    }
    return count;
 }
```

(e) 6 points

Consider the declarations given below. In the righthand column, for each of the 6 sections of code, write whether it produces no error, a runtime error, or a compile-time error. Assume that each section is independent of the others.

Hint: It helps to draw objects.

```
interface I1 {};
interface I2 {};

class A0 {};
class A1 implements I1 {};
class A2 implements I2 {};
class A3 extends A1 implements I2 {};
class A4 extends A0 implements I1 {};
```

```
(1) A0 c1= new A4();

(2) A1 c2= new I1();

(3) I1 c3= new A3();

(4) I2 c8= (A3)(new A1());

(5) I1 c7= (A1)(new A3());

(6) I2 c4= new A4();
    I2 c6= new A2();
    c4= c6;
```

Answers to e: (1) no error (2) compile-time error (3) no error (4) run-time error (5) no error (6) compile-time error

(f) 5 points.

Consider the class definitions given below. In the righthand column, for each of the 5 sections of code, write whether it executes the method defined in C1, executes the method defined in C2, produces a compile-time error, or produces a runtime error. Assume that each section is independent of the others.

Hint: It helps to draw objects.

```
public abstract class C1 {
    public void f1(){...}
    public void f2() {...};
    public abstract void f3();
}

public class C2 extends C1 {
    public void f2() {...};
    public void f3() {...};
    public void f4() {...};
}
```

```
(1) C1 c= new C1();
    c.f2();

(2) C1 c= new C1();
    c.f3();

(3) C1 c= (C1) new C2();
    c.f1();

(4) C1 c= (C1) new C2();
    c.f2();

(5) C1 c= (C1) new C2();
    c.f3();
```

Answers to f: (1) compile-time error (2) compile-time error (3) method in C1 (4) method in C2 (5) method in C2

3. Exception handling (15 Points)

Execute the three calls E.m(-1, false); E.m(0, false); and E.m(1, false); on procedure m shown below and write the output of the calls to println in the places provided on the right. If executing a call to E.m does not result in any output, write "none".

```
public class E {
  public static void m(int x, boolean b) {
    System.out.println("1");
    int c= 1 / (x + 1);
    try {
      System.out.println("2");
      c= x / (x - 1);
      System.out.println("3");
      if (!b)
        throw new RuntimeException();
      System.out.println("4");
    } catch (RuntimeException e) {
      System.out.println("5");
    } catch (Exception e) {
      System.out.println("6");
      if (b) throw new RuntimeException();
      System.out.println("7");
    }
    System.out.println("8");
  }
}
```

Console for `E.m(-1, false)`:

1

Console for `E.m(0, false)`:

1
2
3
5
8

Console for `E.m(1, false)`:

1
2
5
8

4. Recursion (16 Points)

(a) 8 points

To the right is a declaration of a node of a singly linked list. Example: Using [4, 3] to represent a linked list with two nodes p and q, we have p.val = 4 and p.next = q, q.val = 3 and q.next = null.

```
public class Node {
    public Integer val;
    public Node next;
}
```

Write procedure change, below. Use of a loop will give you 0 points.

```
  /** p is (a pointer to) a node of a singly linked list. It may be null.
   * Change the value in p to the value in p.next (if p.next is not null)
   * and do the same for the rest of the nodes in the linked list.
   * For example, calling change on the list [3, 6, 8, 4] should change
   * the list to [6, 8, 4, 4]. */
 public static void change(Node p) {
      if (p == null || p.next == null) return;
      p.val= p.next.val;
      change(p.next);
  }
```

(b) 8 points

Write the body of the following procedure. Do not use a loop or class String. Use of either gives you 0 points.

```
  /** Return the sum of the odd digits in n.
   * Example: if n = 1045268, return 6.
   * Precondition: 0 <= n. */
 public static int sumOdd(int n) {
      if (n < 10) return  n%2 == 0 ? 0 : n;
      return sumOdd(n/10) + (n%2 == 0 ? 0 : n%10);
 OR
     if (n==0) return 0;
     if (n%2 == 1) return n % 10 + sumOdd(n/10);
     return sumOdd(n/10);
}
```

5. Object-Oriented Programming (30 points)

Below is the definition of `Student`, which you will be using throughout this problem:

```
public class Student {
  private String netID;
  private String lastName;

  /** Constructor: Student with unknown netid and last name. */
  public Student() {}

  /** Constructor: Student with netid id and last name ln. */
  public Student(String id, String ln) {
    netID= id;  lastName= ln;
  }
}
```

(a) 10 points Below are two class declarations. Complete the bodies of their constructors and function `questionCount` in class `Solution`. Be careful; pay attention to access modifiers.

