Note: Hand a hard-copy of this quiz in before Wednesday’s lecture.

1 Exceptions

Consider the following code fragment:

```java
1 int[] array = new int[5];
2 int sum = 0;
3 for (int i = 0; i <= 5; i++) {
4    array[i] = i;
5    sum += i;
6 }
7 System.out.println(sum);
```

1. What does the code fragment above output when it is executed?
   (a) 10 (b) 15 (c) 21 (d) ArrayIndexOutOfBoundsException (e) NullPointerException

Consider the following code fragment:

```java
1 public class Person {
2    private String lastName;
3    public Person() {
4        this.lastName = "Sarlo";
5    }
6    public String getLastName() {
7        return lastName;
8    }
9 }
```

```java
1 public class Test {
2    private static Person p;
3    public static void main(String[] args) {
4        System.out.println(p.getLastName());
5    }
6 }
```

2. What does the code fragment above output when it is executed?
   (a) "Sarlo" (b) "Person@42gf32"
   (c) NullPointerException (d) ConstructorEmptyArgumentsException
Consider the following Java interface for the Stack ADT:

```java
public interface StackADT {
    public void push(T element);
    public T pop();
    public T peek();
}
```

Consider now the following code fragment:

```java
StackADT<String> s = new StackADT<String>();
StackADT<String> q = new StackADT<String>();
if (s.equals(q)) {
    System.out.println("equal");
} else {
    System.out.println("different");
}
```

3. What does the code fragment above output when it is executed?
(a) "equal"  (b) "different"  (c) Nothing. The above code has compilation errors.

Consider now the following code fragment:

```java
public class Foo {
    public static void change(int[] course) {
        try {
            course[5] = 10;
        } catch (NullPointerException e) {
            System.out.print("A");
        }
        System.out.print("B");
    }
    public static void main(String[] args) {
        int students[] = new int[5];
        try {
            change(students);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.print("C");
        }
    }
}
```

4. What does the code fragment above output when it is executed?
(a) "A"  (b) "AB"  (c) "B"  (d) "ABC"  (e) "BC"  (f) "AC"  (h) "C"
Consider now the following code fragment:

```java
1 public class Foo {
2    public static void change(int[] course) {
3        try { course[5] = 10; }
4        catch (ArrayIndexOutOfBoundsException e) { System.out.print("A"); }
5        finally { System.out.print("B"); }
6        System.out.print("C");
7    }
8    public static void main(String[] args) {
9        int students[] = new int[5];
10       try { change(students); }
11       catch (ArrayIndexOutOfBoundsException e) { System.out.print("D"); }
12    }
13 }
```

5. What does the code fragment above output when it is executed?
(a) "A" (b) "AB" (c) "B" (d) "ABC" (e) "BC" (f) "AC" (g) "BCD" (h) "ABCD" (i) "C" (j) "D" (k) "AD" (l) "BD"

Consider now the following code fragment:

```java
1 public class Main {
2    private static Person[] p;
3    public static void main(String[] args) {
4        p = new Person[2];
5        for (int i = 0; i < p.length; i++)
6            p[i] = new Person("A", "B", "C");
7        System.out.println(p[0].toString());
8    }
9 }
```

6. What is the value of 'p' after line 2?
(a) Something like "Person@42gf32" (b) null (c) A memory location

7. What is the value of 'p' after line 4?
(a) Something like "Person@42gf32" (b) null (c) A memory location

8. What is the value of 'p[0]' after line 4?
(a) Something like "Person@42gf32" (b) null (c) A memory location

9. What is the value of 'p[0]' after line 6?
(a) Something like "Person@42gf32" (b) null (c) A memory location

10. What is output during line 8? Assume that the Person class has no toString() method.
(a) Something like "Person@42gf32" (b) NullPointerException (c) "ABC"
Note: Some of these multiple choice questions have multiple correct answers.

2 Concepts

11. What is it called when you see \(< T >\) on the same line as the class name?
(a) Variable typing (b) Inheritance (c) Polymorphism (d) Generics

12. Where in the stack collection does the activity happen?
(a) Bottom of the stack (b) Middle of the stack (c) Top of the stack (d) Beside the stack

13. Where in the queue collection does the activity happen?
(a) Front of the queue (b) Middle of the queue (c) Rear of the queue (d) Beside queue

14. Where in the singly linked list collection does the activity happen?
(a) Front of the list (b) Middle of the list (c) Rear of the list
(d) Always at the front, but after that everywhere can have activity

15. What makes up a method signature?
(a) The number of parameters (b) The types of the parameters
(c) The order of the types of the parameters (d) The return type

16. What is method overloading?
(a) The Java compiler chooses a better method for you than what you tried to invoke
(b) A child class defines a method with the same method signature as a parent method
(c) An exception is generated and the method feels overloaded
(d) Two methods exist with the same name but different method signatures

