University of Toronto
CSC148 – Introduction to Computer Science Fall 2001

Mid Term Test – Section L5101

Duration: 50 minutes
Aids allowed: none

Make sure that your examination booklet has 8 pages (including this one). Write your answers in the spaces provided. Write legibly. You may use page 8 for rough work (tear it off, if you like). If you require more space to answer a question, write on the back of the previous page, and indicate in the answer space where you answer is.

Surname: ______________________

Given name(s): ______________________

Student #: ______________________

TA's name: (please circle below)

Irum Godil　　　Hank Tu　　　Wei Zhang

Please note that if you write the mid term in pencil, you will ***not*** be allowed to submit a remark request.

0.	/ 1
1.	/ 6
2.	/ 15
3.	/ 5
4.	/ 8
Total:	/ 35

Question 0: Administrative

[1 marks]

Write your surname and student number at the top of every page (except page 1) of this test.

Question 1: Design by Contract

[3 x 2 marks each – 6 marks total]

Below are methods with their comments. There is something wrong with each comment. Briefly explain what's wrong with each one (if there is more than one problem, pick the one you feel is most important).

```
/* Searches a queue for an object, v. Returns true if it contains
 * v, false otherwise
 */
static boolean search (Queue h, Object v);
```

Error:

`Searches which Queue?  Should say “Searches h for…”`

```
/* Loops through Vector d to find and return its middle object
 */
static Object middle(Vector d);
```

Error:

`Reference to “loop” give away internal details`

`Or`

`Does not address which object is the “middle” when there is an even number of objects`

`Or`

`Does not explain what happens when d is empty, or when d==null.`

```
/* Returns the length of the longest String in Queue q.
 * Pre: q is not null
 */
static int lengthOfLongest(Queue q);
```

Error:

`Does not explain what happens if q contains no Strings`

Question 2:

[10 marks]

On the last page of the exam, you will find the code for the `ArrayQueue` class and the `Stack` interface. You will be modifying the ArrayQueue class so that it also implements the `Stack` interface. The top of the stack will be the tail of the queue, so that when something is enqueued it is last in line to be dequeued, but first to be popped.

(a) First, change the class declaration to reflect the new implementation:
[2 marks]

`class ArrayQueue` implements Stack, Queue

(b) Next, implement each of the methods from the `Stack` interface. None of these methods should require that you add member variables to the class. Remember that all the member variables and methods from `ArrayQueue` are still present in the class you're adding these methods to. Marks will be awarded for correctness of your algorithm and for code reuse (that is, making sure you don't write code for something for which has already been written). Note that the space below may be far more than you need to complete this section.

Note that you **must** write good internal documentation for your methods.
[13 marks: 8 for implementation and 5 for comments]

```
// Add o to the end (the “top”) of the stack.
// Precondition: the stack must not be full.
void push(o) {

   // since the queue’s tail & stack’s top are at
   // same place, this is simple
   enqueue(o);










}
```

```
   // Remove and return the top element on the stack.
   // Precondition: the stack must not be empty
   Object pop() {

      // we need to maintain the representation invariant for
      // the queue here.
      return contents[size--];

   }

   //Returns true if the stack is empty; return false otherwise
   boolean isEmpty() {

      return size==0;

   }

   // Return true if the stack cannot hold any more items; return
   // false otherwise.
   boolean isFull() {

      // Assertion: the capacity of the queue is equal to contents.length
      return size==contents.length;

   }
}
```

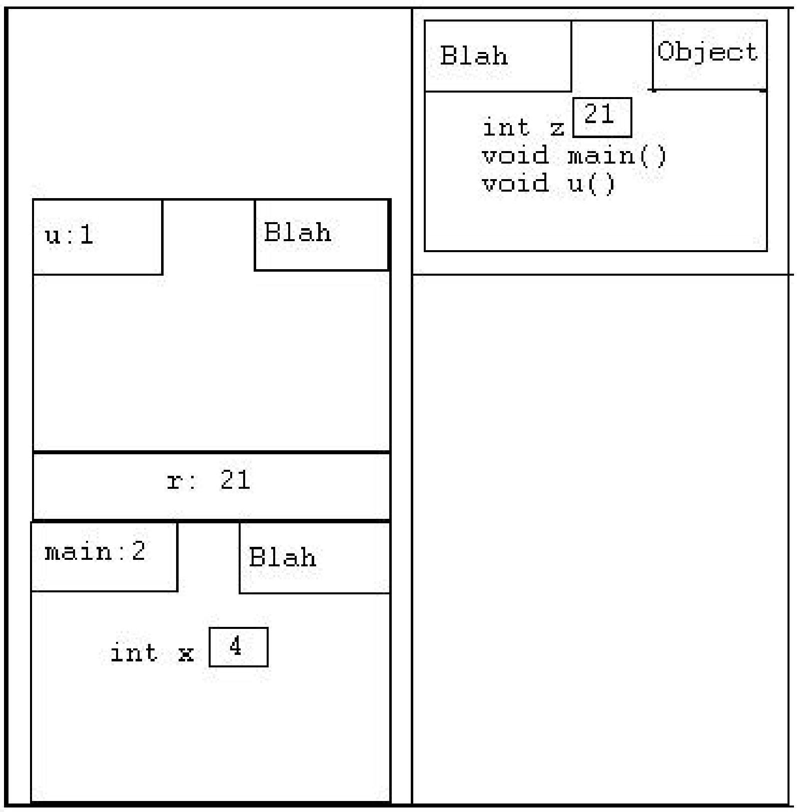
Question 3: Memory Model Tracing

[5 marks]

