Prelim 1 Solution

CS 2110, October 1, 2015, 5:30 PM

	0	1	2	3	4	5	Total
Question	Name	True False	Short Answer	Testing	Strings	Recursion	
Max	1	20	36	16	15	12	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 9 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your booklet is 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.

0. Name 1 point

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

1. True / False (20 points)

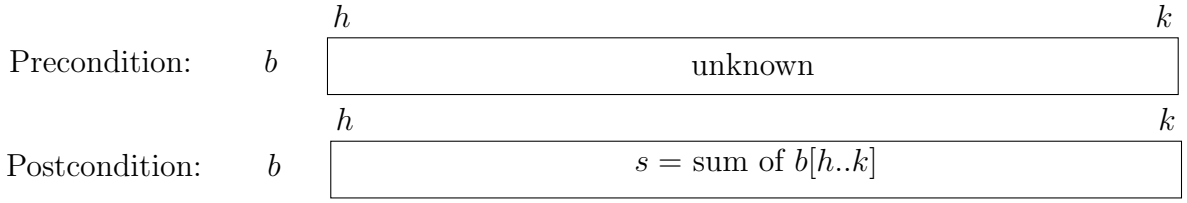
a)	T	F	If `a` is `null`, evaluation of `"" + a` results in a `NullPointerException`.
b)	T	F	Every Java class except `Object` extends **exactly** one other Java class.
c)	T	F	`boolean[]` is a primitive type. It is an array, and an array is an object.
d)	T	F	If class `A` defines a method with signature `public boolean equals(Object)`, we say that class `A` has **overloaded** method `equals`. This is overriding.
e)	T	F	If class `A` defines a method with the signature `public boolean equals(A)`, we say that class `A` has **overridden** method `equals`. This is overloading because the parameter type is different from that in method equals in class Object.
f)	T	F	Space for a local variable declared in an `if`-statement is allocated only if the `if`-condition is true. Space is allocated when the frame for the call is put on the call stack —before the method body is executed..
g)	T	F	If class `B` extends abstract class `A`, then the statement `A o= new B();` is illegal.
h)	T	F	Let `s` be a `String` variable. Suppose `s` is `null` and assertions are enabled. Then, execution of `assert s.length() > 0;` results in an assertion error (as opposed to some other error). It results in a NullPointerException.
i)	T	F	Execution of `C ob= new C(5);` stores the whole new object in `ob`. It stores a pointer to the object, something like C@2400fff0.
j)	T	F	The statement `List<String> b= new List<String>();` is illegal because interfaces cannot be instantiated.
k)	T	F	If class `C` declares a static method `m()` and a non-static field `name`, then `m()` cannot access `name` using the `this` keyword.
l)	T	F	If a local variable `f` in method `m()` has the same name as a field, and there is no setter method for field `f`, the field cannot be accessed within `m()`. Access the field using `this.f`.
m)	T	F	Execution of `Time[] t= new Time[3];` stores in `t` a pointer to an array that contains pointers to 3 newly created `Time` objects. The 3 array elements are set to `null`.
n)	T	F	If abstract class `A` contains abstract method `m()`, and class `B` extends `A` but does not implement method `m()`, then `B` must also be abstract.
o)	T	F	A doubly linked list allows us to access any value in the list in constant time. To access element no. k, start at the head and use the succ fields to march up to element k. In the worst case, takes time proportional to the list length.
p)	T	F	The only reason to make a class abstract is so that you can put abstract methods in it.
q)	T	F	If classes `B` and `C` both extend `A`, then the statement `C c = (C) new B();` is legal. An object of class B can be cast only to classes for which it has a partition: Object, A, and B
r)	T	F	Within a constructor, one can explicitly call a constructor in the superclass using the name of the superclass. Use `super(...);`
s)	T	F	Suppose `v` points at an object. Execution of `v= null;` deletes the object to which `v` points. All it does is store `null` in `v`.
t)	T	F	If class `Dog` extends class `Animal` and `a` is declared to be type `Animal`, then `Dog d= (Dog) a;` will compile but might result in an exception at runtime.

2. Short Answer (36 points)

(a) (4 points) Consider the following code segment, note that `(int)'a'` is 97, `(int)'b'` is 98, and `(char)99` is `'c'`. Execute the segment, writing each value printed to the right of the corresponding `println` statement.

```
char a = 'a';  \\SOLUTION. You must know: (1) + is evaluated left to right,
String b = "b"; \\SOLUTION. (2) if one op of + is a String, the other is
int c = 99;     \\SOLUTION.     converted to a String, and
                \\SOLUTION. (3) char is a number type and if necessary
                \\SOLUTION. it is converted to an int, its internal repr.
System.out.println(a + b + c); \\SOLUTION: ab99
System.out.println(a + b + (char) c); \\SOLUTION: abc
System.out.println(c + a + b); \\SOLUTION: 196b
System.out.println(b + a + c); \\SOLUTION: ba99
```

(b) (8 points) A program segment that is supposed to store into s the sum of values in $b[h..k]$ has the following precondition and postcondition:



A loop with initialization will be written for this problem, using this loop invariant: s is the sum of $b[i..k]$, as shown as the the diagram below.

