**Software testing**

**Software testing** is the process of evaluating and verifying that a software application or system works as expected. The main goals of software testing are to:

1. **Ensure the software is functional** – It performs the tasks it's supposed to do.
2. **Identify defects (bugs)** – Find errors in the code or behavior.
3. **Ensure reliability and performance** – Confirm it runs smoothly under various conditions.
4. **Validate user requirements** – Make sure it meets the needs and expectations of users or clients.

**Types of Software Testing**

Software testing is broadly classified into two main types:

**1. Manual Testing**

* Testers perform tests manually without using any automation tools.
* Examples: exploratory testing, usability testing.

**2. Automated Testing**

* Uses software tools to run tests automatically.
* Examples: unit testing, regression testing using tools like Selenium, JUnit, etc.

**Common Testing Methods**

* **Unit Testing** – Tests individual units or components of the software.
* **Integration Testing** – Tests combined parts of an application to ensure they work together.
* **System Testing** – Tests the complete system as a whole.
* **Acceptance Testing** – Checks if the system meets business requirements; often done by the client or end-user.
* **Regression Testing** – Ensures that new code changes haven’t broken existing functionality.
* **Performance Testing** – Measures responsiveness, speed, and stability.
* **Security Testing** – Checks for vulnerabilities and ensures data protection.

**Why Is Software Testing Important?**

* Prevents costly bugs in production.
* Improves software quality.
* Enhances user satisfaction.
* Reduces maintenance costs.
* Ensures compliance with standards or regulations.

**Levels of Testing**

In software testing, **"levels of testing"** refer to the different stages at which testing is performed during the software development lifecycle (SDLC). Each level focuses on a specific part of the software system to ensure overall quality.

Here are the **four main levels of software testing**:

**1. Unit Testing**

* **What it tests:** Individual components or functions (units) of the code.
* **Who performs it:** Usually developers.
* **Tools used:** JUnit, NUnit, PyTest, etc.
* **Goal:** Ensure that each unit of the software performs as expected.

*Example: Testing a function that calculates the total price of a shopping cart.*

**2. Integration Testing**

* **What it tests:** Interaction between integrated units/modules.
* **Who performs it:** Developers or testers.
* **Types:** Top-down, bottom-up, sandwich, and big bang.
* **Tools used:** JUnit, TestNG, Postman (for APIs), etc.
* **Goal:** Verify that modules or services work together correctly.

*Example: Testing the interaction between a login module and a user dashboard module.*

**3. System Testing**

* **What it tests:** The complete and integrated software system.
* **Who performs it:** Independent testing team (QA).
* **Types:** Functional testing, usability testing, performance testing, etc.
* **Goal:** Validate the overall behavior and compliance with the specified requirements.

*Example: Testing the whole e-commerce platform — login, browse, cart, payment, and order confirmation.*

**4. Acceptance Testing**

* **What it tests:** The system’s readiness for delivery/use.
* **Who performs it:** End users, clients, or testers.
* **Types:** Alpha testing (internal), Beta testing (external).
* **Goal:** Confirm that the software meets business needs and is acceptable to the user.

*Example: A client tests the software in a real-world scenario to decide whether to accept it.*

**Summary Table:**

| **Level** | **Focus Area** | **Performed By** | **Purpose** |
| --- | --- | --- | --- |
| **Unit Testing** | Individual functions/components | Developers | Ensure code correctness |
| **Integration Testing** | Interactions between units | Developers/Testers | Validate module cooperation |
| **System Testing** | Whole application/system | QA/Testers | Check system functionality |
| **Acceptance Testing** | Final product vs. requirements | End Users/Clients | Approve for release or production |

**Functional**, **Non-functional Testing Types**

Software testing can be categorized into several **types** based on the purpose, approach, and scope of testing. Here’s a comprehensive list of the **main types of software testing**, grouped into **functional**, **non-functional**, **maintenance**, and **other specialized testing** types:

**A. Functional Testing Types**

These tests verify that the software behaves as expected based on requirements.

1. **Unit Testing**
   * Tests individual components or functions.
   * Usually done by developers.
2. **Integration Testing**
   * Tests the interaction between integrated modules.
   * Types: Top-down, Bottom-up, Big bang, Incremental.
3. **System Testing**
   * Tests the complete system as a whole.
4. **Acceptance Testing**
   * Verifies the system meets business requirements.
   * Types:
     + **Alpha Testing**: By internal team.
     + **Beta Testing**: By end users.
5. **Smoke Testing**
   * A basic test to check major functionalities after a build.
6. **Sanity Testing**
   * A narrow regression test focusing on specific bug fixes or functionalities.
7. **Regression Testing**
   * Ensures new changes haven’t broken existing functionality.
8. **Interface Testing**
   * Tests the interaction between systems or software components.
9. **End-to-End Testing**
   * Simulates real user scenarios to validate the system flow.

**B. Non-Functional Testing Types**

These tests validate aspects unrelated to specific behaviors or functions.

1. **Performance Testing**
   * Measures responsiveness, stability, and scalability.
   * Includes:
     + **Load Testing**: Under expected load.
     + **Stress Testing**: Beyond normal load.
     + **Spike Testing**: Sudden spikes in load.
     + **Soak (Endurance) Testing**: Long-duration load testing.
2. **Security Testing**
   * Checks for vulnerabilities and data protection.
   * Includes: penetration testing, ethical hacking, vulnerability scanning.
3. **Usability Testing**
   * Evaluates user experience, interface design, and navigation.
4. **Compatibility Testing**
   * Checks performance across different browsers, OS, devices, etc.
5. **Reliability/Recovery Testing**
   * Ensures software recovers from crashes, hardware failures, or network issues.
6. **Maintainability Testing**
   * Measures how easy it is to maintain and update the software.
7. **Portability Testing**
   * Verifies software works across different environments or platforms.
8. **Compliance Testing**
   * Ensures the software adheres to industry or legal standards.

**C. Maintenance Testing Types**

Used after deployment or during upgrades.

1. **Regression Testing** (again)
   * Re-run of existing test cases after updates.
2. **Maintenance Testing**
   * Includes impact analysis and testing for enhancements or bug fixes.

**D. Specialized/Advanced Testing Types**

1. **Exploratory Testing**
   * Tester explores the application freely without predefined test cases.
2. **Ad-hoc Testing**
   * Informal testing to find defects randomly.
3. **Mutation Testing**
   * Deliberate errors are introduced to evaluate the effectiveness of test cases.
4. **A/B Testing**
   * Compares two versions of a webpage or an app to see which performs better.
5. **Static Testing**
   * Code is reviewed without executing it (e.g., code review, static analysis).
6. **Dynamic Testing**
   * Code is executed during testing.

 Ad-hoc testing: It is an informal testing method that is performed without any pre-defined  
test cases. It is basically rely on the tester's experience and also the knowledge he possess  
to explore the functionalities of the application that are being missed during formal  
testing methods.

