University of Toronto
csc148S - Introduction to Computer Science, Spring 2001

Midterm Test Answers

Wednesday February 28, 2001

Duration: 50 minutes

Aids allowed: None

Family Name: ________________ Given names: ________________

Student #: ________________ Tutor: ________________

- There are 6 pages, including this one. The test is out of 50 marks and the value of each question is provided; please use this information to manage your time effectively.

Part A:	______	/ 15
Part B:	______	/ 10
Part C:	______	/ 25
Total	______	/ 50

Part A [15 marks in total]

Consider the following java classes with many details omitted.

```
class Fish extends FishTankItem implements Pet { ... }
class SunFish extends Fish { ... }
class OysterShell extends Shell { ... }
class Shell extends FishTankItem { ... }
class FishTankItem { ... }
class StarFish extends Fish { ... }
```

1. Draw an inheritance hierarchy diagram for these classes. Include the class `Object` in your diagram.

2. For each of the following java statements, indicate if it will compile by circling one of YES or NO.

   (a) `Fish f = new Shell();` — YES $\boxed{\text{NO}}$

   (b) `Fish f = new FishTankItem();` — YES $\boxed{\text{NO}}$

   (c) `FishTankItem ft = new OysterShell();` — $\boxed{\text{YES}}$ NO

3. Assume that you have the statement `Fish f = new SunFish();` earlier in your code. For each of these later casts, indicate by circling one of the options, whether the cast gives a compile-time error, a run-time error or runs fine.

   (a) `(Shell) f` — $\boxed{\text{COMPILE-TIME ERROR}}$ RUN-TIME ERROR RUNS FINE

   (B) `(SunFish) f` — COMPILE-TIME ERROR RUN-TIME ERROR $\boxed{\text{RUNS FINE}}$

   (C) `(FishTankItem) f` — COMPILE-TIME ERROR RUN-TIME ERROR $\boxed{\text{RUNS FINE}}$

4. `Pet.java` contains the following code:

```
public interface Pet {
    public void feed();
    public String getName();
}
```

For the code fragments listed below, indicate by circling one of the options, whether each gives a compile-time error, a run-time error or runs fine.

(a) COMPILE-TIME ERROR RUN-TIME ERROR $\boxed{\text{RUNS FINE}}$

```
SunFish sf = new SunFish();
sf.feed();
```

(b) $\boxed{\text{COMPILE-TIME ERROR}}$ RUN-TIME ERROR RUNS FINE

```
FishTankItem ft = new Fish();
ft.feed();
```

(c) $\boxed{\text{COMPILE-TIME ERROR}}$ RUN-TIME ERROR RUNS FINE

```
FishTankItem ft = new Shell();
ft.feed();
```

(d) COMPILE-TIME ERROR $\boxed{\text{RUN-TIME ERROR}}$ RUNS FINE

```
FishTankItem ft = new FishTankItem();
((Fish) ft).feed();
```

(e) COMPILE-TIME ERROR RUN-TIME ERROR $\boxed{\text{RUNS FINE}}$

```
Fish f = new StarFish();
f.feed();
```

5. We have stack on which we have called the following operations in this order: push(17), push(15), push (12), pop(), pop(), push(20), push(16), pop(), push(4)

Draw a sketch showing the current state of the stack.

```
4
20
17
```

Part B [10 marks]

You would like to design some java code for passing secret messages. Your messages will be coded in a very basic way where each character is represented by an integer. For example the char ‘H’ might be 0 and the character ‘E’ might be 3.

A class that handles coding must provide three operations. It should be able to take a code and character pair (for example ‘E’ and 3) and enter them into the list of codes. It should be able to return a code when given a character. And finally, it should be able to return a character when given a code. This class does not need to worry about breaking the message into individual char elements or reassembling the characters of a decoded message. That will be the job of the client.

You aren't going to actually write the code for the class, that's somebody else's job. Instead you need to write the java interface that their code must meet.
Below we have started a `Coder` interface. Finish it. Use exceptions to handle the situation where the client asks you to code or decode an element which hasn't been entered into the list yet. Make up appropriate exception class names. Include comments in your solution. JavaDoc is not required.

```
public interface Coder {
    // assign the code i to correspond to char 'c'
    // If 'c' already has a code, replace it with 'i'
    // If code 'i' is already used, throw an UnavailableCodeException();
    public void add(char c, int i) throws UnavailableCodeException;
```

Solution:

```
    // return the code for c
    // throws NoCodeException if c doesn't have a code yet
    public int encode(char c) throws NoCodeException();

    // return the char for code i
    // throws InvalidCodeException if i isn't a code
    public char decode(int i) throws InvalidCodeException();
```

Part C [25 marks]

Here is the file `MyQueue.java` and the class `LinkedNode`:

```
public interface MyQueue {
    // add o to the tail of the queue
    public void enqueue(Object o);

    // return the item at the head of the queue
    // and remove it from the queue
    // Pre: the queue is not empty
    public Object dequeue();

    // return the item at the head of the queue
    // but do not remove it.
    // Pre: the queue is not empty
    public Object head();

    // return true iff the queue is empty
    public boolean isEmpty();
}
```

```
public class LinkedNode {
    public LinkedNode(Object o) {
        data = o;
    }
    public Object data;
    public LinkedNode next;
}
```

Write the class `LinkedQueue` which implements `MyQueue` and uses `LinkedNode` objects to store the queue elements. Be neat. You do not need to copy comments from the interface class into your solution.

SOLUTION

```
public class LinkedQueue implements MyQueue {

    private LinkedNode head;

    // add o to the tail of the queue
    public void enqueue(Object o) {
        if (head == null) {
            head = new LinkedNode(o);
                } else {
            LinkedNode current = head;
            while (current.next != null) {
                current = current.next;
                        }
            current.next = new LinkedNode(o);
                }
        }

    // return the item at the head of the queue
    // and remove it from the queue
    // Pre: the queue is not empty
    public Object dequeue() {
        LinkedNode result = head.data;
        head = head.next;
        return result;
    }
```

```
    // return the item at the head of the queue
    // but do not remove it.
    // Pre: the queue is not empty
    public Object head() {
        return head.data;
    }

    // return true iff the queue is empty
    public boolean isEmpty() {
        return (head==null);
    }
}
```

End of Test