Junior Java Developer Skills Assessment

Good morning, team. Welcome to your comprehensive skills evaluation. The purpose of today is not just to test what you know, but to challenge you to apply your knowledge to solve complex problems, much like you will in a real-world development environment. The tasks are designed to be difficult and to make you think deeply about code design, efficiency, and best practices.

The Rules:

1. **No Collaboration:** This is an individual assessment. You are not to discuss the problems or your solutions with other trainees.
2. **Use of Resources:** You may use the official Java documentation and any notes you have taken during the course. You are discouraged from searching for direct solutions to the problems online. The goal is to assess *your* problem-solving ability.
3. **Plagiarism:** Any form of plagiarism will result in an immediate and serious review of your position in this training program. We have tools to detect code plagiarism, and we take this very seriously. All code you submit must be your own.
4. **Time Management:** The day is long and the tasks are substantial. Read through the entire evaluation at the beginning to get a sense of the scope. Allocate your time wisely. There are no extensions.
5. **Submission:** All work must be submitted by the end of the day. Detailed submission instructions are at the end of this document.

A Note on "Humility":

This evaluation is intentionally challenging. It is designed to expose the gaps between knowing a concept and mastering it. Do not be discouraged if you find it difficult. The goal is to identify areas for growth. A humble developer is one who recognizes what they don't yet know and is eager to learn.

Duration: 6 Hours
Scenario: You're applying for a Junior Java Developer position at CodeStart Company

Welcome Message

"We want to see how well you can apply the Java basics you've learned. This assessment focuses on fundamental skills that every Java developer needs in their daily work. Take your time and show us your understanding of the core concepts."

Task 1: Setup Your Workspace (30 Minutes)

"Every developer needs to set up their environment properly."

What you need to do:

1. **Create a Git repository** called `java-skills-[yourname]`
2. **Set up basic folder structure:** `java-skills-yourname/├── src/├── README.md└── .gitignore`
3. **Create a simple compile script** that compiles your Java files
4. **Write a README.md** explaining what this project is about

Example of what we expect:

- Your `.gitignore` should ignore `.class` files and IDE folders
- Your compile script should use `javac` command
- Your README should have your name and project description

Task 2: Basic Java Programming (1.5 Hours)

"Let's see your fundamental Java skills."

Task 2.1: User Information System (45 minutes)

Create a program that handles basic user information:

```
public class UserInfo {
    // Create these variables with proper access modifiers
    // - name (String)
    // - age (int)
    // - email (String)
    // - isActive (boolean)

    // Create constructor that takes all parameters

    // Create getter and setter methods for all variables

    // Create a method that displays user info nicely formatted
    public void displayInfo() {
        // Print user information in a nice format
    }

    // Main method for testing
    public static void main(String[] args) {
        // Create 3 different users
        // Display their information
        // Test your getters and setters
```

```
    }
}
```

Task 2.2: Employee Payroll System (45 minutes)

Create a payroll calculator that handles different employee types and complex pay rules:

```
public class PayrollCalculator {

    // Method to calculate weekly pay based on employee type and hours
    public static double calculateWeeklyPay(String employeeType, double
hoursWorked, double hourlyRate) {
        // Employee types: "FULL_TIME", "PART_TIME", "CONTRACTOR", "INTERN"
        // Use switch case for different rules:
        // FULL_TIME: Regular pay for 40 hours, overtime (1.5x) for hours >
40
        // PART_TIME: Regular pay, no overtime, max 25 hours
        // CONTRACTOR: Flat rate, no overtime rules
        // INTERN: 20% discount from hourly rate, max 20 hours
        // Handle invalid employee types and negative values
    }

    // Method to calculate tax deduction based on pay brackets
    public static double calculateTaxDeduction(double grossPay, boolean
hasHealthInsurance) {
        // Tax brackets using nested if-else:
        // $0-500: 10% tax
        // $501-1000: 15% tax
        // $1001-2000: 20% tax
        // Above $2000: 25% tax
        // If hasHealthInsurance is true, reduce tax by $50
        // Return total tax amount
    }

    // Method to process multiple employees and find statistics
    public static void processPayroll(String[] employeeTypes, double[] hours,
double[] rates, String[] names) {
        // Calculate pay for each employee
        // Find: highest paid employee, lowest paid employee, average pay
        // Count how many employees worked overtime (>40 hours)
        // Display results in a formatted table
        // Handle arrays of different lengths gracefully
    }

    public static void main(String[] args) {
        // Test data:
        String[] types = {"FULL_TIME", "PART_TIME", "CONTRACTOR", "INTERN",
"FULL_TIME"};
        double[] hours = {45, 20, 35, 15, 50};
        double[] rates = {25.0, 18.0, 40.0, 12.0, 30.0};
        String[] names = {"Alice", "Bob", "Charlie", "Diana", "Eve"};

        // Test individual calculations first
        // Then process the entire payroll
        // Show all results clearly
```

```
    }
}
```

Task 3: Working with Strings and Arrays (1.5 Hours)

"String and array manipulation is used everywhere in real applications."

