Prelim 1 Solution

CS 2110, March 15, 2016, 7:30 PM

	0	1	2	3	4	5	Total
Question	Name	True False	Short Answer	Object-Oriented	Recursion	Loop Invariants	
Max	1	20	14	25	19	21	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 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 its good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam.

(signature)

0. Name (1 point)

Ensure that your name and NetID is written on **every** page of this exam.

1. True / False (20 points)

Circle T or F in the table below.

a)	[✓ T]	F	The assignment `Object obj= new boolean[6];` is legal.
b)	[✓ T]	F	An interface can extend another interface.
c)	[✓ T]	F	The expression `’a’ * 1 == ’a’` evaluates to `true`.
d)	T	[✓ F]	Execution of the statements `String s1= "java"; String s2= s1.replace(’v’, ’V’);` makes `s1` reference the string `"jaVa"`.
e)	[✓ T]	F	A class can directly extend at most one superclass and can directly implement an arbitrary number of interfaces.
f)	T	[✓ F]	Let `b` be a non-null `int[]`. The expression `b[b.length]` always throws a `NullPointerException`.
g)	[✓ T]	F	Suppose class `Bar` implements interface `Foo`. The statement `Foo foo= new Bar();` is legal.
h)	T	[✓ F]	If a class does not have a constructor declared in it, an object of that class cannot be created.
i)	T	[✓ F]	`int[] x= new int[0];` causes an error at run time because we cannot instantiate (create) an array of length 0.
j)	T	[✓ F]	Suppose `String` variable `s` references the string `"abc"`. After execution of the expression `s.substring(1)`, `s` references the string `"bc"`.
k)	[✓ T]	F	The Java expression `Integer.valueOf(1234) == Integer.valueOf(1234)` can evaluate to `false`.
l)	[✓ T]	F	Abstract classes can have non-static fields.
m)	[✓ T]	F	Suppose `W` is an abstract class. The declaration `W v;` is legal.
n)	T	[✓ F]	If `b` and `c` are objects of class `A` and class `A` has not overridden `equals`, then `b.equals(c)` will evaluate to `true` if and only if the values of every field in `b` and `c` are equal.
o)	[✓ T]	F	Suppose class `B` extends `A`. Suppose `var` is a public field of class `A`. Suppose variable `x` is of type `B`. Then the expression `x.var` is legal.
p)	T	[✓ F]	Suppose `d` and `f` are variables of type `Object`. If `d` is `null`, then `d.equals(f)` will evaluate to `true` if and only if `f` is `null`.
q)	T	[✓ F]	The default value for a field of type `Double` is `0.0`.
r)	T	[✓ F]	The statement `throw Integer.valueOf(5);` is legal.
s)	T	[✓ F]	If class `Dog` extends class `Animal`, and variable `Animal pet` contains a pointer to a `Dog` object, then the statement `Dog spike= pet;` is legal.
t)	T	[✓ F]	If an **abstract** class implements an interface, it must provide an implementation for every method in that interface.

2. Short Answer (14 points)

(a) 8 points Below is a method `m`. Answer the following questions.

(i) (2 points) What is the result of `m("0")`? 22

(ii) (2 points) What is the result of `m("2")`?
Execution results in `java.lang.ArrayIndexOutOfBoundsException: -1`

(iii) (2 points) What is the result of `m("5.a")`? 0

(iv) (2 points) What is an example of an argument to `m` that would result in `-1`?
Any `String` containing an integer ≥ 6 or ≤ -2 is correct

```
public static int m(String numb) {
        String[] arrB = new String[]{"2", "4", "1", "99", "10", "125", "2"};
        int x= 0;
        try {
             x= Integer.parseInt(numb);
             String value = arrB[x+1];
             int target = Integer.parseInt(value);
             return (40 + target) / (2 - x);
        } catch (NumberFormatException e) {
             return 0;
        } catch (NullPointerException e) {
            return -3;
        } catch (ArithmeticException e) {
            String defaultEntry = arrB[-1];
            return Integer.parseInt(defaultEntry);
        } catch (ArrayIndexOutOfBoundsException e) {
            return -1;
        } catch (Exception e) {
            return -2;
        }
```

(b) 3 points Consider the following application:

```
public class Prelim {
    private int i;
    private static int j;

    public Prelim(int a, int b) {
        i = a;
        j = b;
    }

    public String toString() {
        return i + " " + j;
    }

    public static void main(String[]args) {
        Prelim p1 = new Prelim(0, 1);
        Prelim p2 = new Prelim(2, 3);
        System.out.println(p1 + " " + p2);
    }
}
```

What does running class `Prelim` as an application print to the console?

0 3 2 3

(c) 3 points Consider interfaces `I1` and `I2` and class `C` as defined below:

```
public interface I1 {
    void someGuess();
}
```

```
public interface I2 {
    void someGuess();
}
```

```
public class C implements I1, I2 {
    public void someGuess() {
        System.out.println("When in doubt, choose C");
    }
}
```

Are these definitions valid? Explain.