**MCQ – Software Testing**

**1. Which of the following is not a level of software testing?**

A. Unit Testing  
B. Integration Testing  
C. System Testing  
D. Validation Testing

**2. Which testing level checks individual components?**

A. Unit Testing  
B. System Testing  
C. Acceptance Testing  
D. Integration Testing

**3. What is the main purpose of integration testing?**

A. To test user requirements  
B. To test interactions between modules  
C. To test interfaces and interactions between units  
D. To find memory leaks

**4. System Testing is performed to:**

A. Test performance  
B. Test modules individually  
C. Validate the complete system  
D. Test code logic

**5. Which level of testing involves end users?**

A. Unit Testing  
B. System Testing  
C. Regression Testing  
D. Acceptance Testing

**6. Regression testing is done to:**

A. Fix broken code  
B. Ensure recent changes haven’t affected existing features  
C. Test new modules  
D. Test performance

**7. Smoke testing is also called:**

A. Build Verification Testing  
B. Sanity Testing  
C. White-box Testing  
D. Functional Testing

**8. What is the goal of sanity testing?**

A. Verify build stability  
B. Verify specific functionalities after minor changes  
C. Validate end-to-end flow  
D. Automate all test cases

**9. Alpha testing is done by:**

A. Internal testers at the developer’s site  
B. External users  
C. Automation tools  
D. Beta users

**10. Beta testing is conducted:**

A. By developers  
B. By real users in a real environment  
C. Internally only  
D. Before coding

**11. Performance testing focuses on:**

A. Logical errors  
B. Speed, scalability, and stability  
C. UI bugs  
D. Test case writing

**12. Which level of testing comes after integration testing?**

A. Unit Testing  
B. Acceptance Testing  
C. System Testing  
D. Smoke Testing  
**Answer: C. System Testing**

**13. What is the primary focus of acceptance testing?**

A. Internal logic of the code  
B. Meeting business requirements  
C. Test case coverage  
D. Performance tuning

**ANSWER KEY**

1. Which of the following is not a level of software testing?  
Answer: D. Validation Testing

2. Which testing level checks individual components?  
Answer: A. Unit Testing

3. What is the main purpose of integration testing?  
Answer: C. To test interfaces and interactions between units

4. System Testing is performed to:  
Answer: C. Validate the complete system

5. Which level of testing involves end users?  
Answer: D. Acceptance Testing

6. Regression testing is done to:  
Answer: B. Ensure recent changes haven’t affected existing features

7. Smoke testing is also called:  
Answer: A. Build Verification Testing

8. What is the goal of sanity testing?  
Answer: B. Verify specific functionalities after minor changes

9. Alpha testing is done by:  
Answer: A. Internal testers at the developer’s site

10. Beta testing is conducted:  
Answer: B. By real users in a real environment

11. Performance testing focuses on:  
Answer: B. Speed, scalability, and stability

12. Which level of testing comes after integration testing?  
Answer: C. System Testing

13. What is the primary focus of acceptance testing?  
Answer: B. Meeting business requirements

                                                           ----------------

**Various Testing Techniques**

**Black Box Testing**

* **Focus**: Functionality of the application without knowledge of internal code.
* **Tester’s Knowledge**: No access to internal code or architecture.
* **Main Goal**: Validate outputs against inputs.
* **Used for**: Functional testing, system testing, acceptance testing.
* **Examples**:
  + Input validation
  + UI testing
  + Functional test cases

**White Box Testing (aka Clear Box or Glass Box Testing)**

* **Focus**: Internal structure and logic of the code.
* **Tester’s Knowledge**: Full access to source code and design.
* **Main Goal**: Verify internal operations and logic.
* **Used for**: Unit testing, security testing, code coverage.
* **Examples**:
  + Statement coverage
  + Branch coverage
  + Path testing

**Grey Box Testing**

* **Focus**: Combines elements of both black box and white box testing.
* **Tester’s Knowledge**: Partial knowledge of internal code/architecture.
* **Main Goal**: Test both user-level functionality and internal logic.
* **Used for**: Integration testing, penetration testing, end-to-end testing with insight.
* **Examples**:
  + Session management testing
  + API security with known endpoints

**Similar or Related Testing Techniques**

**Glass Box Testing**

* Another name for White Box Testing.

**Opaque Box Testing**

* Another name for Black Box Testing.

**Clear Box Testing**

* Synonym of White Box Testing.

**Transparent Box Testing**

* Also refers to White Box Testing.

**Extended Concepts & Related Testing Types**

| **Testing Type** | **Description** | **Relation** |
| --- | --- | --- |
| **Unit Testing** | Testing individual units of code | White Box |
| **Functional Testing** | Validating application functionality against requirements | Black Box |
| **Integration Testing** | Testing how modules work together | Grey Box / White Box |
| **System Testing** | Testing the complete integrated system | Black Box |
| **Acceptance Testing (UAT)** | Validating the system against business needs | Black Box |
| **Regression Testing** | Ensuring new changes don’t affect existing features | Black/Grey Box |
| **Security Testing** | Checking for vulnerabilities | Grey Box (esp. in penetration testing) |
| **Penetration Testing** | Simulated attacks on system | Grey Box |
| **Code Coverage Analysis** | Measuring how much code is tested | White Box |
| **Mutation Testing** | Introducing small changes in code to test the test cases | White Box |

**Summary Table**

| **Feature** | **Black Box** | **White Box** | **Grey Box** |
| --- | --- | --- | --- |
| Knowledge of Code | No | Yes | Partial |
| Testing Type | Functional | Structural | Combination |
| Example Use Case | UI Testing | Unit Testing | Security Testing |
| Tester Role | End-user/tester | Developer/Tester | Developer/Test Analyst |
| Speed to Execute | Faster | Slower (in-depth) | Moderate |

**1.** Which of the following best describes black box testing?  
A) Testing internal logic  
B) **Testing without knowledge of internal code**  
C) Testing performed by developers  
D) Testing memory leaks

**2.** What type of testing is black box testing?  
A) Structural  
B) Code-based  
C) **Functional**  
D) Debugging

**3.** Which technique is commonly used in black box testing?  
A) Statement coverage  
B) **Decision coverage**  
C) Equivalence partitioning  
D) Cyclomatic complexity

**4.** Which is **not** a black box testing technique?  
A) Boundary value analysis  
B) Decision table testing  
C) Use case testing  
D) **Control flow testing**

**5.** What is the main goal of black box testing?  
A) Test how the code is written  
B) **Verify functionality against requirements**  
C) Check variable initialization  
D) Improve code performance

**6.** White box testing is also known as:  
A) Behavioral testing  
B) **Structural testing**  
C) System testing  
D) Integration testing

**7.** Which of the following is used in white box testing?  
A) State transition  
B) Error guessing  
C) **Code coverage**  
D) Boundary value

**8.** Which is **not** a white box testing technique?  
A) Statement coverage  
B) Path coverage  
C) Loop testing  
D) **Decision table testing**

**9.** Who typically performs white box testing?  
A) Testers  
B) **Developer**s  
C) Business analysts  
D) Project managers

**10.** Which of the following is the **best** white box testing tool?  
A) Selenium  
B) **JUnit**  
C) LoadRunner  
D) QTP

**11.** Grey box testing is a combination of:  
A) Regression and acceptance testing  
B) Unit and integration testing  
C) **Black box and white box testing**D) Manual and automated testing

**12.** Which of the following has **partial** knowledge of internal code?  
A) Black box testing  
B) White box testing  
C) **Grey box testing**  
D) Fuzz testing

**13.** Grey box testing is usually performed by:  
A) End users  
B) Developers only  
C) **Testers with programming knowledge**  
D) Business analysts

**14.** A test case designed with **database schema knowledge** but without full code access is an example of:  
A) White box testing  
B) Black box testing  
C) **Grey box testing**  
D) Unit testing

**15.** Which of the following is an **advantage** of grey box testing?  
A) Complete code access  
B) Fully unbiased testing  
C) **Balanced approach between code and functionality**  
D) No requirement for domain knowledge

**16.** Which testing type focuses on **boundary values**?  
A) White box  
B) **Black box**  
C) Grey box  
D) All of the above

**17.** Path coverage is most relevant to:  
A) Black box testing  
B) Grey box testing  
C) **White box testing**  
D) Integration testing

**18.** What is a limitation of black box testing?  
A) Cannot check performance  
B**) Cannot verify internal code paths**C) Cannot check system requirements  
D) Requires extensive coding

**19.** In white box testing, which test coverage metric is **least** useful?  
A) Path coverage  
B) Statement coverage  
C) Function coverage  
D) **UI coverage**