```
/** An instance is student's answers
  * to an assignment. */
public class Assignment {
  private boolean[] answers;
  private Student owner;

  /** Constructor: instance with
   *  answers ans and owner s.
   *  Pre: ans is not null. */
  public Assignment(boolean[] ans,
                    Student s) {
      answer= ans;
      owner= s;
   }


  /** = the answers. */
  public boolean[] answers() {...}

  /** = the student. */
  public Student student() {...}

  /** = answer to question q. */
  public boolean getAnswer (int q) {
       return answers[q];
  }
}
```

```
/** An instance is the solution to an
 *  assignment and the max possible score
 *  on the assignment. */
public class Solution extends Assignment {
  int max; // max score on assignment

  /** Constructor: An instance with
   *  answers ans and a null owner.
   *  The max score is m.
   *  Pre: ans.length > 0. */
  public Solution(boolean[] ans, int m) {
      super(ans, null);
      max= m;
   }

  /** Return the number of correct answers
   *  in ans, where this object is the
   *  corresponding solution. */
  public int grade(Assignment ans) {...}

  /** Return the number of questions
   *  in this assignment. */
  public int questionCount() {
      return answers().length;
   }
}
```

(b) 8 points We want to implement function equals in each of classes Assignment and Solution. Below, write the body of the two methods to implement their specifications.

```
  /** This goes in class Assignment.
 *  Return true iff this and asgt are
 *  of the same class and their
 *  owners are the same object. */
 public @Override boolean
                 equals(Object asgt) {
   if (asgt == null ||
       getClass() != asgt.getClass())
     return false;
   Assignment as= (Assignment) asgt;
   return owner == as.owner;
 }
```

```
  /** This goes in class Solution.
 *  Return true iff this and sol are of
 *  the same class, their owners are
 *  the same object, and their max
 *  scores are equal. */
public @Override boolean
                 equals(Object sol) {
   if (!super.equals(sol)) return false;
   Solution so= (Solution) sol;
   return max == so.max;
 }
```

(c) 6 points Below is a declaration of class Grader. Complete its constructor.

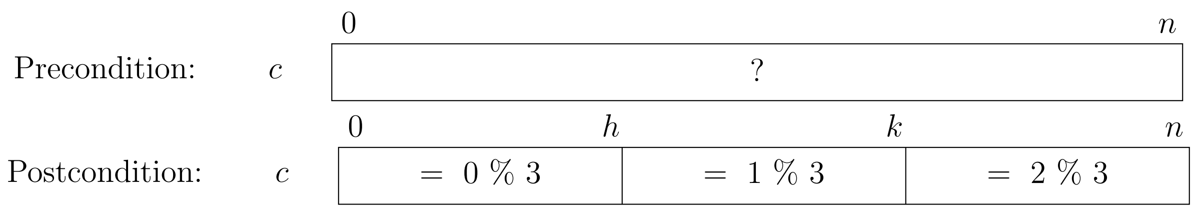
```
/** An instance is a grader for an assignment
 *  for a number of students. */
public class Grader implements GraderName   {
    private String name;   // grader's last name
    private Student[] st;  // students to grade
    private Solution sol;  // solution to assignment

    /** Constructor: Grader with last name name for
     *  students st, grading assignment with solution sol. */
    public Grader(String name, Student[] st, Solution sol) {
        this.name= name;
        this.st= st;
        this.sol= sol;
     }

    /** Return the last name of the grader. */
    public @Override String graderName() {
       return name;
    }
 }
```

(d) 6 points Below is a declaration of interface GraderName. Above, do whatever is necessary in class Grader to have it implement GraderName.

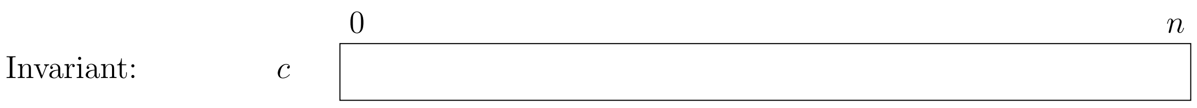
```
public interface GraderName {
    /** Return the last name of the grader. */
    String graderName();
}
```

6. Loop Invariants (10 points)

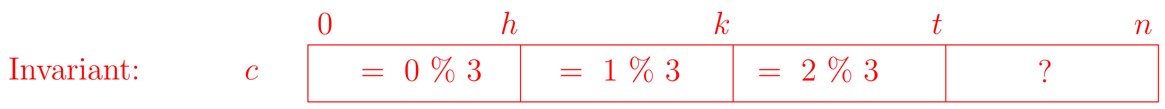
(a) 6 points Consider the following precondition, invariant, and postcondition for array c.



Complete the invariant below to generalize the above array diagrams. You will have to introduce a new variable. Place your variables carefully; ambiguous answers will be considered incorrect. Note: Several different invariants can be drawn; draw any one of them.



Sample Answer:



(b) 4 points Write down the four loopy questions to be answered in proving that a loop is correct.

1. How does it start —what initialization makes the invariant true?
2. When can it stop —what condition together with the invariant makes the result true?
3. How the repetend make progress toward termination?
4. How does the repetend keep the invariant true?