

✓ The .....method in Java is used to in the retrieve or fetch the top element of the Stack

1/1

- ☐ poll()
- ☐ element()
- ☒ peek()
- ☐ remove()



✓ Given `List<String> list = new ArrayList<>()`. Which of the following is VALID?1/1

- ☒ `list.add("1");`
- ☐ `list.add(new Integer(1));`
- ☐ `list.add(1);`
- ☐ `list.add(new ArrayList());`



✗ Which of the following statements is TRUE about binary search trees?

0/1

- ☒ The value of a parent is larger than its left child but is smaller than its right child.
- ☐ The value of a parent is smaller than its left child but is larger than its right child.
- ☐ The inorder traversal of a binary search tree visits the nodes in sorted order.
- ☐ The value of a parent is larger than both children.



Correct answer

- ☒ The inorder traversal of a binary search tree visits the nodes in sorted order.
- ☒ The value of a parent is larger than its left child but is smaller than its right child.

✓ Which of the following statements declares a class named AA with a generic type E? 1/1

☐ class <E> AA {

☒ class AA<E> { ✓

☐ class E<AA> {

☐ class <AA>E {

✓ Which of the following statements is FALSE about implementing a stack or queue using a list? 1/1

☐ Using a list to implement a stack or queue allows for dynamic resizing, as the list can grow or shrink based on the number of elements added or removed.

☒ Queue is best implemented using composition while Stack is best using inheritance. ✓

☐ Composition is better than inheritance because composition does not inherit unnecessary methods from super class.

☐ A list provides the operations of adding elements at the end (push) and removing elements from the end (pop), which align with the Last-In-First-Out (LIFO) principle of a stack.

✓ Which of the following statements is FALSE about recursive loop ?

1/1

- ☐ Recursive loops typically consist of a base case, which defines the condition when the loop should terminate, and a recursive case, which calls the loop again with a modified input or state.
- ☐ A recursive loop is a loop construct that calls itself repeatedly until a specific condition is met.
- ☐ Recursive loops should be used with caution as they can consume a significant amount of memory and processing power if not implemented properly
- ☒ A recursive method can run just fine without any base case. ✓

✓ In the implementation of ArrayList, which of the following are TRUE?  
Choose TWO.

1/1

- ☐ capacity is reduced by 1 if an element is deleted from the list.
- ☐ capacity is always greater than size.
- ☒ size is reduced by 1 if an element is deleted from the list. ✓
- ☐ size is the length of the array used to store the elements in the list.
- ☒ size indicates the number of elements in the list. ✓

✗ Which of the following statements are TRUE?

0/1

☒ The removal order of the elements in a Queue is first-in-first-out. ✓

☐ Duplication of elements is not allowed in a PriorityQueue.

☒ A Queue orders its elements according to their natural ordering using the Comparable interface if no Comparator is specified. ✗

☒ A Queue orders its elements according to the Comparator if a Comparator is specified in the constructor. ✗

Correct answer

☒ The removal order of the elements in a Queue is first-in-first-out.

✓ Analyze the following code.

1/1

```
import java.util.*;

public class Question{
    public static void main(String[] args)
    {
        TreeSet<String> tSet= new TreeSet<>();

        tSet.add("Queen");
        tSet.add("Jack");
        tSet.add("King");

        System.out.println(tSet.last());
    }
}
```

- ☐ The program displays King
- ☒ The program displays Queen ✓
- ☐ The program cannot compile, because the last() method is not defined in Set.
- ☐ The program may display Queen, Jack, King
- ☐ The program displays Jack

✗ What is the output of the following code?

0/1

```
public class TestLinkedHashSet
{
    public static void main(String[] args)
    {

        Set<String> set = new LinkedHashSet<>();

        set.add("Paris");
        set.add("New York");
        set.add("Beijing");
        set.add("New York");

        System.out.println(set);
    }
}
```

- ☒ ['Paris', 'New York', 'Beijing'] ✗
- ☐ [Paris, New York, Beijing, New York]
- ☐ ["Paris", "New York", "Beijing"]
- ☐ [Paris, New York, Beijing]

Correct answer

- ☒ [Paris, New York, Beijing]

✓ What is the output of the following program?

1/1

\*Tips: poll() is a function to remove an element in queue.

```
Compile Undo Cut Copy Paste Find... Close Source Code
1 import java.util.LinkedList;
2 import java.util.Queue;
3
4 public class QueueExample {
5     public static void main(String[] args) {
6         Queue<Integer> queue = new LinkedList<>();
7
8         queue.add(2);
9         queue.add(4);
10        queue.add(6);
11
12        if (queue.size() >= 2) {
13            LinkedList<Integer> tempList = new LinkedList<>(queue);
14            tempList.add(2, 7);
15            queue = new LinkedList<>(tempList);
16        }
17
18        System.out.println(queue);
19
20        queue.poll(); // similar to dequeue in queue
21        queue.poll();
22
23    }
24 }
```

☒ [2, 4, 7, 6]



☐ [2, 4, 6, 2, 7]

☐ [2, 4, 6, 7]

☐ [7, 6]

☐ [2, 4]

✗ The generic type E of class BB must be compared with other variable in the 0/1 class. How should you declare class BB?

- ☒ class BB <E> implements Comparable<E> { ✗
- ☐ class BB<E implements Comparable<E>> {
- ☐ class BB <E extends Comparable<E>> {
- ☐ class BB <E> extends Comparable<E> {
- ☐ class BB <E> {

Correct answer

- ☒ class BB <E extends Comparable<E>> {



- ✗ Implement the following found method which finds whether an element exists in a binary search tree. The method returns true if the element exists, returns false otherwise. Select correct sequence of line numbers that represents a correct implementation. Not all lines are used. 0/1

```
public boolean found(E e){
```

```
1  TreeNode<E> current = root;  
2  return false;  
3  return true;  
4  while (current != null)  
5      if (e.compareTo(current.element) == 0)  
6          if (e == current.element)  
7              else if (e < current.element)  
8                  else if (e > current.element)  
9                      if (e.compareTo(current.element) < 0)  
10                     else  
11                         current = current.right;  
12                         current = current.left;  
  
}
```

☐ 1, 4, 5, 2, 9, 11, 10, 12, 2

☐ 1, 4, 5, 3, 9, 12, 10, 11, 2

☒ 1, 4, 6, 3, 9, 12, 10, 11, 2



☐ 1, 4, 5, 2, 8, 12, 10, 11, 2

☐ Other: .....

Correct answer

☒ 1, 4, 5, 3, 9, 12, 10, 11, 2

✓ For the following recursive method, what is the base case(exit case)?

1/1

```
public static void Recursive(int n)
{
    if (n>=1) {
        System.out.print(n);
        Recursive(n/2);
    }
}
```

☐ n > 1

☐ n <=1

☒ n < 1



☐ no base case

✓ What is the output of the following program?

1/1

```
import java.util.HashMap;

public class HashMapExample {
    public static void main(String[] args) {

        HashMap<String, Integer> map = new HashMap<>();

        map.put("apple", 10);
        map.put("banana", 5);
        map.put("orange", 8);
        map.put("grape", 15);
        map.put("kiwi", 3);
        map.put("banana", 10);

        System.out.println(map.get("banana"));
    }
}
```

- ☐ 5
- ☐ 8
- ☐ 15
- ☒ 10
- ☐ 3