**20.** Which of these is **true** about grey box testing?  
A) No access to code  
B) Only external interfaces are tested  
C) Only developers can perform it  
D**) Internal data structures are partially known**

//batches example

package jdbcmysql;

import java.sql.\*;  
public class PreparedStatementEx{  
    public static void main(String[] args) {  
        String url = "jdbc:mysql://localhost:3306/java\_selenium"; // Replace with your DB URL  
        String user = "root";  
        String password = "root@123";

        String insertSQL = "INSERT INTO users (name, email) VALUES (?, ?)";  
        String delSQL ="delete from users where name=?";  
        //String updSQL="update users set name=? where name='Harry Potter'";  
        String updSQL="update users set NAME=? where id=8";  
  
        try(  
            Connection conn = DriverManager.getConnection(url, user, password);  
  
            PreparedStatement pstmtIns = conn.prepareStatement(insertSQL);  
            PreparedStatement pstmtDel = conn.prepareStatement(delSQL);  
            PreparedStatement pstmtUpd= conn.prepareStatement(updSQL);  
  
          )     
          {  
            // Set values for the placeholders  
          pstmtIns.setString(1, "Adam");  
          pstmtIns.setString(2, "adam@example.com");  
          pstmtIns.addBatch();  
  
          pstmtIns.setString(1, "Kavin");  
          pstmtIns.setString(2, "kavin@example.com");  
          pstmtIns.addBatch();  
  
          pstmtIns.setString(1, "Miller");  
          pstmtIns.setString(2, "miller@example.com");  
          pstmtIns.addBatch();  
  
          pstmtIns.executeBatch();  
          //conn.commit();  
          System.out.println("Batch Executed Successfully");  
  
  
  
          /\*  //For deleting  
            pstmtDel.setString(1,"Adam");  
  
            //For updating  
            pstmtUpd.setString(1,"Harry");  
  
            //Result  
            int rowsInserted = pstmtIns.executeUpdate();  
            int rowsDeleted= pstmtDel.executeUpdate();  
            int rowsUpdated= pstmtUpd.executeUpdate();  
  
            System.out.println(rowsInserted + " row(s) inserted.");  
            System.out.println(rowsDeleted+ " row(s) deleted.");  
            System.out.println(rowsUpdated+ " row(s) updated.");  
        \*/  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}

Callable interface

package jdbcmysql;

import java.util.concurrent.Callable;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.Future;

// Callable is a functional interface with only 1 abstract method "call()"  
// It is a part of "java.util.concurrent" package.  
// This interface is the parent interface. Callable interface is   
// not extending any Parent interface  
/\* With Runnable interface we cannot return a value.  
\* With Callable interface we can return a value.  
\*/

class MyCallable implements Callable<String>{  
  public String call() throws Exception{  
    Thread.sleep(1000);  
    return "With Callable interface we can return a value";  
  }  
  
}

public class CallableExample {  
  public static void main(String[] args) throws Exception{  
    //Step 1 Creates a Thread pool and invoke the call()   
    //method of a Callable Interface  
    ExecutorService executor = Executors.newSingleThreadExecutor();  
  
    // Step 2: Creates a Callable task  
    MyCallable task = new MyCallable();   
  
    //Step 3: We will return the future object  
    Future<String> future = executor.submit(task);   
    String result = future.get();  
    System.out.println(result);  
    executor.shutdown();  
  }  
}

**Software Testing Questions (4-5 liners)**

1. **Explain the importance of software testing in the software development life cycle.**

Software testing is very important phase in SDLC as it performs intended tasks correctly, operates under expected conditions. Also it has many benefits like early bug detection which reduces the cost and enhances the satisfaction of a user and gains their trust.

1. **Discuss common challenges faced during software testing and suggest possible solutions.**

Software testing often faces hurdles like incomplete or frequently changing requirements, which can be addressed by involving testers early and adopting agile practices. Time constraints are another issue, solvable through test automation and prioritizing high-risk areas. A lack of skilled testers can impact quality, so regular training and cross-functional collaboration are key. Test environment instability and poor test data management can lead to unreliable results—using containerization, synthetic data, and dedicated environments helps mitigate this. Lastly, communication gaps and difficulties with third-party integrations can be resolved through better collaboration tools, mock services, and shared ownership of quality.

1. **Differentiate between manual testing and automated testing with examples.**

Manual Testing

The human testers perform it manually. Some of examples are exploratory, usability testing. Manual testing is Flexible and inituitve.

Automated Testing

It is a tool to execute tests automatically. It is fast.

Examples: unit testing, regression testing with selenium, junit.

1. **Compare and contrast functional testing and non-functional testing.**

Functional focuses on what the system does. Some of the examples for for functional testing ---unit, integration and acceptance testing. The main goal of functional testing is to validate the business logic. It uses tools like Junit, TestNG.

Non functional testing focuses on how the system performs. Examples of non functional testing – performance, security, usability testing. It validates the system attributes.

1. **Explain black-box testing, white-box testing, and grey-box testing with real-world scenarios.**

**Black-Box Testing** involves evaluating software without any knowledge of its internal code or logic. The tester focuses purely on inputs and expected outputs to ensure the system behaves correctly from the user's perspective. For example, a tester validating the login functionality of a banking app enters a username and password and checks whether the app logs in successfully or displays an error—without knowing how the authentication works behind the scenes.

**White-Box Testing** requires full access to the internal code and structure of the software. Testers (often developers) write tests that cover specific logic paths, conditions, and branches to ensure the internal operations function as intended. For instance, a developer testing a fund transfer algorithm checks whether the balance updates correctly, transactions fail when funds are insufficient, and valid transaction IDs are generated.

**Grey-Box Testing** is a blend of both approaches, where the tester has partial knowledge of the internal workings. This allows for more informed testing of both functionality and data flow. A common scenario is a security analyst testing a password reset feature—they know the app sends a reset token via email and stores it in the database, so they verify the token format, intercept the email, and check whether expired tokens are properly rejected.

1. **Discuss the significance of regression testing.**

Regression testing makes sure that new changes do not interrupt the exisiting functionality.

It re runs the previous test cases after the updates.

Maintains software stability and reliability.

Detects side effects of bug fixes.

1. **Explain the four levels of software testing: unit testing, integration testing, system testing, and acceptance testing.**

Unit testing: It is used for testing individual components and it is mostly used by developers.

Integration testing: It shows the test interactions between modules and it can use dby developers or testers.

System testing:It is used validate the complete system and it is performed by QA and testers..

Acceptance testing: It confirm the Business requirements and readiness and it is performed by clients or end users.

1. **Why is unit testing considered the foundation of software testing? Discuss its tools and best practices.**

Unit testing is the foundation because it happens at the earliest stage of the software development lifecycle. By testing each code unit, developers can catch and fix bugs at the cheapest to resolve. Tools for unit testing include JUnit for Java and PyTest for Python. Best practices involve writing small, focused tests for each unit of code, making sure the tests are independent of each other, and using these tests as a form of documentation for the code

**What is JUnit in Java?**

JUnit is a **unit testing framework** for the Java programming language. It is used to write and run repeatable tests to ensure that individual parts (units) of your program (usually methods in classes) work as expected.

JUnit is part of the xUnit family of testing frameworks and is one of the most popular testing tools in Java development. It is widely used for Test-Driven Development (TDD).

