IT Skill Test GIC Myanmar

Duration: 30 Minutes Total Questions: 8

---

1. You are debugging a Java application and encounter the following code snippet:

```
public void processData(List<Integer> numbers) {
    for (int i = 0; i < numbers.size(); i++) {
        if (numbers.get(i) % 2 == 0) {
            numbers.remove(i);
        }
    }
}
```

The method is supposed to remove all even numbers from the list. However, it's not working as expected. What is the issue and how can it be fixed?

A.
```
for (int i = numbers.size() - 1; i >= 0; i--) {
        if (numbers.get(i) % 2 == 0) {
            numbers.remove(i);
        }
    }
```

B.
```
for (Iterator<Integer> iter = numbers.iterator(); iter.hasNext();) {
        if (iter.next() % 2 == 0) {
            iter.remove();
        }
    }
```

C.
```
for (int i = 0; i < numbers.size(); i++) {
        if (numbers.get(i) % 2 == 0) {
            numbers.remove(i);
            i--;
        }
```

```
    }
```

```
D.numbers.removeIf(n -> n % 2 == 0);
```

2. What is the output of the following Java code?

```
public class Test {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "Hello";
        String str3 = new String("Hello");

        System.out.println(str1 == str2);
        System.out.println(str1 == str3);
        System.out.println(str1.equals(str3));
    }
}
```

A.true
  false
  true

B.true
  true
  true

C.false
  false
  true

D.true
  false
  false

3. You are reviewing a colleague's code and come across the following method:

```
public static int findMax(int[] numbers) {
    int max = numbers[0];
    for (int i = 1; i < numbers.length; i++) {
        if (numbers[i] > max) {
            max = numbers[i];
        }
    }
    return max;
}
```

What potential issue should you point out?

A.The method doesn't handle negative numbers correctly

B.The loop should start from index 0, not 1

C.The method doesn't handle an empty array

D.The method is inefficient and should use Arrays.sort() instead

4. You're working on a Java application that needs to read a large text file line by line. Which of the following code snippets is the most efficient and follows best practices for resource management?

A.
```
BufferedReader reader = new BufferedReader(new FileReader("file.txt"));
String line;
while ((line = reader.readLine()) != null) {
    // process line
}
reader.close();
```

B.
```
try (BufferedReader reader = new BufferedReader(new FileReader("file.txt")))
{
    String line;
    while ((line = reader.readLine()) != null) {
        // process line
    }
```

```
}
```

C.
```
Scanner scanner = new Scanner(new File("file.txt"));
while (scanner.hasNextLine()) {
    String line = scanner.nextLine();
    // process line
}
scanner.close();
```

D.
```
List<String> lines = Files.readAllLines(Paths.get("file.txt"));
for (String line : lines) {
    // process line
}
```

5. You are tasked with implementing a method that checks if a given string is a palindrome (reads the same backwards as forwards). Which of the following implementations is the most efficient?

A.
```
public boolean isPalindrome(String s) {
    return s.equals(new StringBuilder(s).reverse().toString());
}
```

B.
```
public boolean isPalindrome(String s) {
    for (int i = 0; i < s.length() / 2; i++) {
        if (s.charAt(i) != s.charAt(s.length() - 1 - i)) {
            return false;
        }
    }
    return true;
}
```

C.
```
public boolean isPalindrome(String s) {
    return s.equals(new StringBuffer(s).reverse().toString());
}
```

D.
```
public boolean isPalindrome(String s) {
    return Stream.of(s.split(""))
        .equals(Stream.of(s.split("")).sorted(Comparator.reverseOrder()));
}
```

6. You are debugging a Java application that processes user input. The following method is throwing a NullPointerException, but you're not sure why. Identify the line causing the issue and the correct fix:

```
public void processInput(String input) {
    String[] parts = input.split(",");
    for (int i = 0; i <= parts.length; i++) {
        System.out.println(parts[i].trim());
    }
}
```

A.for (int i = 0; i < parts.length; i++) {

B.String[] parts = input.split(",", -1);

C.if (input != null) { String[] parts = input.split(","); }

D.System.out.println(parts[i] != null ? parts[i].trim() : "");

7. You're working on a legacy system that uses a custom logging mechanism. The following method is supposed to log messages with timestamps, but it's not working as expected. What's the issue and how would you fix it?

```
public class Logger {
    private static final String DATE_FORMAT = "yyyy-MM-dd HH:mm:ss";
    private SimpleDateFormat formatter = new
SimpleDateFormat(DATE_FORMAT);

    public void log(String message) {
        String timestamp = formatter.format(new Date());
        System.out.println(timestamp + ": " + message);
    }
}

// Usage in a multi-threaded environment
Logger logger = new Logger();
ExecutorService executor = Executors.newFixedThreadPool(10);
for (int i = 0; i < 100; i++) {
```

```
    executor.submit(() -> logger.log("Test message"));
}
```

A. Change 'private SimpleDateFormat formatter' to 'private static SimpleDateFormat formatter'

B. Synchronize the log method: 'public synchronized void log(String message)'

C. Use ThreadLocal<SimpleDateFormat> instead of a single SimpleDateFormat instance

D. Replace SimpleDateFormat with DateTimeFormatter: 'private final DateTimeFormatter formatter = DateTimeFormatter.ofPattern(DATE_FORMAT);'

8. You're reviewing code and come across the following method. What potential issue do you see, and how would you improve it?

```
public static List<Integer> findDuplicates(List<Integer> numbers) {
    List<Integer> duplicates = new ArrayList<>();
    for (int i = 0; i < numbers.size(); i++) {
        for (int j = i + 1; j < numbers.size(); j++) {
            if (numbers.get(i).equals(numbers.get(j))) {
                duplicates.add(numbers.get(i));
            }
        }
    }
    return duplicates;
}
```

A. Use a Set<Integer> instead of List<Integer> for duplicates to avoid duplicate entries

B. Change the inner loop to start from 0 instead of i + 1

C. Use Collections.frequency(numbers, numbers.get(i)) > 1 to check for duplicates

D. Replace the entire method with numbers.stream().filter(i -> Collections.frequency(numbers, i) > 1).distinct().collect(Collectors.toList())