Task 3.1: Student Grade Manager (60 minutes)

Create a system to manage student grades:

```
public class GradeManager {

    // Method to reverse student names in an array
    public static String[] reverseStudentNames(String[] names) {
        // Reverse each individual name (like "John" becomes "nhoJ")
        // Keep the array order the same
        // Return the modified array
    }

    // Method to calculate letter grades
    public static char getLetterGrade(int score) {
        // Use if-else conditions:
        // 90-100: 'A', 80-89: 'B', 70-79: 'C', 60-69: 'D', below 60: 'F'
    }

    // Method to find students who need to retake exam
    public static String[] findFailingStudents(String[] names, int[] scores)
{
        // Return array of names where score is below 60
        // Use loops to check each student
    }

    public static void main(String[] args) {
        String[] students = {"Alice", "Bob", "Charlie", "Diana"};
        int[] scores = {95, 67, 45, 78};

        // Test all your methods
        // Display results clearly
    }
}
```

Task 3.2: Text Processor (30 minutes)

Create a program that processes text input:

```
public class TextProcessor {

    // Count words in a sentence
    public static int countWords(String sentence) {
        // Split the sentence and count words
```

```
        // Handle empty strings
    }

    // Replace specific words
    public static String replaceWord(String text, String oldWord, String
newWord) {
        // Replace all occurrences of oldWord with newWord
    }

    public static void main(String[] args) {
        // Test with: "Java is fun and Java is powerful"
        // Count words
        // Replace "Java" with "Programming"
    }
}
```

Task 4: Object-Oriented Programming (2 Hours)

"Now let's see your OOP skills with a practical example."

Task 4.1: Library Book System (90 minutes)

Create a simple library management system:

```
// Base class
public class Book {
    // Protected variables (so child classes can access them)
    protected String title;
    protected String author;
    protected int pages;
    protected boolean isAvailable;

    // Constructor
    public Book(String title, String author, int pages) {
        // Initialize all variables
        // Set isAvailable to true by default
    }

    // Basic methods
    public void borrowBook() {
        if (isAvailable) {
            isAvailable = false;
            System.out.println(title + " has been borrowed.");
        } else {
            System.out.println(title + " is not available.");
        }
    }

    public void returnBook() {
        isAvailable = true;
        System.out.println(title + " has been returned.");
    }
```

```
    public void displayInfo() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Pages: " + pages);
        System.out.println("Available: " + isAvailable);
    }

    // Getters (create all of them)
    public String getTitle() { return title; }
    // ... create the rest
}

// Child class
public class Textbook extends Book {
    private String subject;
    private int edition;

    // Constructor that calls parent constructor
    public Textbook(String title, String author, int pages, String subject,
int edition) {
        // Call parent constructor using super()
        // Initialize subject and edition
    }

    // Override the displayInfo method
    @Override
    public void displayInfo() {
        // Call parent's displayInfo using super.displayInfo()
        // Then add subject and edition information
    }

    // Add getter methods for subject and edition
}

// Main class to test everything
public class Library {
    public static void main(String[] args) {
        // Create 2 regular books
        // Create 1 textbook
        // Test borrowing and returning
        // Display information for all books
    }
}
```

Task 4.2: Static vs Non-Static Practice (30 minutes)

Create a simple counter system:

```
public class VisitorCounter {
    // Static variable to count total visitors
    private static int totalVisitors = 0;

    // Non-static variable for individual session
    private int sessionVisits;
    private String visitorName;
```

```
    // Constructor
    public VisitorCounter(String name) {
        this.visitorName = name;
        this.sessionVisits = 0;
        totalVisitors++; // Increment total when new visitor is created
    }

    // Non-static method
    public void recordVisit() {
        sessionVisits++;
        System.out.println(visitorName + " visited. Session visits: " +
sessionVisits);
    }

    // Static method
    public static void displayTotalVisitors() {
        System.out.println("Total visitors today: " + totalVisitors);
    }

    // Static method to get total (getter)
    public static int getTotalVisitors() {
        return totalVisitors;
    }

    public static void main(String[] args) {
        // Create 3 different visitors
        // Have each visitor record some visits
        // Display total visitors using static method
    }
}
```

Task 5: Problem Solving with Loops and Conditions (1.5 Hours)

"These are the kinds of problems you'll solve in real programming jobs."