**Key Features of JUnit:**

* Annotations to identify test methods
* Assertions to test expected vs actual results
* Test runners to execute tests
* Integration with build tools (e.g., Maven, Gradle) and IDEs (e.g., IntelliJ, Eclipse)

**Common JUnit Annotations**

| **Annotation** | **Description** |
| --- | --- |
| @Test | Marks a method as a test method |
| @BeforeEach | Runs before each test method |
| @AfterEach | Runs after each test method |
| @BeforeAll | Runs once before all tests in the class |
| @AfterAll | Runs once after all tests in the class |
| @Disabled | Disables a test method |
| @DisplayName | Sets a custom name for the test method |

**Common JUnit Methods (Assertions)**

*JUnit provides a set of assertion methods to check values.*

| **Method** | **Purpose** |
| --- | --- |
| assertEquals(expected, actual) | Checks if two values are equal |
| assertNotEquals(expected, actual) | Checks if two values are not equal |
| assertTrue(condition) | Checks if the condition is true |
| assertFalse(condition) | Checks if the condition is false |
| assertNull(value) | Checks if the value is null |
| assertNotNull(value) | Checks if the value is not null |
| assertThrows(Exception.class, () -> ...) | Expects an exception to be thrown |
| assertAll(...) | Group multiple assertions |
| fail() | Forces a test to fail |

**Example JUnit Test**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

public class CalculatorTest {

   @Test

   void testAddition() {

       int result = 2 + 3;

       assertEquals(5, result);

   }

   @Test

   void testIsEven() {

       boolean isEven = (4 % 2 == 0);

      assertTrue(isEven);

   }

   @Test

   void testNullValue() {

       String str = null;

       assertNull(str);

   }

}

 Calculatortest

package test.java;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.Test;  
import main.java.Calculator;

public class CalculatorTest {  
  Calculator calc = new Calculator();  
  
  @Test  
  public void testAdd() {  
    assertEquals(64,calc.add(60, 4));  
  }  
  
  @Test  
  public void testSub() {  
    assertEquals(20,calc.sub(50, 30));  
  }  
  
  @Test  
  public void testMul() {  
    assertEquals(100,calc.mul(10, 10));  
  }  
  
  @Test  
  public void testDiv() {  
    assertEquals(25,calc.div(100, 4));  
  }  
}

A.  
Create a class: Box and calculate the following using   
respective methods with parameters.  
1.   Calculate the Area of a circle  
2.  Calculate the area of a triangle  
3.  Calculate the area of a square  
4.  Calculate the Area of a rectangle  
5.  Calculate the Simple Interest (p,r,t)  
6.  Calculate the Amount without parameters  
        Amount = SI + P(1+r/100)^t  
  
  
B.  
Create a Test Class to test all the above methods.

**package** main.java;

**public** **class** Box {

**public** **double** areaOfCircle(**double** radius) {

**return** Math.***PI***\*radius\*radius;

}

**public** **double** areaOfTriangle(**double** base, **double** height) {

**return** 0.5\*base\*height;

}

**public** **double** areaOfSquare(**double** side) {

**return** side\*side;

}

**public** **double** areaOfRectangle(**double** length, **double** breadth) {

**return** length\*breadth;

}

**public** **double** simpleInterest(**double** principal, **double** rate, **double** time) {

**double** result=(principal\*rate\*time)/100;

**return** result;

}

**public** **double** amount(**double** principal, **double** rate, **double** time) {

**double** SimpleInterest=simpleInterest(principal,rate,time);

**double** compound = principal \* Math.*pow*((1 + rate / 100), time);

**return** SimpleInterest + compound;

}

}

**package** test.java;

**import** **static** org.junit.jupiter.api.Assertions.*assertEquals*;

**import** org.junit.Test;

**import** main.java.Box;

**public** **class** BoxTest {

Box box=**new** Box();

@Test

**public** **void** testAreaOfCircle() {

**double** delta=0.1;//we can use this when we use math.pi

*assertEquals*(78.5, box.areaOfCircle(5),delta);

}

@Test

**public** **void** testAreaOfTriangle() {

*assertEquals*(0.72, box.areaOfTriangle(1.2, 1.2));

}

@Test

**public** **void** testAreaOfSquare() {

*assertEquals*(6.25, box.areaOfSquare(2.5));

}

@Test

**public** **void** testAreaOfRectangle() {

*assertEquals*(0.52, box.areaOfRectangle(1.3, 0.4));

}

@Test

**public** **void** testSimpleInterest() {

*assertEquals*(99.0, box.simpleInterest(1000, 2.2, 4.5));

}

@Test

**public** **void** testAmount() {

*assertEquals*(1202.5, box.amount(1000,5,2));

}

}

create a class that will perform the following operations:  
1.  insert the data in the table.  
2.  delete the data from the table.  
3.  update the data in the table.  
Perform a Test Case for every method mentioned above.

**package** main.java;

**import** java.sql.\*;

**public** **class** Movies {

**private** Connection con;

**public** **void** connect() **throws** SQLException {

con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/java\_selenium", "root", "root");

}

**public** **int** insertMovie(String title, String director, String genre, **int** release\_year) **throws** SQLException {

String query = "INSERT INTO movies (title, director, genre, release\_year) VALUES (?, ?, ?, ?)";

PreparedStatement stmt = con.prepareStatement(query);

stmt.setString(1, title);

stmt.setString(2, director);

stmt.setString(3, genre);

stmt.setInt(4, release\_year);

**return** stmt.executeUpdate();

}

**public** **int** updateMovieTitle(String oldTitle, String director, String newTitle) **throws** SQLException {

String query = "UPDATE movies SET title = ? WHERE title = ? AND director = ?";

PreparedStatement stmt = con.prepareStatement(query);

stmt.setString(1, newTitle);

stmt.setString(2, oldTitle);

stmt.setString(3, director);

**return** stmt.executeUpdate();

}

**public** **int** deleteMovieByYear(**int** release\_year) **throws** SQLException {

String query = "DELETE FROM movies WHERE release\_year = ?";

PreparedStatement stmt = con.prepareStatement(query);

stmt.setInt(1, release\_year);

**return** stmt.executeUpdate();

}

**public** **void** close() **throws** SQLException {

**if** (con != **null** && !con.isClosed()) {

con.close();

}

}

}

**package** test.java;

**import** main.java.Movies;

**import** org.junit.jupiter.api.\*;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**public** **class** MoviesTest {

**private** Movies moviestest;

@BeforeEach

**public** **void** setUp() **throws** Exception {

moviestest = **new** Movies();

moviestest.connect();

System.***out***.println("Connection established.");

}

@Test

@DisplayName("Insert Movie Test - The Dark Knight")

**public** **void** testInsertMovie() **throws** Exception {

**int** result = moviestest.insertMovie("The Dark Knight", "Christopher Nolan", "Action", 2008);

*assertEquals*(1, result);

}

@Test

@DisplayName("Update Movie Title Test - Interstellar to Tenet")

**public** **void** testUpdateMovieTitle() **throws** Exception {

moviestest.insertMovie("Interstellar", "Christopher Nolan", "Sci-Fi", 2014);

**int** result = moviestest.updateMovieTitle("Interstellar", "Christopher Nolan", "Tenet");

System.***out***.println("Update result: " + result);

*assertEquals*(1, result);

}

@Test

@DisplayName("Delete Movie Test - Parasite")

**public** **void** testDeleteMovie() **throws** Exception {

moviestest.insertMovie("Parasite", "Bong Joon-ho", "Thriller", 2019);

**int** result = moviestest.deleteMovieByYear(2019);

*assertEquals*(1, result);

}

@AfterEach

**public** **void** closing() **throws** Exception {

moviestest.close();

System.***out***.println("Connection closed.");

}

}

**MCQ - JUnit (Testing)**

**What is JUnit used for in Java?**

A. Database management  
B**. Unit testing**  
C. GUI design  
D. Web development

**2. Which of the following annotations is used to indicate a test method in JUnit 4?**

A. @TestCase  
B. @Before  
C. **@Test**  
D. @RunWith

**3. Which assertion method is used to check equality in JUnit?**

A. assertEqual()  
B. **assertEquals()**C. checkEquals()  
D. testEqual()

**4. In JUnit 5, which annotation is used before each test method?**

A. @Before  
B. @BeforeTest  
C. **@BeforeEach**  
D. @Setup

**5. What does the @AfterEach annotation do in JUnit 5?**

A. Runs after all tests  
B. Runs once before all tests  
**C. Runs after each test method**  
D. Runs only if the test fails