Yes. `C` can legally implement two interfaces and `C` provides a definition for all of the methods declared in the interfaces it implements

3. Object-Oriented Programming (25 points)

(a) 3 points You are hired by RolePlayGames Inc, and you need to design the next game taking place in a fantasy world. You are in charge of designing a class of wizards that are trained to perform ice spells. Your characters have duels with each to practice their skills.

Your first job is to design class `IceWizard`. Complete the body of method `launchSpell`, below. You do not need to assert preconditions.

Solution

```
public class IceWizard {
    private String name;     //name of the student. Is not null
    private int magicShield; //magic shield of the student. >= 0

    /** Constructor: instance with name (!= null) and shield (>= 0) */
    public IceWizard(int shield, String name) {
        magicShield= shield;
        this.name= name;
    }

    /** Return the student's name **/
    public String getName() { return name; }

    /** Return true iff this student has any magic shield remaining. **/
    public boolean isShielded() { return magicShield > 0; }

    /** If this student is shielded, launch an ice spell at opponent. To do that:
      * Reduce the opponent's shield by 20, but don't let it get below 0.
      * Precondition: opponent is not null. **/
    public void launchSpell(IceWizard opponent) {


        if (isShielded()) {
            opponent.magicShield= Math.max(0, opponent.magicShield - 20);
        }


    }
}
```

(b) 3 points A new teammate is working on a second type of Wizard, `FireWizards`. These wizards specialize in fire spells instead of ice spells. But these wizards' spells reduce magic shields by a factor of three rather than strictly decrease them. Here is your teammate's design for these students:

```
public class FireWizard {
    private String name; //Name of the student. Is not null
    private int magicShield; //magic shield of the student. >= 0

    /** Constructor: instance with name (!= null) and shield (>= 0) */
    public FireWizard(int shield, String name) {
        magicShield= shield;
        this.name= name;
    }

    /** Return the student's name **/
    public String getName() { return name; }

    /** Return true iff this student has any magic shield remaining. **/
    public boolean isShielded() { return magicShield > 0; }

    /** If this student is shielded, launch a fire spell at opponent. To do that:
      * Divide the opponent's shield by 3, rounding down.
      * Precondition: opponent is not null. **/
    public void launchSpell(FireWizard opponent) {


        if (isShielded()) {
            opponent.magicShield= opponent.magicShield / 3;
        }


    }
}
```

(c) 19 points Your boss just told you that, in the inter-disciplinary duel games, some `IceWizards` might have to duel against `FireWizards`. She has asked you to write a class `Duel` with a method `duel` that performs a duel and returns the name of the winner. But now you realize a limitation of you and your teammate's design: `IceWizards` can `launchSpells` only at other `IceWizards`, not at `FireWizards`, and vice versa.

Code up a new class `Wizard` below that addresses the design problem and mark up the earlier `IceWizard` and `FireWizard` classes with the changes necessary to take advantage of class `Wizard`. Then fill in the code for `duel` below using the new class so that `duel` works for both `IceWizards` and `FireWizards`. You can omit comments.

You will be graded on the quality of your design and the accuracy of your code. Any markup that appears to be intentionally ambiguous will be considered incorrect. Also, recall that keyword `protected` makes a member visible to all subclasses.

Solution

```java
public abstract class Wizard {
    private String name;
    protected int magicShield;

    public Wizard(int shield, String name) {
        magicShield = shield;
        this.name = name;
    }

    public String getName() { return name; }

    public boolean isShielded() { return magicShield > 0; }

    public abstract void launchSpell(Wizard opponent);
}

public class IceWizard extends Wizard {
    public IceWizard(int shield, String name) { super(shield, name); }

    public void launchSpell(Wizard opponent) {
        if (isShielded()) {
            opponent.magicShield = Math.max(0, opponent.magicShield - 20);
        }
    }
}

public class FireWizard extends Wizard {
    public FireWizard(int shield, String name) { super(shield, name); }

    public void launchSpell(Wizard opponent) {
        if (isShielded()) {
            opponent.magicShield = opponent.magicShield / 3;
        }
    }
}

public class Duel {
    /** Perform a duel between two wizards.
      * They take turns launching a spell at each other, starting with w1.
      * The duel stops when either w1 or w2 is no longer shielded.
      * Return the name of the student that is still shielded.
      * Precondition: w1 and w2 are not null and are shielded */
    public static String duel(Wizard w1, Wizard w2) {
        while (w1.isShielded() && w2.isShielded()) {
            w1.launchSpell(w2);
            w2.launchSpell(w1);
        }
        return w1.isShielded() ? w1.getName() : w2.getName();
    }
}
```

4. Recursion (19 Points)

(a) 10 points We want to compute the product of the values in a linked list `ll` using a call like `nodeProduct(ll.getHead())`. Complete function `nodeProduct` below according to its specification. Use recursion. **Do not use a loop.** You can assume every node has exactly one predecessor, except for the head which has none.

```
public class LinkedList {
    private Node head= null;
    public Node getHead() { return head; }
}
```

```
public class Node {
    public Node next;
    public int value;
}
```

Solution

```
/** Return the product of the values of node and its
  * successors. If node is null, return 1. */
public static int nodeProduct(Node node) {
    if (node == null) return 1;
    return node.value * nodeProduct(node.next);
}
```

(b) 9 points Write the body of function `neg` given below. **Use recursion, not a loop. Do not** change `n` into a `String` and then use `String` operations; use `int` operations `/` and `%`. You can use function `negDigit`, given below.