17. What is method overriding?
(a) The Java compiler chooses a better method for you than what you tried to invoke
(b) A child class defines a method with the same method signature as a parent method
(c) An exception is generated and the method feels overloaded
(d) Two methods exist with the same name but different method signatures

18. Stacks are useful for changing the order of something in what way?
(a) Sorting (b) Reversing (c) Preserving (doesn’t change the order) (d) Randomizing

19. Queues are useful for changing the order of something in what way?
(a) Sorting (b) Reversing (c) Preserving (doesn’t change the order) (d) Randomizing

20. What can change between different implementations for the same collection?
(a) The result of the ADT’s operations
(b) The amount of work done by the implementation
(c) The amount of memory used by the implementation
21. Evaluate the postfix expression by drawing in Figure 1 and showing the contents of the stack after each input.

22. The answer would have been different if the stack was implemented using a linked list instead of an array.
   (a) True  (b) False

23. After evaluating a well-formed postfix expression, the stack will contain 1 element; this is the answer to the expression.
   (a) True  (b) False
Consider a stack implemented using a singly linked list where `top` and `count` are instance variables pointing to the first node of the list and giving the number of data items in the stack, respectively (see Figure 2). Consider now the following implementation of `push()`:

```java
private void push(T newValue) {
    LinearNode<T> newNode = new LinearNode<T>(newValue);
    newNode.setNext(top);
    top = newNode;
    count++;
}
```

24. Starting from the original example stack in Figure 2, draw the stack after performing `push(c)`, using the code fragment above showing the implementation of `push()` (include top and count).

Consider now the following code fragment:

```java
private void push2(T newValue) {
    LinearNode<T> newNode = new LinearNode<T>(newValue);
    newNode.setNext(top);
    top = top.getNext();
    count++;
}
```

25. Starting from the original example stack shown in Figure 2, draw the stack after performing `push2(c)`, using the code fragment above showing the implementation of `push2()` (include top and count).
4 Queues

Consider a queue implemented using a circular array where front, rear, and count are instance variables pointing to the front of the queue, the rear of the queue, and containing the number of elements in the queue, respectively (see Figure ?? for an example of such a queue). Consider now the following implementation of the enqueue() operation:

```java
1 public void enqueue(T element) {
2     if (size() == queue.length)
3         expandCapacity();
4     queue[rear] = element;
5     rear = (rear + 1) % queue.length;
6     count++;
7 }
```

26. Consider Figure 5. What sequence of operations (using enqueue() and dequeue()) could have already happened to result in this queue? Assume that the queue started empty and that we have only been adding letters in alphabetical order.

enqueue("A"), enqueue("B"), enqueue("C"), enqueue("D"), dequeue(), dequeue(), or something to that effect.

27. Starting from the original example queue shown in Figure 5 draw the queue after performing both enqueue("E"); and enqueue("F");:

28. At what point will enqueue() method call the expandCapacity() method?

When count == queue.size(), or when front == rear.
29. How can you use a queue to represent a repeating key when encoding a string?

Store the repeating key in a queue. When you want to shift a letter: call dequeue() and store the number, shift the letter the corresponding number of times, and enqueue() the number back to the queue.

30. Compare the original Caesar cipher to a cipher that uses a repeating key.

The original Caesar cipher shifts every letter by the same amount. Once you figure out the number, you can solve the cipher. Since a repeating key shifts each letter by a different amount, it is much harder to solve. For example, it is possible that the the same letter in the original string is encoded as different letters at different points in the message.

31. Use the repeating key \( \{3, 4, 1\} \) to encode the string “Hello World”. Show all of your work:

\[
\begin{align*}
'H' + 3 &= 'K', \ Q = \{4, 1, 3\} \\
'e' + 4 &= 'i', \ Q = \{1, 3, 4\} \\
'l' + 1 &= 'm', \ Q = \{3, 4, 1\} \\
'l' + 3 &= 'o', \ Q = \{4, 1, 3\} \\
'o' + 4 &= 's', \ Q = \{1, 3, 4\} \\
'W' + 1 &= 'X', \ Q = \{3, 4, 1\} \\
'o' + 3 &= 'r', \ Q = \{4, 1, 3\} \\
'r' + 4 &= 'v', \ Q = \{1, 3, 4\} \\
'l' + 1 &= 'm', \ Q = \{3, 4, 1\} \\
'd' + 3 &= 'g', \ Q = \{4, 1, 3\} \\
\end{align*}
\]