**6. Which JUnit version introduced @BeforeEach and @AfterEach?**

A. JUnit 3  
B. JUnit 4  
C. **JUnit 5**  
D. JUnit 2

**7. Which method will fail a test explicitly in JUnit?**

A. assertFail()  
B. testFail()  
C. **fail()**  
D. throwError()

**8. Which annotation is used to run code once before all test methods in a class (JUnit 5)?**

A. @BeforeEach  
B. **@BeforeAll**  
C. @BeforeClass  
D. @Init

**9. What is the correct return type of a test method in JUnit?**

A. int  
B. String  
C. **void**  
D. boolean

**10. What is used to group multiple assertions in JUnit 5?**

A. **assertAll()**  
B. allAssertions()  
C. assertGroup()  
D. testGroup()

**11. In JUnit, which exception is thrown when a test fails?**

A. **AssertionFailedError**  
B. TestFailException  
C. IllegalArgumentException  
D. RuntimeException

**12. How do you ignore a test method in JUnit 4?**

A. @Skip  
B. @Disable  
C. **@Ignore**  
D. @NotTest

**13. How do you ignore a test method in JUnit 5?**

A. @Ignore  
B. **@Disabled**  
C. @Skip  
D. @Omit

**14. Which dependency do you need to include for using JUnit 5 in Maven?**

A. junit:junit  
B. junit:junit-api  
C. **org.junit.jupiter:junit-jupiter**  
D. org.junit:junit-vintage

**15. Which annotation is used to specify the order of test execution in JUnit 5?**

A. @TestOrder  
B. **@Order**  
C. @TestSequence  
D. @ExecutionOrder

**16. What is the purpose of @Nested in JUnit 5?**

A. **To declare inner tests**  
B. To create a new test class  
C. To run tests in parallel  
D. To skip nested classes

**17. What is the default test runner in JUnit 4?**

A. JUnitCore  
B. **TestNG**  
C. BlockJUnit4ClassRunner  
D. SuiteRunner

**18. Which method is used to check if a value is true in JUnit?**

A. **assertTrue()**  
B. checkTrue()  
C. isTrue()  
D. assert()

**19. Which annotation is used to create test lifecycle methods in JUnit 5?**

A. @Lifecycle  
B. @TestLife  
C. **@BeforeEach and @AfterEach**  
D. @RunWith

**20. What annotation is used in JUnit to run parameterized tests?**

A. @RunWith(Parameterized.class)  
B. **@ParameterizedTest**C. @WithParameters  
D. Both A and B

**21. Which method is used in JUnit to check for null values?**

A. assertIsNull()  
B. **assertNull(**)  
C. isNull()  
D. assertEquals(null)

**22. What is Assertions.assertTimeout() used for?**

A. To retry failed tests  
B. **To check if code executes within a time limit**  
C. To delay test execution  
D. To log test time

**23. How do you test exceptions in JUnit 5?**

A. assertError()  
B. **assertThrows()**  
C. throwsException()  
D. expectException()

**24. What does @Test(expected = Exception.class) do in JUnit 4?**

A. Asserts a value is expected  
B. **Marks test as failed if no exception is thrown**  
C. Ignores the test  
D. Runs the test repeatedly

**25. What feature of JUnit allows you to test multiple inputs?**

A. Loops  
B. Test Suite  
C. **Parameterized Tests**  
D. Repeated Tests

**26. How do you test repeated execution in JUnit 5?**

A. @Loop  
B. @TestRepeat  
C. @**RepeatedTest**  
D. @MultipleTest

**27. What is the purpose of @TestInstance in JUnit 5?**

A. To allow test inheritance  
B. **To define test lifecycle**  
C. To enable parameterized tests  
D. To create mock objects

**28. Which of the following is true about JUnit?**

A. It is only for GUI testing  
B. It runs only on Windows  
C. **It is a unit testing framework**  
D. It cannot be used with Maven

**29. What is the function of assertSame() in JUnit?**

A. Checks if values are same by content  
B. **Checks if references point to the same object**  
C. Checks for different objects  
D. Checks null values

Calculator.java  
==============  
public int divide(int a,int b){  
  if(b==0) throw new IllegalArgumentException("Cannot divide by zero");  
  return a/b;    
}

CalculatorTest.java  
===================  
@Test  
public void testDivide(){  
  Exception ex = assertThrows  
  (IllegalArgumentException.class,  
     ()->calculator.divide(5,0);  
  assertEquals("cannot Divide By Zero",ex.getMessage());  
  });

**What is JUnit in Java?**

JUnit is a **unit testing framework** for the Java programming language. It is used to write and run repeatable tests to ensure that individual parts (units) of your program (usually methods in classes) work as expected.

JUnit is part of the xUnit family of testing frameworks and is one of the most popular testing tools in Java development. It is widely used for Test-Driven Development (TDD).

**Key Features of JUnit:**

* Annotations to identify test methods
* Assertions to test expected vs actual results
* Test runners to execute tests
* Integration with build tools (e.g., Maven, Gradle) and IDEs (e.g., IntelliJ, Eclipse)

**Common JUnit Annotations**

| **Annotation** | **Description** |
| --- | --- |
| @Test | Marks a method as a test method |
| @BeforeEach | Runs before each test method |
| @AfterEach | Runs after each test method |
| @BeforeAll | Runs once before all tests in the class |
| @AfterAll | Runs once after all tests in the class |
| @Disabled | Disables a test method |
| @DisplayName | Sets a custom name for the test method |

**Common JUnit Methods (Assertions)**

*JUnit provides a set of assertion methods to check values.*

| **Method** | **Purpose** |
| --- | --- |
| assertEquals(expected, actual) | Checks if two values are equal |
| assertNotEquals(expected, actual) | Checks if two values are not equal |
| assertTrue(condition) | Checks if the condition is true |
| assertFalse(condition) | Checks if the condition is false |
| assertNull(value) | Checks if the value is null |
| assertNotNull(value) | Checks if the value is not null |
| assertThrows(Exception.class, () -> ...) | Expects an exception to be thrown |
| assertAll(...) | Group multiple assertions |
| fail() | Forces a test to fail |

**Example JUnit Test**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

public class CalculatorTest {

   @Test

   void testAddition() {

       int result = 2 + 3;

       assertEquals(5, result);

   }

   @Test

   void testIsEven() {

       boolean isEven = (4 % 2 == 0);

      assertTrue(isEven);

   }

   @Test

   void testNullValue() {

       String str = null;

       assertNull(str);

   }

}

**Test Planning and Design**

*Test Planning and Test Design are two critical stages in the software testing lifecycle.*

***They help ensure that testing is thorough, organized, and aligned with project goals.***

**1. Test Planning**

**What is Test Planning?**

Test planning is the process of defining the scope, objectives, approach, and resources required for testing activities. It results in a Test Plan document, which acts as a blueprint for all testing efforts.

**Key Components of a Test Plan:**

| **Section** | **Description** |
| --- | --- |
| **Test Objectives** | What is being tested and why. |
| **Test Scope** | What **will** and **won’t** be tested. |
| **Test Strategy** | Overall testing approach (manual, automated, black box, etc.). |
| **Test Environment** | Hardware, software, and network setups required. |
| **Test Schedule** | Timeline for each test activity. |
| **Resources & Roles** | Who will perform what tasks (testers, developers, tools). |
| **Entry & Exit Criteria** | Conditions to start and stop testing. |
| **Risk Management** | Potential issues and mitigation strategies. |
| **Deliverables** | What will be produced (test cases, bug reports, metrics). |

**Example:**

*In a* ***banking app*** *project, the test plan may state:*  
**"Functional testing will be performed on the login, account summary, and fund transfer modules using a black-box approach. Testing will begin on Sept 1 and end by Sept 20. Selenium will be used for automation."**

**2. Test Design**

**What is Test Design?**

Test design is the process of creating detailed test cases and test data based on the requirements and specifications defined in the planning phase.