Answer the following four questions, which gives you all the parts for the loop. Be extremely careful! Do not simply write the loop you are thinking of; use the invariant shown above.

(i) What initial assignment to s and i makes the invariant true?
$s = 0, i = k + 1$

(ii) What condition along with the invariant implies the postcondition?
$i == h$

(iii) In writing the body of the loop, how do you make progress toward termination?
$i = i - 1$

(iv) In writing the body of the loop, how do you keep the invariant true while making progress toward termination?
$s = s + b[i]$ // (put this after the progress statement)

(c) (6 points) Indicate which of the following three statements about linked lists are true. Assume you have direct access only to the head of a singly linked list or the head and tail of a doubly linked list. Circle **all** that are true.

(i) Finding the largest element in a sorted doubly linked list takes constant time.

(ii) Inserting a value in the middle of a doubly linked list takes constant time. false. You have to search for the middle element.

(iii) Inserting a value after the fourth element of a singly linked list takes time proportional to the length of the list. false. Constant time.

(d) (6 points) Consider class A, below. Write down the statements S1, S2, S3, S4, S5, and S6 that are executed, in the order they are executed, for the following procedure calls:

1. first(3); Statements executed: S2 S3 S4 S6 S1
2. first(0); Statements executed: S2 S5 S6 S1
3. first(6); Statements executed: S2 S3

```
public class A {
    public static void first(int i) {
            second(i);
            S1;
    }

    public static void second(int i) {
        S2;
        try {
            int b= 5/i;
            S3;
            if (i == 6) throw new Exception();
            S4;
        } catch (ArithmeticException e) {
            S5;
        }
        S6
    }
}
```

(e) (4 points) Write down the steps in executing a procedure call. You need to know this to understand method calls! Look at first recursion lecture, where we discussed it.

- Push a frame for the call on the stack
- Assign argument values to parameters
- Execute method body
- Pop frame from stack, leaving return value on the top of the stack

(f) (4 points) Consider the following classes.

```
class C {
    public String toString() {
        return "I'm a C";
    }
}
class B extends C {
    public String toString() {
        return "I'm a B";
    }
    public static void main(String[] args) {
        B b= new B();     // Suppose the value stored in b is B@12
        C c= b;
        Object o= b;
        System.out.println(b.toString());
        System.out.println(c.toString());
        System.out.println(o.toString());
    }
}
```

What is printed to the console when running class `B` as an application?There is only one object; the overriding toString() method is always called.
I'm a B
I'm a B
I'm a B

(g) (4 points) Consider the following class.

```
public class Suit {
    public static final Suit CLUBS= new Suit();
    public static final Suit SPADES= new Suit();
    public static final Suit HEARTS= new Suit();
    public static final Suit DIAMONDS= new Suit();
    private Suit() {}
}
```

Is there a better way to represent Suits in Java? If so, rewrite class `Suit` below in that way. If not, explain briefly why the above class is the best way to represent Suits.
Solution

```
public enum Suit {CLUBS, SPADES, HEARTS, DIAMONDS}
```

3. Testing (16 points)

(a) 8 points Consider the following method. (The median of a collection is the "middle" value if the collection has an odd number of elements or the arithmetic mean of two "middle" values otherwise.)

```
/** Return the median value in arr.
  * Precondition: arr is not null and arr.length >= 1   */
public static int median(int[] arr) {
    int n= arr.length / 2;
    // if arr has an odd number of elements
    if (n %2 != 0)
        return arr[n];
    // n has an even number of elements
    return(arr[n - 1] + arr[n]) / 2;
}
```

- This code has a serious bug. Write a test to illustrate the bug: **Solution**

```
@Test
public void testMedian() {
    assertEquals(2, median(new int[] {2, 1, 3}));
}
```

- Modify the precondition so the method meets its specification.
  Precondition: arr is not null, arr.length >= 1, and arr is sorted

(b) 8 points Consider the following code, which computes the value of the first argument raised to the value of the second.

```
/** Return a^n
  * Precondition: n >= 0  */
public static int power(int a, int n) {
    // if n is 0
    if (n == 0) return 1;
    // n > 0
    return power(a * a, n / 2);
}
```

- Unfortunately this code also has a serious bug. Write a test to illustrate the bug:
  Solution

  ```
  @Test
  public void testPower() {
      assertEquals(125, power(5, 3));
  }
  ```

