CSE1030Z Exam
Saturday, April 20, 2013
2:00 PM–5:00 PM

This is a closed book test. No aids are permitted except for a non-electronic dictionary. Answer the questions in the spaces provided on the question sheets. You may use the back of the pages if you need more space for your answers.

Name and student number: _____

1. / 20

2. / 20

3. / 12

4. / 13

5. / 25

6. / 10

1. (20 total points)

   (a) (2 points) What are all of the meanings of the keyword `final` in Java?

   Solution:
   1. a variable can only be assigned to once
   2. a class cannot be extended
   3. a method cannot be overridden

   (b) (2 points) What is the most important thing to remember when working with static methods in an inheritance hierarchy?

   Solution:
   There is no dynamic dispatch on static methods.

   (c) (2 points) What is the main difference between a singleton and a multiton?

   Solution:
   A singleton is one instance that represents a single sate.
   A multiton has one instance for each unique state.

   (d) (2 points) What is the main difference between aggregation and composition?

   Solution:
   Composition implies ownership (or common lifetime), whereas aggregation does not.

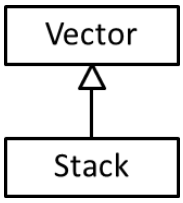
   (e) (2 points) What is meant by the term *event-driven programming*?

   Solution:
   The flow of program control is determined by events (and listeners of events).

(f) (2 points) What two conditions are required to ensure that a recursive method terminates?

Solution:
1. a base case is reached
2. the size of problem in each recursive invocation gets smaller

(g) (2 points) The UML class diagram for the `java.util.Stack` implementation of a stack is shown below; `Vector` implements the `List` interface so it has list methods such as `get`, `add`, and `remove`.



Explain why you agree or disagree with this implementation of a stack.

Solution:
I disagree with this implementation because a stack should not support all of the operations in List.

(h) (2 points) As an implementer, you can choose to implement a method so that it has preconditions on the parameters or you can validate the parameters. Explain how your choice (use preconditions or perform validation) affects the clients of your method.

Solution:
The client has no guarantees if you use preconditions.
The client has guarantees specified in the postcondition if you use validation.

(i) (2 points) What are the names of the two main operations supported by a queue data structure?

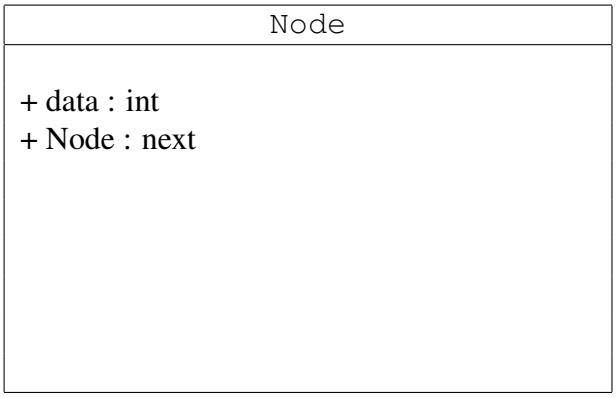
Solution:
1. enqueue
2. dequeue

(j) (2 points) What is a major difference between an interface and an abstract class?

Solution:
1. An interface has no attributes.
2. An interface cannot provide default implementations of its methods.

2. (20 total points) Consider the linked list data structure that we studied in CSE1030. Suppose that you have a linked list of `int` values.

   (a) (2 points) Fill in the attributes section of the UML class diagram for the nodes of the linked list.



   (b) (5 points) Complete the recursive algorithm `removeLast` that removes the last element from the linked list. You do not need to use Java; plain English will suffice, but you should refer to the attribute names you used in part (a) of this question. You can assume that the list has at least 2 elements (i.e., do not worry about the head of the list).

```
removeLast(Node n):
if (n.next.next == null) {
  n.next = null;
}
removeLast(n.next)
```

(c) (4 points) Prove that your algorithm in part (b) is correct.

Solution:
1. (prove base case is correct) If the node after n has no successor (n.next.next == null is true) then we know that n.next is the last node in the list; to remove it we can set n.next to null, which is exactly what the base case does.

2. (prove the recursive invocation is correct) Assume that removeLast(n) removes the last node in the list with head n. If n.next is not the last node in the list then we can treat it as the head node of a list and remove its last node. To do this, we should invoke removeLast(n.next), which is exactly what the recursive does.

(d) (4 points) Prove that your algorithm in part (b) terminates.

Solution:
1. (define the size of the problem) The size of the problem is m the number of elements in the list.

2. (show that each recursive invocation solves a smaller problem) The recursive invocation removes the last node from a list that does not include the current node n; this is a problem of size $m - 1 < m$.

(e) (1 point) State the recurrence relation that describes the running time of your algorithm in part (b). You can assume that checking the base case can be done in one unit of time (or constant time).