**Key Activities in Test Design:**

| **Activity** | **Description** |
| --- | --- |
| **Requirement Analysis** | Understand what needs to be tested. |
| **Test Scenario Identification** | High-level descriptions of what to test. |
| **Test Case Design** | Detailed steps, inputs, expected outcomes. |
| **Test Data Creation** | Input values and environmental data required for tests. |
| **Traceability Matrix** | Mapping test cases to requirements for full coverage. |

**Example of a Test Case:**

| **Field** | **Example** |
| --- | --- |
| **Test Case ID** | TC\_Login\_001 |
| **Description** | Verify login with valid credentials |
| **Steps** | 1. Open login page 2. Enter valid username and password 3. Click login |
| **Test Data** | Username: user123 Password: pwd123 |
| **Expected Result** | User is redirected to the dashboard |
| **Actual Result** | (Filled during testing) |
| **Status** | Pass(Green Bar)/Fail(Red Bar) |

**Summary: Test Planning vs. Test Design**

| **Aspect** | **Test Planning** | **Test Design** |
| --- | --- | --- |
| **Goal** | Define *what* and *how* to test | Create *detailed* test cases |
| **Focus** | Strategy, resources, scheduling | Test scenarios, steps, test data |
| **Output** | Test Plan document | Test cases, test scripts, test data |
| **Performed By** | Test Manager / Lead | QA Engineers / Testers |

**Test Execution in Software Testing**

**Test Execution** is the phase where the actual testing takes place — test cases are run, results are recorded, and any deviations (bugs or defects) are logged for fixing.

**What is Test Execution?**

Test execution is the process of running the test cases designed during the test design phase and comparing actual results with expected results to determine if the software behaves as intended.

**Key Activities in Test Execution**

| **Step** | **Description** |
| --- | --- |
| **1. Test Environment Setup** | Make sure the hardware/software environment is ready (e.g., test server, database, network, external jar files to be imported). |
| **2. Test Case Execution** | Run manual or automated test cases. |
| **3. Record Results** | Log actual results and compare with expected outcomes. |
| **4. Defect Reporting** | If the test fails, log a **defect/bug** in a tracking tool (e.g., Jira, Bugzilla). |
| **5. Retesting and Regression Testing** | Once a defect is fixed, re-run test cases and check related functionality. |
| **6. Test Status Reporting** | Track progress (e.g., % passed, failed, blocked) and report to stakeholders. |

**Example:**

| **Test Case ID** | **Description** | **Expected Result** | **Actual Result** | **Status** | **Defect ID** |
| --- | --- | --- | --- | --- | --- |
| TC\_Login\_001 | Valid login credentials | Redirect to dashboard | Redirect to dashboard | Pass | – |
| TC\_Login\_002 | Invalid password | Show error message | Login page reloads | Fail | BUG\_105 |
| TC\_Logout\_003 | Click logout button | Redirect to login screen | Redirect to login screen | Pass | – |

**Tools Commonly Used in Test Execution**

| **Type** | **Tools** |
| --- | --- |
| **Test Management** | TestRail, Zephyr, Xray, HP ALM |
| **Bug Tracking** | Jira, Bugzilla, Mantis |
| **Automation** | Selenium, Cypress, JUnit, TestNG |
| **CI/CD Execution** | Jenkins, GitLab CI/CD |

**Test Execution Status Terms**

| **Status** | **Meaning** |
| --- | --- |
| **Pass** | Test ran successfully and matched expectations. |
| **Fail** | Test ran but didn’t meet the expected result. |
| **Blocked** | Test couldn’t be executed due to an external issue (e.g., network down). |
| **Not Run** | Test case wasn’t executed yet. |
| **Skipped** | Test was intentionally skipped (e.g., not relevant for this cycle). |

**Best Practices for Test Execution**

* Prioritize critical test cases first.
* Log defects clearly with steps, screenshots, and logs.
* Communicate regularly with the development team.
* Continuously update test status and reports.
* Track defect fixes and retest promptly.
* **Defect Management in Software Testing**
* **Defect Management** is the structured process of identifying, documenting, prioritizing, tracking, and resolving defects (bugs) found during the software development and testing life cycle.
* **What Is a Defect (Bug)?**
* A **defect** is a flaw or error in a software product that causes it to behave incorrectly or unexpectedly, not meeting the requirements or specifications.
* **Defect Management Process Steps**
* Defect Life Cycle (also called the Bug Life Cycle):

| **Step** | **Description** |
| --- | --- |
| **1. Defect Detection** | A tester or user finds unexpected behavior in the software. |
| **2. Defect Logging** | The issue is logged in a **bug tracking tool** (e.g., Jira, Bugzilla) with details. |
| **3. Defect Triage** | Team analyzes, validates, and assigns priority and severity. |
| **4. Defect Assignment** | The bug is assigned to a developer to fix. |
| **5. Defect Fixing** | The developer investigates and resolves the issue. |
| **6. Retesting** | The tester re-executes the failed test cases to verify the fix. |
| **7. Closure** | If the fix works, the defect is marked **Closed**. If not, it's **Reopened**. |

* **Defect Life Cycle (Status Flow)**
* New → Assigned → Open → Fixed → Retest → Verified → Closed
* ↘ Reopen ↖
* **Key Defect Attributes**

| **Attribute** | **Description** |
| --- | --- |
| **Defect ID** | Unique identifier |
| **Title** | Short summary |
| **Description** | Detailed explanation with steps to reproduce |
| **Severity** | Impact on system (e.g., Critical, Major, Minor) |
| **Priority** | Urgency to fix (e.g., High, Medium, Low) |
| **Status** | Current state (e.g., New, Open, Closed) |
| **Environment** | Where it occurred (e.g., Browser, OS) |
| **Attachments** | Screenshots, logs, video evidence |

**Unit Testing 🡪 System Testing**

Moving from **unit testing** to **system testing** in the software testing lifecycle involves a structured progression through different testing levels. Each level increases in scope and complexity.

**1. Unit Testing**

* **Goal:** Test individual units (functions/methods) of code in isolation.
* **Who performs it:** Developers.
* **Tools:** JUnit, NUnit, PyTest, etc.
* **Key Focus:** Correctness of logic, inputs/outputs, edge cases.

**2. Integration Testing**

* **Goal:** Verify interactions between units/modules.
* **Types:**
  + **Top-down**
  + **Bottom-up**
  + **Big bang**
  + **Incremental**
* **Who performs it:** Developers or testers.
* **Tools:** TestNG, JUnit (with mocks), Postman (for APIs), etc.
* **Key Focus:** Data flow, interfaces, communication between modules.

**3. System Testing**

* **Goal:** Validate the complete and fully integrated application against requirements.
* **Who performs it:** QA/Testers.
* **Environment:** Close to production.
* **Types of System Testing:**
  + Functional Testing
  + Non-functional Testing (Performance, Security, Usability)
* **Tools:** Selenium, JMeter, TestRail, etc.
* **Key Focus:** End-to-end workflows, user scenarios, overall behavior.

