

Each of the following questions has the same value. If there are several parts, the credit will be divided evenly among those parts. Remember that your grade is determined by what you write on this exam paper rather than on your "actual knowledge", so write carefully and accurately, think before writing, and proofread your work when you are finished.

1. Write Java code to initialize the elements of an array of integers so that the i^{th} element has the value $2*i+3$. Write your code below the following array declaration.

```
int [] array = new int[1000];
// initialize the elements of the array as instructed above
for (int i=0; i<array.length; i++) {
    array[i]=2*i+3;
}
```

2. Write Java code to remove the first element from a linked list whose header is the variable named `firstNode` and store the address of that deleted node in a reference variable named `deletedNode`. If the list is empty, you should store a null value in `deletedNode`. Assume that a node has two public fields named "data" (of type `int`) and "next" (of type `Node`).

```
Node deletedNode; // address of deleted node
Node firstNode; // the list header
// assume there is some code here that modifies the list in some way
// now, delete the first node (if there is one, consider all possibilities)
deletedNode=firstNode;
if(firstNode != null) {
    firstNode = firstNode.next;
}
```

3. Write Java code to append a new node (whose address is in the variable named `newNode`) to the end of the linked list whose header is the variable named `firstNode`. Assume that a node has two public fields named "data" (of type `int`) and "next" (of type `Node`).

```
Node firstNode; // the list header
// assume there is some code here that modifies the list in some way
Node newNode = new Node(100); // new node with data=100
// now, add the new node to the end of the list
if (firstNode != null) { // if the list is not empty
    Node pointer = firstNode; // start at head of list
    while(pointer.next != null) { //while "pointer" is not the last node
        pointer=pointer.next;
    } // "pointer" is now the last node in the original list
    pointer.next=newNode;
}
else { firstNode=newNode;}
```

4. Characterize the Java classes that implement the Serializable interface. What properties or capabilities do these classes have?

Instances of classes that implement the Serializable interface can be written to (and hence, read from) an output stream.

5. Characterize the Java Collections Framework classes that implement the List interface.

The Java Collections Framework classes that implement the List interface store items in a sequence and provide methods that give users complete control over the location of items in that sequence.

6. A Java class definition might contain *fields*, *constructors* and *methods*. Briefly describe the purpose of each of these components of a class definition.

Fields store values of variables used by instances of a class.

Constructors are invoked when an instance of a class is created, to properly initialize that instance.

Methods are functions or routines that contain executable code and are called to perform specific tasks.

7. Write a complete Java class definition for a node in a singly-linked list. Declare its fields to be public, so you won't need any accessor or mutator methods to access them.

```
public class Node {
    public Object data; // information stored in this node
    public Node next; // address of the next node in the list

    public Node(Object o) {
        data=o;
        next=null;
    }
}
```

8. Which Java API classes would you use if you had to read input data from a binary file? What would you need to know about the data in the binary file in order to use these classes and their methods correctly?

The `FileInputStream` class can be used to create an input stream connected to a particular file. Once the input stream has been created, a `DataInputStream` object can be created and connected to that file input stream. The `DataInputStream` object provides methods such as `readInt()`, `readDouble()`, `readChar()`, etc.

Of course, you have to know which of the above methods to use in order to read the file's contents correctly. This requires you to know the format of the data in the input file.

9. How are *try blocks* and *catch blocks* used in Java programs? When is it necessary to use them?

If you use a Java method that could throw one or more exception types AND if you want to handle one or more of those exception if they are thrown, then you must enclose the code that could throw the exception in a try block and code one or more catch blocks following the try block to handle the exception types you wish to handle.

10. What are the three important issues to address (checklist items) when designing or coding a recursive algorithm?

1. You must identify a base case which can be solved without making further recursive calls. Your recursive method must contain some logic that identifies this base case and solves it correctly.
2. Whenever a recursive call is made, it must be invoked for a simpler version of the problem than the version the calling routine is trying to solve. This successive simplification of the problem assures that the base case will eventually be reached.
3. Each invocation of a recursive method solves a version of the same problem. A method that makes a recursive call must know how to use the solution of one or more simpler instances of a problem to solve a more complex version of that same problem.

11. Code a **recursive** method named `sumSquares` with the signature below to compute the sum of the squares of the first `N` integers, where `N` is an input parameter for the method. Note that the return value is supposed to be a long integer.

```
public long sumSquares(int N) { // assume N >= 0
    if(N==0) { return 0L;}
    return N*N+sumSquares(N-1);
}
```

12. The recursive binary search method below is called from a main routine by a statement such as the following

```
int found = binarySearch(A, 0, A.length-1, x);
```

where A is a sorted array of integers and target is the value to be searched for. It is supposed to return the subscript of the position in the array where the target value is found (if it is in the array) and it is to return the value -1 if the target value is not found.

```
public static int binarySearch(int[] A, int low, int high, int target) {
    int mid=(low+high)/2;
    if (target==A[mid]) { // target found
        return mid;
    }
    else if (low > high) { // target can't be found
        return -1;
    }
    else if (target < A[mid]) { // search indices from low to mid
        return binarySearch(A, low, mid, target);
    }
    else { // search indices from mid to high
        return binarySearch(A, mid, high, target);
    }
}
```

Will the above recursive binary search method work correctly? Why or why not? If it doesn't work correctly, how can it be repaired?

This fails to work correctly if $high=1+low$ and the target value is larger than $A[mid]$, or if $high=low$ and the target is not the value in $A[low]$. In these cases $mid=low$ because integer division always rounds downward, so the recursive call fails to simplify the problem instance when the recursive call is made.

The method can be repaired by making changing the first recursive call to `binarySearch(A, low, mid-1, target)` and changing the second recursive call to `binarySearch(A, mid+1, high, target)`

13. The standard usage of the word *stack* is as a reference to an abstract data type with certain properties or capabilities. Code a Java interface that describes this abstract data type.

```
public interface stack {
    public void push(Object o); // add an item to the stack
    public Object pop();       // remove and return the most recently added item
    public Object peek();      // return (without removing) most recently added item
}
```

14. The Java language gives programmers two techniques for creating programs that can execute concurrently. One way is to define a class that extends the Thread class. Another way is to define a class that implements the Runnable interface. Which of these two methods is preferred? Why is it preferred?

Implementing the Runnable interface is the preferred technique. Because Java doesn't support multiple inheritance, a Java class can extend at most one other class. On the other hand, a Java class can implement any number of interfaces, so you don't sacrifice any versatility by your decision to implement the Runnable interface.

The amount of coding work you need to do is the same. You have to code a run() method that defines the functionality of your concurrent method.

15. Suppose an algorithm has time complexity $T(N)=O(N^2)$ when solving a problem of size N.

If $T(10)=2$ minutes (i.e. a problem of size $N=10$ can be solved in 2 minutes), how large a problem would you expect to be able to solve in 1 hour? An approximate answer is OK, or you can leave your answer in the form of an arithmetical expression.

The time to solve a problem of size N is proportional to N^2 . Thus, $T(N)=c \cdot N^2$ or, equivalently, $T(N)/N^2$ is a constant.

Solving $2/(10^2)=60/(N^2)$ yields $N^2 = 3000$ and N is the square root of 3000 which is approximately 54.