$T(n) = T(n-1) + 1$

(f) (4 points) Show how to solve the recurrence relation from part (d) to find the big-O running time of your algorithm. You may assume that $T(1) = 1$.

If you do not feel confident in your solution to part (c), then show how to solve the following recurrence relation instead:

$$\begin{aligned} T(n) &= T(n-2) + 2 \\ &= (T(n-2-2) + 2) + 2 \\ &= T(n-4) + 4 \\ &= (T(n-2-4) + 2) + 4 \\ &= T(n-6) + 6 \\ &= (T(n-6-2) + 2) + 6 \\ &= T(n-8) + 8 \\ &= T(n-k) + k \end{aligned}$$

$$T(1) = 1 \implies k = n-1$$

$$\begin{aligned} T(n) &= T(1) + n - 1 \\ &\in O(n) \end{aligned}$$

3. (12 total points)

(a) (7 points) Suppose that you have a `Stack` class that has only the following features:

- the elements are of type `int`
- a default constructor that creates an empty stack
- a method `isEmpty` that returns true if the stack is empty
- two methods `push` and `pop` that correspond to the two fundamental stack operations

Describe how you would write a (static) method that makes a copy of a stack. A postcondition of your method must be that the state of the stack `t` when the method finishes is the same as when the method started. Try to avoid using additional data structures (such as lists and arrays) if possible. Functional Java code is not required; for example, the first step of your method might be

1. make an empty stack named result.

```
public static Stack copy(Stack t)
```

1. make an empty stack named result
2. make an empty stack named tmp
3. while t is not empty
   (a) pop t
   (b) push the popped value onto tmp
4. while tmp is not empty
   (a) pop tmp
   (b) push the popped value onto result
   (c) push the popped value onto t
5. return result

(b) (5 points) Suppose that you have a `Queue` class that has only the following features:

- the elements are of type `int`
- a default constructor that creates an empty queue
- a method `size` that returns the number of elements in the queue
- two methods corresponding to the two fundamental queue operations

Describe how you would write a (static) method that makes a copy of a queue. A postcondition of your method must be that the state of the queue `q` when the method finishes is the same as when the method started. Try to avoid using additional data structures (such as lists and arrays) if possible. Functional Java code is not required; for example, the first step of your method might be

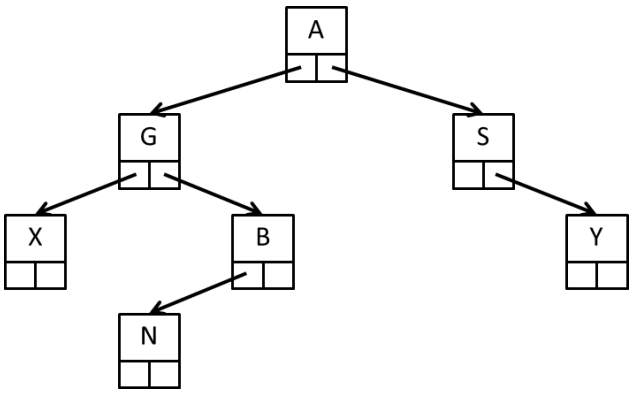
1. make an empty queue named result.

```
public static Queue copy(Queue q)
```

1. make an empty queue named result
2. n = q.size()
3. for (int i = 0; i $<$ n; i++)
   (a) value = dequeue q
   (b) enqueue value into result
   (c) enqueue value into q
4. return result

4. (13 total points)

(a) (3 points) What are the inorder, preorder, and postorder traversals of the following binary tree?



inorder : X, G, N, B, A, S, Y

preorder : A, G, X, B, N, S, Y

postorder : X, N, B, G, Y, S, A

(b) (4 points) Draw the binary search tree having integer valued elements created by inserting the elements in the following order: 50, 75, 17, 25, 50, 18, 30, 80.

(c) (4 points) Draw the binary search tree having integer valued elements created by inserting the elements in the following order: 80, 75, 50, 50, 30, 25, 17, 18.

(d) (2 points) What is the worst-case (big-O) running time when searching for an element in a binary search tree? Under what conditions does the worst-case running time occur?

Solution:
O(n)
This occurs when the elements are inserted in (almost) sorted order (ascending or descending).

5. (25 total points) In geometry, we can think of a square as being a rectangle with all sides having the same length. Suppose that you wish to implement the classes `Rectangle` and `Square` where `Square` is-a `Rectangle`.

   (a) (3 points) Suppose that every `Rectangle` has a `width` and a `height`. Every `Square` has a `width` and a `height` and a class invariant `width == height` is true. Suppose that `Rectangle` has only one method:

```
/*
 * Postcondition: Sets the width of the rectangle
 *                leaving the height unchanged
 */
public void setWidth(int newWidth) { ... }
```

   What problem do you have when you implement `Square`?