**Steps in Transition:**

| **Step** | **Description** |
| --- | --- |
| 1. **Complete Unit Testing** | Ensure all functions/classes are tested with high code coverage. |
| 2. **Set Up Continuous Integration (CI)** | Automate running unit tests on each commit or build. |
| 3. **Identify Integration Points** | Understand where modules interact; define interfaces. |
| 4. **Conduct Integration Testing** | Test data exchange, API calls, database interactions, etc. |
| 5. **Prepare Test Environment** | Set up system test environment resembling production. |
| 6. **Review Requirements & Specs** | Use functional requirements to derive system test cases. |
| 7. **Create System Test Plans & Scripts** | Include end-to-end test cases, performance checks, etc. |
| 8. **Perform System Testing** | Execute full application tests covering all features and workflows. |
| 9. **Log and Track Bugs** | Report and retest defects found during system testing. |

**Summary Flow:**

Unit Testing → Integration Testing → System Testing

      ↓                ↓                 ↓

  Test Code       Test Module       Test Entire App

In Isolation      Interactions        End-to-End

                                                                          ===========

 1. Write a program to print reverse tables  
2. Write a program to print all alphabets from a to z  
3. Write a program to print the ASCII values  
4. Write a program to find the factorial value of any number  
5. Write a program to enter the numbers till the user wants and at the end it should display the count of positive, negative and zeros entered  
6. Write a program to enter the numbers till the user wants and at the end the program should display the largest and smallest numbers entered  
7. Write a program to find number and sum of all integer between 100 and 200 which are divisible by 9  
8. Write a program to find 1s complement of a number in java  
9. Write a program to find 2s complement of a Binary number in java  
10. Write a program to print fibonacci series upto n terms

**package** main.java;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** Programs {

// 1. Reverse Multiplication Table

**public** List<String> reverseTable(**int** num) {

List<String> table = **new** ArrayList<>();

**for** (**int** i = 10; i >= 1; i--) {

table.add(num + " x " + i + " = " + (num \* i));

}

**return** table;

}

// 2. Alphabets a to z as string with spaces

**public** String getAlphabets() {

StringBuilder sb = **new** StringBuilder();

**for** (**char** ch = 'a'; ch <= 'z'; ch++) {

sb.append(ch).append(' ');

}

**return** sb.toString().trim();

}

// 3. ASCII values of a to z,

**public** List<String> printASCIIValues() {

List<String> asciiList = **new** ArrayList<>();

**for** (**char** ch = 'a'; ch <= 'z'; ch++) {

asciiList.add(ch + " = " + (**int**) ch);

}

**return** asciiList;

}

// 4. Factorial using

**public** **int** factorial(**int** num) {

**if** (num < 0) **throw** **new** IllegalArgumentException("Negative numbers not supported");

**int** factorial = 1;

**for** (**int** i = 2; i <= num; i++) {

factorial \*= i;

}

**return** factorial;

}

// 5. Count positive, negative, zero

**public** **int**[] countNumbers(**int**[] numbers) {

**int** pos = 0;

**int** neg = 0;

**int** zero = 0;

//using enhanced for loop

**for** (**int** num : numbers) {

**if** (num > 0)

pos++;

**else** **if** (num < 0)

neg++;

**else**

zero++;

}

**return** **new** **int**[] {pos, neg, zero};

}

// 6. Find min and max

**public** **int**[] findMinMax(**int**[] numbers) {

**if** (numbers == **null** || numbers.length == 0)

**throw** **new** IllegalArgumentException("Array must not be empty");

**int** max = numbers[0];

**int** min = numbers[0];

//using enhanced for loop

**for** (**int** num : numbers) {

**if** (num > max) max = num;

**if** (num < min) min = num;

}

**return** **new** **int**[] {max, min};//storing result in array.

}

// 7. Numbers divisible by 9 between 100 and 200 - returns int array of those numbers

**public** **int**[] divisibleByNine() {

**int** count = 0;

**int** sum = 0;

// Find how many numbers between 100 and 200 are divisible by 9

**for** (**int** i = 100; i <= 200; i++) {

**if** (i % 9 == 0) {

count++;

sum += i;

}

}

**return** **new** **int**[]{count, sum};

}

// 8. 1's complement of integer

**public** **int** onesComplement(**int** num) {

**return** ~num;

}

// 9. 2's complement of integer using ~ + 1

**public** **int** twosComplement(**int** num) {

**return** onesComplement(num) + 1;

}

// 10. Fibonacci series

**public** **int**[] fibonacci(**int** n) {

**if** (n <= 0) **return** **new** **int**[0];

**int**[] fib = **new** **int**[n];

fib[0] = 0;

**if** (n > 1)

fib[1] = 1;

**for** (**int** i = 2; i < n; i++) {

fib[i] = fib[i-1] + fib[i-2];

}

**return** fib;

}

}

**package** test.java;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**import** main.java.Programs;

**import** java.util.List;

**public** **class** ProgramsTest {

Programs programs = **new** Programs();

@Test

**void** testReverseTable() {

List<String> expected = List.*of*(

"5 x 10 = 50",

"5 x 9 = 45",

"5 x 8 = 40",

"5 x 7 = 35",

"5 x 6 = 30",

"5 x 5 = 25",

"5 x 4 = 20",

"5 x 3 = 15",

"5 x 2 = 10",

"5 x 1 = 5"

);

*assertEquals*(expected, programs.reverseTable(5));

}

@Test

**void** testGetAlphabets() {

String expected = "a b c d e f g h i j k l m n o p q r s t u v w x y z";

*assertEquals*(expected, programs.getAlphabets());

}

@Test

**void** testPrintASCIIValues() {

List<String> expected = List.*of*(

"a = 97", "b = 98", "c = 99", "d = 100", "e = 101", "f = 102", "g = 103",

"h = 104", "i = 105", "j = 106", "k = 107", "l = 108", "m = 109", "n = 110",

"o = 111", "p = 112", "q = 113", "r = 114", "s = 115", "t = 116", "u = 117",

"v = 118", "w = 119", "x = 120", "y = 121", "z = 122"

);

*assertEquals*(expected, programs.printASCIIValues());

}

@Test

**void** testFactorial() {

*assertEquals*(1, programs.factorial(0));

*assertEquals*(1, programs.factorial(1));

*assertEquals*(120, programs.factorial(5));

}

@Test

**void** testCountNumbers() {

**int**[] input = {1, -2, 3, 4 ,5 ,-5, -6, -9, 0, 10, 0};

**int**[] expected = {5, 4, 2};

*assertArrayEquals*(expected, programs.countNumbers(input));

}

@Test

**void** testFindMinMax() {

**int**[] input = {5, 3, 8, -2, 0, 11};

**int**[] expected = {11, -2};

*assertArrayEquals*(expected, programs.findMinMax(input));

}

@Test

**void** testDivisibleByNine() {

**int**[] result = programs.divisibleByNine();

**int** expectedCount = 11;

**int** expectedSum = 1683;

*assertEquals*(expectedCount, result[0]);

*assertEquals*(expectedSum, result[1]);

}

@Test

**void** testOnesComplement() {

*assertEquals*(-6, programs.onesComplement(5));

*assertEquals*(0, programs.onesComplement(-1));

*assertEquals*(-1, programs.onesComplement(0));

}

@Test

**void** testTwosComplement() {

*assertEquals*(-5, programs.twosComplement(5));

*assertEquals*(1, programs.twosComplement(-1));

*assertEquals*(0, programs.twosComplement(0));

}

@Test

**void** testFibonacci() {

**int**[] expected = {0, 1, 1, 2, 3};

*assertArrayEquals*(expected, programs.fibonacci(5));

}

}

1.  Java program to print the highest frequency character in a String.  
2.  Java program to Replace First Occurrence Of Vowel With ‘-‘ in String.  
3.  Java program to count alphabets, digits and special characters.  
4.  Java program to remove blank space from string.  
5.  Java program to Concatenate two strings.

 package test.java;

import main.java.StringPrograms;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

public class StringProgramsTest {

StringPrograms programs = new StringPrograms();