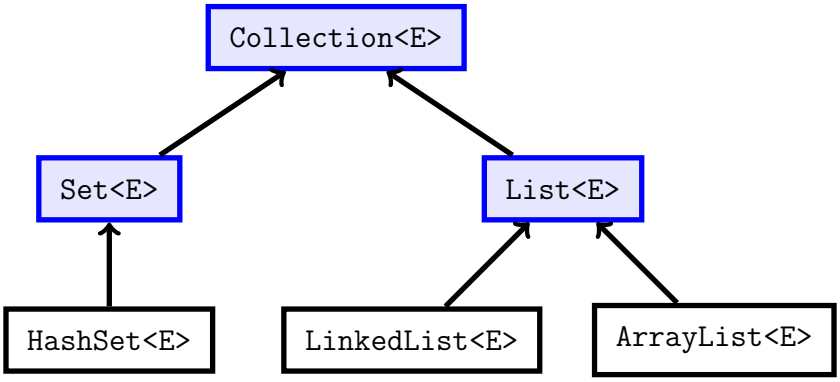
- Modify the **code** so the method meets its specification:

  Solution

  ```
  /** Return a^n.
    * Precondition: n >= 0.  */
  public static int power(int a, int n) {
      if (n == 0) return 1;
      // n > 0
      if (n % 2 == 0) return power(a * a, n / 2);
      return a * power(a * a, n / 2);
  }
  ```

Reference: Java Collections

Following is a reference to selected classes (unshaded) and interfaces (shaded) from the `Collections` library, as well as the most important methods defined in the interfaces. This information may be useful in completing the exercise on the opposite page.




```
interface Collection<E> {
  boolean add(E);
  void clear();
  boolean contains(Object);
  boolean equals(Object);
  boolean isEmpty();
  boolean remove(Object);
  int size();
  ...
}

interface Set<E> extends Collection<E> {
  ...
}

interface List<E> extends Collection<E> {
  boolean add(int, E);
  E get(int);
  E remove(int);
  E set(int, E);
  ...
}
```

To iterate over a collection, use a `for` loop:

```
Collection<E> c = ...;
for(E e : c) {
   ...
}
```

4. Strings and Collections (15 points)

(a) 7 points Suppose we have a method `areAnagrams` (you wrote such a method in A2):

```
/** Return true iff s and t are anagrams of each other. */
public static boolean areAnagrams(String s, String t) { ... }
```

Complete method `anagramIndices` below according to its specification.

Solution

```
/** Return a sorted list with the indices of Strings in l that are anagrams of s.
  * Precondition: l != null, s != null.  */
public static ArrayList<Integer> anagramIndices(ArrayList<String> l, String s) {
    ArrayList<Integer> b= new ArrayList<Integer>();
    for (int i= 0; i < l.size(); i= i+1) {
        if (areAnagrams(s, l.get(i))) {
            b(i);
        }
    }
    return b;
}
```

(b) 8 points Complete method `getChars` below according to its specification. Note that `Set` is an interface, so you will have to use an implementing class to construct the actual set.

Solution 1

```
/** Return a set of Characters in b, but exclude the blank character (space).
  * Example: getChars(['a', ' ', ' ', 'b', ' ', c, ' ']) is {'a', 'b', 'c'} */
public static Set<Character> getChars(char[] b) {
    HashSet<Character> chars= new HashSet<Character>();
    for (char c : b) {
        if (c != ' ') {
            chars.add(c);
        }
    }
    return chars;
}
```

5. Recursion (12 Points)

(a) 6 points Define define the fosterial function $f(n, k)$ as follows:

- $f(1, k) = f(0, k) = 1$
- If $n > 1$ and n is divisible by k, $f(n, k)$ is the product of $f(n-1, k)$, $f(n-2, k)$, and n.
- If $n > 1$ and n is not divisible by k, $f(n, k)$ is the product of $f(n-1, k)$, and n.

Complete function `fosterial` below according to its specification.

```
/** Return the fosterial value f(n,k).
  * Precondition: n >= 0, k > 0 */
public static int fosterial(int n, int k) {
    if (n <= 1)  return 1;
    if (n % k == 0)  return n * fosterial(n - 1, k) * fosterial(n - 2, k);
    return n * fosterial(n - 1, k);
}
```

(b) 6 points Complete the following function body. Do not change `n` into a `String` or other `char`-valued thing. Work only with type `int`.

Solution 1

```
/** Return true iff the decimal representation of n has a 5 in it.
  * Example hasFive(515253) is true, hasFive(12346) is false.
  * Precondition: n >= 0. */
public static boolean hasFive(int n) {
    if (n == 0)  return false;
    return n % 10 == 5 || hasFive(n / 10);
}
```

Solution 2

```
/** Return true iff the decimal representation of n has a 5 in it.
  * Example hasFive(515253) is true, hasFive(12346) is false.
  * Precondition: n >= 0. */
public static boolean hasFive(int n) {
    return n > 0 && (n % 10 == 5 || hasFive(n / 10));
}
```