Task 5.1: Simple Business Logic (60 minutes)

Create a shopping discount calculator:

```
public class ShoppingCart {

    // Calculate total price with discounts
    public static double calculateTotal(double[] prices, String customerType)
{
        double total = 0;

        // First, calculate sum of all prices using a loop

        // Then apply discount based on customer type using switch:
        // "REGULAR": no discount
```

```
        // "PREMIUM": 10% discount
        // "VIP": 20% discount

        return total;
    }

    // Find most expensive item
    public static double findMostExpensive(double[] prices) {
        // Use loop to find highest price
        // Handle empty array
    }

    // Count items above a certain price
    public static int countExpensiveItems(double[] prices, double threshold)
{
        // Count how many items cost more than threshold
    }

    public static void main(String[] args) {
        double[] cart = {25.99, 45.50, 12.99, 89.99, 15.75};

        // Test with different customer types
        // Find most expensive item
        // Count items over $30

        System.out.println("Regular customer total: $" + calculateTotal(cart,
"REGULAR"));
        System.out.println("Premium customer total: $" + calculateTotal(cart,
"PREMIUM"));
        System.out.println("VIP customer total: $" + calculateTotal(cart,
"VIP"));
    }
}
```

Task 5.2: Pattern Problems (30 minutes)

Create some simple pattern generators:

```
public class PatternMaker {

    // Print a number triangle
    public static void printNumberTriangle(int rows) {
        // Print pattern like:
        // 1
        // 1 2
        // 1 2 3
        // 1 2 3 4
        // (for rows = 4)
    }

    // Print multiplication table
    public static void printMultiplicationTable(int number, int limit) {
        // Print: number x 1 = result, number x 2 = result, etc.
        // Up to the limit
    }
```

```
    public static void main(String[] args) {
        printNumberTriangle(5);
        System.out.println();
        printMultiplicationTable(7, 10);
    }
}
```

Task 6: Git and Documentation (30 Minutes)

"Professional developers use Git every day."

What you need to do:

1. **Create branches for your work:**
   - `feature/basic-programming` for Tasks 2
   - `feature/arrays-strings` for Task 3
   - `feature/oop-practice` for Task 4
   - `feature/problem-solving` for Task 5
2. **Make commits with good messages:**
   - `"Add UserInfo class with basic properties"`
   - `"Complete SimpleCalculator with switch cases"`
   - `"Implement Book class with inheritance"`
3. **Write a summary document** (`SOLUTIONS.md`) explaining:
   - What you learned while doing each task
   - Which parts were challenging
   - How you tested your code
   - One thing you want to improve

What We're Looking For

Code Quality (40 points):

- Does your code compile and run without errors?
- Are your variable names clear and meaningful?
- Did you use proper Java naming conventions?
- Are your methods organized and easy to understand?

Concept Understanding (40 points):

- **Variables & Types**: Proper use of int, String, boolean, arrays
- **Control Structures**: Correct use of if-else, switch, loops
- **OOP Basics**: Classes, objects, inheritance, method overriding
- **Static vs Non-Static**: Understanding when to use each

- **Access Modifiers**: Proper use of public, private, protected

Problem Solving (20 points):

- Can you break down problems into smaller steps?
- Do you handle edge cases (empty arrays, null values)?
- Are your solutions logical and efficient?

Success Levels:

Ready for Junior Role (80-100%):

- All code works correctly
- Shows good understanding of Java basics
- Clean, readable code
- Proper Git usage

Getting There (60-79%):

- Most code works with minor issues
- Understands core concepts but some confusion
- Code is mostly readable
- Basic Git usage

Needs More Practice (40-59%):

- Some code works but has errors
- Shows partial understanding
- Code is hard to follow
- Minimal Git usage

Keep Learning (<40%):

- Significant errors in most code
- Limited understanding of concepts
- Poor code organization
- No proper Git workflow

Tips for Success:

1. **Start simple** - Get basic functionality working first
2. **Test as you go** - Run your code frequently to catch errors early
3. **Read error messages** - They usually tell you what's wrong
4. **Use meaningful names** - `studentName` is better than `s`
5. **Add comments** - Explain tricky parts of your code

6. **Don't panic** - Take breaks when you're stuck

Submission Checklist:

- [ ] All Java files compile without errors
- [ ] Each class has a main method for testing
- [ ] Git repository has proper branch structure
- [ ] SOLUTIONS.md document completed
- [ ] Code uses proper naming conventions
- [ ] All requirements attempted (even if not perfect)

Remember: This is about showing your understanding of Java fundamentals. We want to see how you think and solve problems, not just perfect code. Good luck!