**Q1**

public class Question1 {

public static void main(String[] args) {

System.out.println(new Question1().fibonacci(10));

}

private int fibonacci(int n) {

if (n == 1)

return n;

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: The termination condition is incorrectly coded, causing infinite recursion. The proper base case is missing.

**Mark:** 1 mark for error, 1 mark for correct explanation

**Q2**

public class Question2 {

public static void main(String[] args) {

System.out.println(new Question2().factorial(5));

}

public int factorial(int n) {

if (n == 0)

return 0;

else

return n \* factorial(n - 1);

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation** : Error Line: 7. Incorrect base case return value; factorial(0) should return 1, not 0.

**Mark:** 1 mark for error, 1 mark for correct explanation

**Q3:**

public class Question3 {

private int[] cache;

public static void main(String[] args) {

Question3 q = new Question3();

q.cache = new int[11];

System.out.println(q.cachedFibonacci(10));

}

public int cachedFibonacci(int n) {

if (n <= 1)

return n;

if (cache[n] != 0)

return cache[n];

cache[n] = cachedFibonacci(n - 1) + cachedFibonacci(n - 2);

return cache[n];

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: There is no error, base case and caching mechanism are working correctly. The expected output is 55.

**Mark:** 1 mark for no error, 1 mark for correct explanation

**Q4:**

public class Question4 {

public static void main(String[] args) {

System.out.println(new Question4().sumArray(new int[] {1, 2, 3, 4, 5}, 0));

}

public int sumArray(int[] arr, int index) {

if (index >= arr.length)

return 0;

return arr[index] + sumArray(arr, index);

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: **Error Line: 10**. Index not incremented, causing infinite recursion. Should be sumArray(arr, index + 1).

**Mark:** 1 mark for error, 1 mark for correct explanation

**Q5:**

public class Question5 {

private static int[] cache;

public static void main(String[] args) {

cache = new int[5]; // Subtle Error: cache size is too small

System.out.println(new Question5().cachedFactorial(5));

}

public int cachedFactorial(int n) {

if (n == 0)

return 1;

if (cache[n] == 0)

cache[n] = n \* cachedFactorial(n - 1);

return cache[n];

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: The cache array size is too small. Since we are calculating the factorial of 5, the cache needs to be able to store the results for factorial(0) through factorial(5). The array should be initialized with size 6 to avoid an **ArrayIndexOutOfBoundsException** when storing cache[5].

**Mark:** 1 mark for error, 1 mark for correct explanation

**Q6:**

public class Question6 {

public static void main(String[] args) {

System.out.println(new Question6().calculate(10));

}

public int calculate(int n) {

if (n < 0)

return 0;

if (n == 0)

return 1;

return calculate(n) + calculate(n - 1); // Error: recursive call with the same value

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: **Error Line: 11**. Incorrect recursive call with calculate(n) causing infinite recursion. Should be calculate(n - 1) only.

**Mark:** 1 mark for error, 1 mark for correct explanation

**Q7:**

public class Question7 {

public static void main(String[] args) {

System.out.println(new Question7().recursivePrint("hello"));

}

public String recursivePrint(String str) {

if (str.isEmpty())

return "";

System.out.println(str);

return recursivePrint(str);

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: **Error Line: 10**. Recursive call with the same string causes infinite recursion. Should be recursive Print(str.substring(1)).

**Mark:** 1 mark for error, 1 mark for correct explanation

**Q8:**

**public class Question8 {**

**public static void main(String[] args) {**

**int[] numbers = {1, 2, 3, 4, 5};**

**System.out.println(new Question8().recursiveSum(numbers, 0));**

**}**

**public int recursiveSum(int[] numbers, int index) {**

**if (index == numbers.length - 1)**

**return numbers[index];**

**return numbers[index] + recursiveSum(numbers, index+1);**

**}**

**}**

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: There is no error in this code, the index is incremented properly; the expected output is 15.

**Mark:** 1 mark for no error, 1 mark for correct explanation

**Q9:**

class Node {

int value;

Node left;

Node right;

public Node(int value) {

this.value = value;

this.left = null;

this.right = null;

}

public Node(int value, Node left, Node right) {

this.value = value;

this.left = left;

this.right = right;

}

}

public class Question9 {

public static void main(String[] args) {

// Creating a sample binary tree:

// 1

// / \

// 2 3

Node root = new Node(1, new Node(2), new Node(3));

System.out.println("Sum of tree values: " + new Question9().traverseTree (root));

}

// Traverse the binary tree and sum the values of all nodes

public int traverseTree(Node node) {

if (node == null)

return 0;

return node.value + traverseTree (node.left);

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

**Explanation**: Only traverses left nodes, ignoring the right subtree, leading to incomplete tree traversal.

**Mark:** 1 mark for error, 1 mark for correct explanation

**Q10:**

public class Question10 {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5};

System.out.println(new Question10().recursiveSum(numbers, 0));

}

public int recursiveSum(int[] numbers, int index) {

if (index == numbers.length - 1)

return numbers[index];

return numbers[index] + recursiveSum(numbers, index);

}

}

**Question**: Review the program provided above and identify any errors within it. If you find no errors, state the expected output of the program. Explain your answer.

***Explanation***: **Error Line: 11**. Index is not incremented, leading to infinite recursion. The correct call should be recursiveSum(numbers, index + 1).

**Mark:** 1 mark for error, 1 mark for correct explanation