   Solution:

   Square cannot maintain its class invariant because the postcondition says that the width and height can be different.

   (b) (3 points) A fellow student says that you can fix the problem in `Square` by overriding `setWidth` and changing the postcondition of the `Square` version to:

```
/*
 * Postcondition: Sets the width of the square;
 *                also changes the height so that
 *                width == height is true
 */
public void setWidth(int newWidth) { ... }
```

   Explain why you agree or disagree with your fellow student.

   Solution:
   Disagree; Squares are no longer substitutable for Rectangles if you change the postcondition in this way.

   (This is a weakening of the postcondition from Rectangle.)

(c) (3 points) Another student says that you can fix the problem in `Square` by overriding `setWidth` and changing the postcondition of the `Rectangle` version to:

```
/*
 * Postcondition: Sets the width of the rectangle.
 */
public void setWidth(int newWidth) { ... }
```

Explain why you agree or disagree with your fellow student.

Solution:
Agree; this postcondition makes no promises about the height so Square could set its height when it sets its width.

Disagree; setWidth still seems like it should only change the width of the rectangle.

(The change is technically correct, but potentially confusing.)

(d) (4 points) Another student says that you can fix the problem in `Square` by adding a new method to `Square`:

```
/*
 * Postcondition: Sets the length of each side of the square.
 */
public void setLength(int newLength) { ... }
```

Explain why you agree or disagree with your fellow student.

Solution:
Disagree; Square inherits setWidth so the original problem is still present.

(e) (3 points) Another student says that you can fix the problem in `Square` by overriding `setWidth` so that it throws an exception:

```
/*
 * Postcondition: Operation not supported for squares;
 *                always throws an exception.
 */
public void setWidth(int newWidth) throws Exception { ... }
```

Explain why you agree or disagree with your fellow student.

Solution:
Disagree; the Rectangle version does not throw an unchecked exception, so the overridden version cannot throw one either.

(This change will not even compile.)

(f) (5 points) Another student says that you can fix the problem in `Square` by making `Rectangle` immutable and modifying the postcondition of the `Rectangle` version of `setWidth` to:

```
/*
 * Postcondition: Returns a new rectangle having width
 *                equal to newWidth and height equal to
 *                this rectangles height.
 */
public Rectangle setWidth(int newWidth) { ... }
```

Explain why you agree or disagree with your fellow student. Does `Square` need to override `setWidth` in this solution?

Solution:
Agree; now that the width cannot be changed there is nothing preventing Square from maintaining its class invariant.

Square does not need to override setWidth because there is nothing specific to Squares in the method.

(g) (4 points) Another student says that you can fix the problem in `Square` by inverting the inheritance relationship so that `Rectangle` is-a `Square`. Under this model, `Square` now defines the following method:

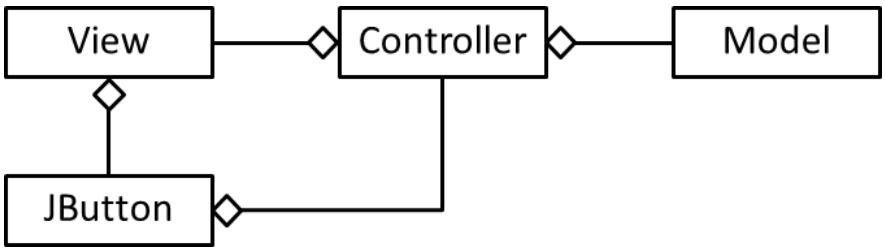
```
/*
 * Postcondition: Sets the width of the square.
 */
public void setWidth(int newWidth) { ... }
```

and `Rectangle` overrides it to set its width independently from its height.

Explain why you agree or disagree with your fellow student.

Solution:
Disagree; if Rectangle is substitutable for Square then it must maintain the class invariant of Square (width == height is true) which cannot be done if Rectangles can change their width and height independently.

6. (10 total points) Suppose that you have a GUI application where the user can interact only with one button. The UML class diagram for such an application is shown below:



(a) (3 points) Explain why the controller has a reference to the view.

Solution:
The controller needs to invoke view methods (to get and set the state of the view).

(b) (3 points) Explain why the controller has a reference to the model.

Solution:
The controller needs to invoke model methods (to get and set the state of the model).

(c) (1 point) Explain why the view has a reference to the button.

Solution:
The view constructs the button (because the button is part of the user interface or the view).

(d) (3 points) Explain why the button has a reference to the controller. Should this relationship be the other way around (i.e., should the controller have a reference to the button)?

Solution:
The button needs to invoke the actionPerformed method in the controller whenever an event is fired.

No, the relationship is correct because the button knows when it is pressed (it is the source of the event) and it needs to inform listeners when an event occurs.