Coding Questions for a Senior Java Developer Interview

1. Reverse a String Without Using Built-in Functions

```
public class ReverseString {
    public static String reverse(String str) {
        char[] chars = str.toCharArray();
        int left = 0, right = chars.length - 1;
        while (left < right) {
            char temp = chars[left];
            chars[left] = chars[right];
            chars[right] = temp;
            left++;
            right--;
        }
        return new String(chars);
    }

    public static void main(String[] args) {
        System.out.println(reverse("Java Developer"));
    }
}
```

2. First Non-Repeating Character

```
import java.util.LinkedHashMap;
import java.util.Map;

public class FirstNonRepeating {
    public static char firstUniqueChar(String str) {
        Map<Character, Integer> charCount = new LinkedHashMap<>();
        for (char ch : str.toCharArray()) {
            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);
        }
        for (Map.Entry<Character, Integer> entry : charCount.entrySet()) {
            if (entry.getValue() == 1) return entry.getKey();
        }
        return '_';
    }

    public static void main(String[] args) {
        System.out.println(firstUniqueChar("aabbcdeff"));
    }
}
```

3. Find Missing Number in Array

```
public class MissingNumber {
    public static int findMissing(int[] nums) {
```

```
        int n = nums.length;
        int xor = n;
        for (int i = 0; i < n; i++) {
            xor ^= i ^ nums[i];
        }
        return xor;
    }

    public static void main(String[] args) {
        int[] nums = {3, 0, 1};
        System.out.println(findMissing(nums));
    }
}
```

4. Detect Loop in Linked List

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; next = null; }
}

public class DetectLoop {
    public static boolean hasCycle(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) return true;
        }
        return false;
    }
}
```

5. Thread-safe Singleton

```
public class Singleton {
    private static volatile Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
```

```
        }
        return instance;
    }
}
```

6. Print Numbers in Order Using Threads

```
import java.util.concurrent.Semaphore;

public class PrintInOrder {
    private static final Semaphore sem1 = new Semaphore(1);
    private static final Semaphore sem2 = new Semaphore(0);
    private static final Semaphore sem3 = new Semaphore(0);

    static class Task implements Runnable {
        private final int number;
        private final Semaphore current;
        private final Semaphore next;

        public Task(int number, Semaphore current, Semaphore next) {
            this.number = number;
            this.current = current;
            this.next = next;
        }

        public void run() {
            while (true) {
                try {
                    current.acquire();
                    System.out.print(number + " ");
                    Thread.sleep(500);
                    next.release();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        }
    }

    public static void main(String[] args) {
        new Thread(new Task(1, sem1, sem2)).start();
        new Thread(new Task(2, sem2, sem3)).start();
        new Thread(new Task(3, sem3, sem1)).start();
    }
}
```