Solution

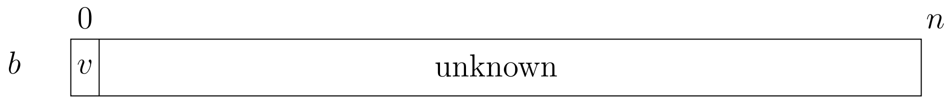
```
/**  Return the neg of n, as a String. The neg of n is defined as follows:
  * (1) a 0 in n's decimal representation is replaced by an underscore '_'.
  * (2) a digit d > 0 in n's decimal representation is replaced by 10-d.
  *  Precondition: n > 0   ---Make sure you see this
  *  Example: for n = 43, return  "67" and for n = 10203, return  "9_8_7" */
public static String neg(int n) {
    if (n < 10) return negDigit(n);
    return neg(n / 10) + negDigit(n % 10);
}
```

```
/** Return the neg of digit d, as defined in the spec of neg.
  * Precondition: 0 <= n < 10 */
public static String negDigit(int d) {
    return d == 0 ? "_" : "" + (10 - d);
}
```

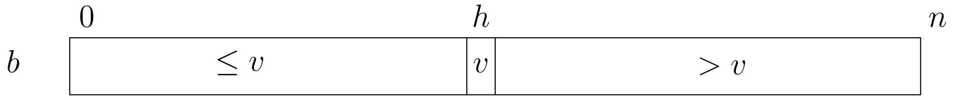
5. Loop Invariants (21 points)

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

Precondition:



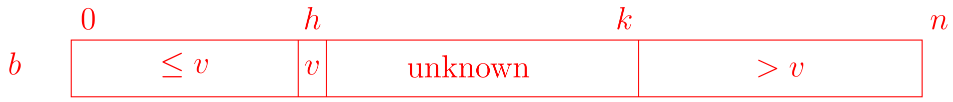
Postcondition:



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

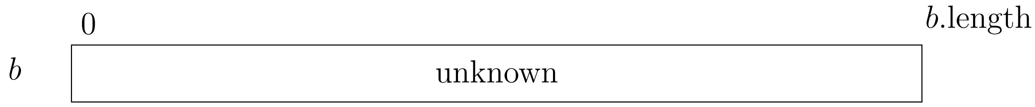
Solution

Invariant:

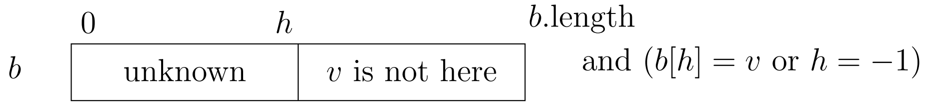


(b) 8 points Consider the following precondition, postcondition, and invariant.

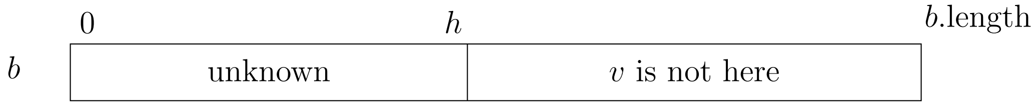
Precondition:



Postcondition:



Invariant:



Complete function `linearSearch` below according to its specification using the precondition, postcondition, and invariant above. You will be graded on how well you follow the invariant and use the four loopy questions.

Solution

```
/** Return the index of the last occurrence of v in b.
  * If v is not in b, return -1
  * Precondition: b is not null */
public static int linearSearch(int[] b, int v) {
    int h= b.length - 1;
    while (h != -1 && b[h] != v) {
        h= h - 1;
    }
    return h;
}
```

(c) 7 points Consider the following loop with initialization:

```
//Store the sum of m..n in z
//Precondition Q: m <= n
int z= 0;
int k= m;
//invariant P: z = sum of m..k and m <= k <= n
while (k <= n) {
    z= z + k;
    k= k + 1;
}
//Postcondition R: z = sum of m..n
```

Is the above loop invariant correct? If you think the loop invariant is correct, you must explain why *every* loopy question is satisfied. If you think the loop invariant is incorrect, you must pick *one* loopy question and explain why it is not satisfied.
The above loop invariant is **not** correct. The initialization does not make the invariant true, the opposite of the loop guard contradicts the invariant, and when $k = n$ the loop body does not keep the invariant true.