1. **Palindrome Check:**
   * Write a Java program to check if a given string is a palindrome.
2. **Fibonacci Series:**
   * Write a Java program to print the first 10 numbers of the Fibonacci series.
3. **Prime Number:**
   * Write a Java program to check if a given number is prime.

**Object-Oriented Programming**

1. **Class and Object:**
   * Create a Person class with attributes name and age. Write a method to display the details of the person.
2. **Inheritance:**
   * Create a class Employee with attributes name and salary. Inherit it to create a class Manager with an additional attribute department. Write methods to display the details of both classes.
3. **Polymorphism:**
   * Demonstrate method overloading and method overriding in Java.

**Collections**

1. **ArrayList:**
   * Write a Java program to create an ArrayList of strings, add 5 names to it, and then iterate through the list to display each name.
2. **HashMap:**
   * Write a Java program to create a HashMap where keys are employee IDs and values are employee names. Add some entries and display the key-value pairs.
3. **Sorting:**
   * Write a Java program to sort an ArrayList of integers in ascending order.

**Exception Handling**

1. **Try-Catch:**
   * Write a Java program to demonstrate exception handling using try-catch blocks. Include scenarios for ArithmeticException and NullPointerException.
2. **Custom Exception:**
   * Create a custom exception class InvalidAgeException. Write a program that throws this exception if the age provided is less than 18.

**File Handling**

1. **Read File:**
   * Write a Java program to read the content of a text file and display it on the console.
2. **Write File:**
   * Write a Java program to write some text to a file.

**Test Automation Scenarios**

1. **Login Test:**
   * Write a Java method to automate the login process on a web application using Selenium WebDriver. Assume the login form has fields username and password, and a login button.
2. **Waits in Selenium:**
   * Explain and demonstrate the use of implicit wait and explicit wait in Selenium WebDriver.
3. **Page Object Model:**
   * Explain the Page Object Model (POM) design pattern. Create a simple example to demonstrate how to use POM in a Selenium test.
4. **Data-Driven Testing:**
   * Write a Java program to read test data from an Excel file using Apache POI and use it to perform data-driven testing in Selenium.
5. **API Testing:**
   * Write a Java program to send a GET request to a REST API and verify the response status code.

**Miscellaneous**

1. **Singleton Pattern:**
   * Implement the Singleton design pattern in Java.
2. **JDBC:**
   * Write a Java program to connect to a MySQL database, execute a simple query, and display the results.

**Advanced Topics (Optional)**

1. **Multithreading:**
   * Write a Java program to demonstrate the creation of a thread by extending the Thread class and by implementing the Runnable interface.
2. **Lambda Expressions:**
   * Write a Java program to demonstrate the use of lambda expressions in sorting a list of strings.

These questions cover a wide range of skills that are useful for an automation tester, from basic Java programming to specific scenarios encountered in test automation. They can be used for practice or as part of an interview process to assess a candidate's proficiency in Java and automation testing concepts.

//////////////////////////////////////////////////////////////////////////////////////////////////////////

If you're an automation tester looking to practice and improve your Java skills, it's essential to focus on areas that are particularly relevant to test automation. Below is a structured approach to help you get started:

**1. Fundamentals of Java**

* **Object-Oriented Programming (OOP) Concepts**: Practice classes, objects, inheritance, polymorphism, encapsulation, and abstraction.
* **Data Structures**: Understand arrays, lists, sets, maps, and queues. These are often used in automation for handling test data.
* **Control Structures**: Practice loops (for, while), conditional statements (if, else if, switch), and exception handling (try-catch-finally).

**2. Java Basics for Automation**

* **File Handling**: Learn how to read and write to files, which is crucial for managing test data and results.
* **Collections Framework**: Practice using ArrayList, HashMap, and other collections for storing and manipulating data.
* **String Manipulation**: Practice string operations such as concatenation, comparison, and searching. These are often needed for parsing and validating text in automation.

**3. Test Automation Tools Integration**

* **JUnit/TestNG**: Practice writing test cases using JUnit or TestNG. Understand annotations like @Test, @Before, @After, @DataProvider, etc.
* **Selenium WebDriver**: Practice automating web applications. Focus on interacting with different web elements (e.g., buttons, forms, dropdowns), handling alerts, frames, and windows.
* **Maven/Gradle**: Practice setting up and managing your project dependencies using Maven or Gradle.

**4. Framework Development**

* **Page Object Model (POM)**: Practice creating a Page Object Model framework. This involves creating classes for each page of your application, which encapsulate the web elements and the actions that can be performed on those elements.
* **Data-Driven Testing**: Practice writing test cases that run multiple times with different data sets. Use Excel files, CSVs, or databases as data sources.
* **Keyword-Driven Testing**: Implement a simple keyword-driven framework where keywords define the action to be taken, and the framework reads these keywords from an external file.

**5. Advanced Java for Automation**

* **Multithreading**: Learn how to implement multithreading for executing test cases in parallel. This is particularly useful for large test suites.
* **APIs and RestAssured**: Practice testing RESTful APIs using Java. RestAssured is a popular library for this.
* **Design Patterns**: Understand and implement design patterns like Singleton, Factory, and Strategy, which can be useful in building a robust automation framework.

**6. Continuous Integration/Continuous Deployment (CI/CD)**

* **Jenkins**: Practice setting up a Jenkins job to run your automated tests. Learn how to configure triggers and view test results.
* **Version Control with Git**: Practice using Git for version control. Learn how to create branches, merge code, and resolve conflicts.

**7. Code Quality and Best Practices**

* **Static Code Analysis**: Use tools like SonarQube to analyze the quality of your code. This helps in maintaining high standards in your automation scripts.
* **Refactoring**: Practice refactoring your code to improve readability, reduce complexity, and adhere to best practices.

**8. Practical Projects**

* **Automate a Website**: Choose a website (like an e-commerce site) and create a full test automation suite that covers various functionalities like login, search, checkout, etc.
* **API Testing**: Build a test suite for a sample REST API, including CRUD operations.
* **CI/CD Integration**: Set up a small project in Git, automate the build process with Jenkins, and run your tests automatically with every commit.

**Resources for Learning and Practice:**

* **Books**:
  + "Java: The Complete Reference" by Herbert Schildt.
  + "Effective Java" by Joshua Bloch.
  + "Clean Code" by Robert C. Martin.
* **Online Courses**:
  + **Udemy**: Courses on Selenium WebDriver, TestNG, and Maven.
  + **Pluralsight**: Java courses with a focus on automation testing.
  + **Coursera**: Specializations in Java programming.
* **Practice Platforms**:
  + **LeetCode/HackerRank**: Practice Java coding challenges to improve problem-solving skills.
  + **GitHub**: Look for open-source projects to contribute to, which can give you real-world experience.

**Tips for Effective Practice:**

* **Set up a Realistic Environment**: Try to replicate real-world scenarios in your practice environment.
* **Practice Regularly**: Consistency is key. Dedicate time each day or week to practice.
* **Pair Programming**: If possible, collaborate with peers to learn new techniques and get feedback on your code.
* **Review and Refactor**: Always review your code for improvements. Refactoring helps in learning better coding practices.

This structured approach should help you build a strong foundation in Java for automation testing, making you more proficient and confident in your testing role.