Hello World = Kimos Xrvmg
5 Linked Lists

32. Starting from the linked list shown in Figure 7, assume that we have created a new *LinearNode* called newNode holding the letter E, and we wish to add it between D and F. Trace through the *add()* method shown below to add E between D and F, filling in the table below as you go. Interpret the statement *element < front.getElement()* to be comparing alphabetical order.

```java
public void add(T element) {
    LinearNode<T> newNode = new LinearNode<>(element);
    LinearNode<T> current = front;
    LinearNode<T> previous = null;
    while ((current != null) && (current.getValue() < element)) {
        previous = current;
        current = current.getNext();
    }
    if (current == front) {
        newNode.setNext(front);
        front = newNode;
    } else {
        previous.setNext(newNode);
        newNode.setNext(current);
    }
}
```

```
<table>
<thead>
<tr>
<th>Location in code</th>
<th>current.getValue()</th>
<th>previous.getValue()</th>
<th>element</th>
<th>Enter loop?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 5 (first time)</td>
<td>“B”</td>
<td>null</td>
<td>“E”</td>
<td>Yes</td>
</tr>
<tr>
<td>Line 5 (second time)</td>
<td>“D”</td>
<td>“B”</td>
<td>“E”</td>
<td>Yes</td>
</tr>
<tr>
<td>Line 5 (third time)</td>
<td>“F”</td>
<td>“D”</td>
<td>“E”</td>
<td>No</td>
</tr>
</tbody>
</table>
```

33. Draw the linked list after E has been added.

```
front → B → D → F
```

Figure 8
Starting from the linked list shown in Figure 7, we wish to remove D from the middle of the list. Trace through the `remove()` method shown below to remove D, drawing the appropriate arrows in each diagram (see Figure 9) from front, current, previous, B, D, and F. Include the arrow representing null where appropriate.

```java
1 public T remove(T element) {
2    if (isEmpty()) then throw new EmptyCollectionException("Linked List");
3    LinearNode<T> current = front;
4    LinearNode<T> previous = null;
5    boolean found = false;
6    while (current != null && !found) {
7        if (element.equals(current.getElement())) found = true;
8        else {
9            previous = current;
10           current = current.getNext();
11        }
12    }
13    if (!found) then throw new ElementNotFoundException("Linked List");
14    if (size() == 1) then front = null;
15    else if (current.equals(front)) then front = current.getNext();
16    else previous.setNext(current.getNext());
17    return current.getElement();
18 }
```

Figure 9: Draw your answer on this diagram. (a) The linked list right before entering the while loop. (b) The linked list after 1 iteration of the loop. (c) The linked list after removing D.
35. Starting from the linked list shown in Figure 7 assume that we have created a new LinearNode called newNode holding the letter A, and we wish to add it to the front of the linked list. Trace through the addToFront() method shown below to add A at the front of the list, drawing the appropriate arrows in the diagram (see Figure 10) from front, newNode, A, B, D, and F. Include the arrow representing null where appropriate.

```java
public void addToFront(T element) {
    LinearNode<T> newNode = new LinearNode<>(element);
    if (front == null) {
        front = newNode;
    } else {
        front = newNode;
        newNode.setNext(front);
    }
}
```

![Diagram](image)

Figure 10
6 Algorithm Design

Write your algorithm in Java code or in detailed Java-like pseudocode like the one used in the lecture notes.

36. Consider an array $A$ storing $n > 1$ integers. Write an algorithm $\text{inverse}(A, n)$ that reverses the order of the integers stored in $A$. So, for example, if the array $A$ is as in the figure on the left, the modified array must be as in the figure on the right.

![Figure 11](image)

You can either use a stack or a queue in your solution, but not both; you need to decide which of these auxiliary data structures you need to use to answer this question. You cannot use any other additional data structures.

The only operations that you can perform on the stack are $\text{push}()$, $\text{pop}()$, $\text{peek}()$, and $\text{isEmpty}()$; you can write $\text{Stack auxStack = new Stack()}$ to create an empty stack. If you decide to use a queue the only operations that you can perform on the queue are $\text{enqueue}()$, $\text{dequeue}()$, $\text{first}()$, and $\text{isEmpty}()$; you can write $\text{Queue auxQueue = new Queue()}$ to create an empty queue.

If you use pseudocode, you must use sufficient detail. For example, you can write $A[i] = x$ or $s.push(A[i])$, but you cannot write statements like ”add x to the array A”, or ”remove element x from A”.

```java
1 public void inverse(int[] A, int n) {
  2   Stack auxStack = new Stack();
  3   for (int i = 0; i < A.length; i++)
  4     auxStack.push(A[i]);
  5   for (int i = 0; i < A.length; i++)
  6     A[i] = auxStack.pop();
  7 }

1 public void inverse(int[] A, int n) {
  2   Queue auxQueue = new Queue();
  3   for (int i = 0; i < A.length; i++)
  4     auxQueue.enqueue(A[i]);
  5   for (int i = A.length - 1; i >= 0; i--)
  6     A[i] = auxQueue.dequeue();
  7 }
```