Here is a memory model (this leaves out the address & object for argv, but don't worry about that). This memory model is from running the `main()` method in class Blah.

Below are 4 possible classes `Blah`. Indicate which class matches this memory model by circling the letter above the class.

(the line numbering for method u() in the memory model may be slightly off – don't let this bother you)



A:

```
class Blah {
   static int z = 5;
   static int y = 17;
   public static void main(String[]
      argv) {

      int x = 4;
      u(21);
   }
   static void u (int r) { //no code }
}
```

B:

```
Class Blah {
   static int z = 5;
   int y = 17;
   public static void main(String[]
      argv) {

      int x = z – 1;
      u(x+y);
   }
   static void u(int r) { Blah.z = r; }
}
```

C: ✍ correct answer

```
class Blah {
   static int z = 3;

   public static void main(String[]
      argv) {

      int x = 4;
      u(25-x);
   }
   static void u(int r) { z = r; }
}
```

D:

```
class Blah {
   int z = 21;

   public static void main(String[]
      argv) {

      int x = 5;
      u(z);
   }
   static void u(int r) { // no code }
}
```

Question 4: Exceptions

[8 marks]

Below is a method that does something with an array:

```
// Sets the first n elements of array a to zero
public static void setZero(int[] a, int n) {
   for (int i=0; i<n; i++) {
      a[i] = 0;
   }
}
```

(a)

[4 marks]
The problem with this method is that n may be larger than the length of a. If this happens, the program will crash. Instead, it would be better if the method threw a specific exception if n were too large. Assume there is already an exception defined called `ArrayTooSmallException` that extends `Exception`. Use the space below to rewrite the `setZero` method so that it will throw the `ArrayTooSmallException` when the array is too small for the value n. Note that if n is too large, no elements of `a` should be set to zero.

```
public static void setZero(int[] a, int n) throws ArrayTooSmallException {

   if (n > a.length) {
      throw new ArrayTooSmallException;
   }
   for (int i=0; i<n; i++) {
      a[i] = 0;
   }
}
```

(b)

[4 marks]
Below is code that declares an array of ints and then asks the user repeatedly how many of the elements should be set to zero. The code is not yet complete. Complete the code for the `for` loop so that it will attempt to use the `setZero` method to set the first n slots in `theArray` to zero. Each time through the loop, your code should print nothing if it works correctly, or "failure!" if they entered a value for n that is too large. You may assume the user will input an int greater than or equal to zero.

```
public static void main(String[] argv) throws IOException {
   BufferedReader br = new BufferedReader(new
      InputStreamReader(System.in));

   int[] theArray = int[50];

   for(int j=0; j < 500; j++ ) {
      System.out.println("How many to set to zero?");
      int n = Integer.parseInt(br.readLine());

      try {
         setZero(theArray,n);
      } catch (ArrayTooSmallException) {
         System.out.println("failure!");
      }
   }
} // main
```

```
public class ArrayQueue implements Queue {
   // The number of elements in me
   private int size;

   // The elements in me, stored in contents[0..size-1]
   private Object[] contents;

   /* Representation Invariant:
    * If size is 0, I am empty, and size=0; otherwise:
    *    contents[0] is the head, contents[size-1] is the tail,
    *    contents[0..size-1] contains the Objects in the order they
    *    were inserted
    */

   // Append o to me.
   public void enqueue(Object o) {
      contents[size++] = o;
   }

   // Remove and return my front Object.
   // Precondition: I am not empty
   public Object dequeue() {
      Object head = contents[0];
      for (int i=0; i < size-1; i++) {
         contents[i]=contents[i+1];
      }
      size--;
      return head;
   }

   // Return my front Object.
   // Precondition: I am not empty.
   public Object head() { return contents[0]; }

   // Return the number of Objects in me.
   public Object size() { return size; }
}
```

```
interface Stack {
   // a First-In, Last Out data structure where the object most recently
   // pushed onto the stack is the first to be “popped” off.

   // Add o to the end (the “top”) of the stack.
   // Precondition: the stack must not be full.
   void push(o);

   // Remove and return the top element on the stack.
   // Precondition: the stack must not be empty
   Object pop();

   //Returns true if the stack is empty; return false otherwise
   boolean isEmpty();

   // Return true if the stack cannot hold any more items; return
   // false otherwise.
   boolean isFull();
}
```