// 1. Test highest frequency character

@Test

void testHighestFrequencyChar() {

assertEquals('r', programs.highestFrequencyChar("programming")); // 'r' comes before 'g' and 'm'

assertEquals('l', programs.highestFrequencyChar("hello")); // 'l' appears twice

assertEquals('a', programs.highestFrequencyChar("aaaabbbcc")); // 'a' appears 4 times

}

// 2. Test replace first vowel with '-'

@Test

void testReplaceFirstVowel() {

assertEquals("h-llo", programs.replaceFirstVowel("hello"));

assertEquals("-bc", programs.replaceFirstVowel("abc"));

assertEquals("t-st", programs.replaceFirstVowel("test"));

assertEquals("xyz", programs.replaceFirstVowel("xyz")); // No vowel

}

// 3. Test count alphabets, digits, special characters

@Test

void testCountAlphaDigitSpecial() {

int[] result = programs.countAlphaDigitSpecial("Hello123@#");

assertArrayEquals(new int[]{5, 3, 2}, result);

result = programs.countAlphaDigitSpecial("ABCdef456\*&^");

assertArrayEquals(new int[]{6, 3, 3}, result);

result = programs.countAlphaDigitSpecial("123456");

assertArrayEquals(new int[]{0, 6, 0}, result);

}

// 4. Test remove spaces

@Test

void testRemoveSpaces() {

assertEquals("helloworld", programs.removeSpaces("hello world"));

assertEquals("JavaProgramming", programs.removeSpaces("Java Programming"));

assertEquals("nospace", programs.removeSpaces("nospace"));

}

// 5. Test concatenate strings

@Test

void testConcatenateStrings() {

assertEquals("HelloWorld", programs.concatenateStrings("Hello", "World"));

assertEquals("Java123", programs.concatenateStrings("Java", "123"));

assertEquals("Test", programs.concatenateStrings("", "Test"));

}

}

**package** main.java;

**public** **class** StringPrograms {

// 1. Highest frequency character in a string

**public** **char** highestFrequencyChar(String input) {

**int**[] freq = **new** **int**[256]; // For all ASCII characters

**for** (**char** ch : input.toCharArray()) {

freq[ch]++;

}

**int** maxFreq = 0;

**char** result = 0;

**for** (**char** ch : input.toCharArray()) {

**if** (freq[ch] > maxFreq) {

maxFreq = freq[ch];

result = ch;

}

}

**return** result;

}

// 2. Replace first vowel with '-'

**public** String replaceFirstVowel(String input) {

String vowels = "aeiouAEIOU";

String result = "";

**boolean** replaced = **false**;

**for** (**int** i = 0; i < input.length(); i++) {

**char** ch = input.charAt(i);

**if** (!replaced && vowels.indexOf(ch) != -1) {

result += '-';

replaced = **true**;

} **else** {

result += ch;

}

}

**return** result;

}

// 3. Count alphabets, digits, and special characters

**public** **int**[] countAlphaDigitSpecial(String input) {

**int** alphabets = 0;

**int** digits = 0;

**int** special = 0;

**for** (**char** ch : input.toCharArray()) {

**if** (Character.*isLetter*(ch)) {

alphabets++;

} **else** **if** (Character.*isDigit*(ch)) {

digits++;

} **else** {

special++;

}

}

**return** **new** **int**[]{alphabets, digits, special};

}

// 4. Remove spaces from a string

**public** String removeSpaces(String input) {

String result = "";

**for** (**char** ch : input.toCharArray()) {

**if** (ch != ' ') {

result += ch;

}

}

**return** result;

}

// 5. Concatenate two strings

**public** String concatenateStrings(String str1, String str2) {

**return** str1 + str2;

}

}

1.  Write a Java program to shuffle elements in an array list.  
2.  Write a Java program to extract a portion of an array list.  
3.  Write a Java program to increase an array list size.  
4.  Write a Java program to replace the second element of an ArrayList with the specified element.  
5.  Write a Java program to append the specified element to the end of a linked list.  
6.  Write a Java program to insert the specified element at the front of a linked list  
7.  Write a Java program to clone a hash set to another hash set.  
8.  Write a Java program to convert a hash set to a tree set.  
9.  Write a Java program to get a collection view of the values contained in this map.  
10.  Write a Java program to get a collection view of the keys contained in this map.

**package** main.java;

**import** java.util.\*;

**public** **class** CollectionProgram {

// 1. Shuffle elements in an array list

**public** **static** **void** shuffleArrayList(List<String> list) {

Collections.*shuffle*(list);

}

// 2. Extract a portion of an array list

**public** **static** List<String> extractPortion(List<String> list, **int** fromIndex, **int** toIndex) {

**return** **new** ArrayList<>(list.subList(fromIndex, toIndex));

}

// 3. Increase an array list size (Add nulls to increase capacity)

**public** **static** **void** increaseArrayListSize(List<String> list, **int** newSize) {

**while** (list.size() < newSize) {

list.add(**null**);

}

}

// 4. Replace the second element of an ArrayList

**public** **static** **void** replaceSecondElement(List<String> list, String newElement) {

**if** (list.size() >= 2) {

list.set(1, newElement);

}

}

// 5. Append element to end of linked list

**public** **static** **void** appendToLinkedList(LinkedList<String> list, String element) {

list.addLast(element);

}

// 6. Insert element at the front of linked list

**public** **static** **void** insertAtFront(LinkedList<String> list, String element) {

list.addFirst(element);

}

// 7. Clone a hash set to another hash set

**public** **static** <T> HashSet<T> cloneHashSet(HashSet<T> original) {

**return** (HashSet<T>) original.clone();

}

// 8. Convert hash set to tree set

**public** **static** <T> TreeSet<T> convertToTreeSet(HashSet<T> hashSet) {

**return** **new** TreeSet<>(hashSet);

}

// 9. Get collection view of values in a map

**public** **static** <K, V> Collection<V> getValues(Map<K, V> map) {

**return** map.values();

}

// 10. Get collection view of keys in a map

**public** **static** <K, V> Set<K> getKeys(Map<K, V> map) {

**return** map.keySet();

}

}

**package** main.java;

**import** java.util.\*;

**public** **class** CollectionProgramMain {

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

List<String> arrayList = **new** ArrayList<>(Arrays.*asList*("One", "Two", "Three", "Four", "Five"));

System.***out***.println("Original ArrayList: " + arrayList);

CollectionProgram.*shuffleArrayList*(arrayList);

System.***out***.println("Shuffled ArrayList: " + arrayList);

List<String> portion = CollectionProgram.*extractPortion*(arrayList, 1, 3);//prints only 1,2 index

System.***out***.println("\nExtracted Portion: " + portion);

CollectionProgram.*increaseArrayListSize*(arrayList, 7);

System.***out***.println("\nArrayList after size increase: " + arrayList);

CollectionProgram.*replaceSecondElement*(arrayList, "TwentyFive");

System.***out***.println("\nAfter replacing 2nd element: " + arrayList);

LinkedList<String> linkedList = **new** LinkedList<>(Arrays.*asList*("Red", "Green"));

CollectionProgram.*appendToLinkedList*(linkedList, "Blue");

System.***out***.println("\nLinkedList after append: " + linkedList);

CollectionProgram.*insertAtFront*(linkedList, "Yellow");

System.***out***.println("\nLinkedList after insert at front: " + linkedList);

HashSet<String> hashSet = **new** HashSet<>(Arrays.*asList*("A", "B", "C"));

HashSet<String> clonedSet = CollectionProgram.*cloneHashSet*(hashSet);

System.***out***.println("\nCloned HashSet: " + clonedSet);

TreeSet<String> treeSet = CollectionProgram.*convertToTreeSet*(hashSet);

System.***out***.println("\nTreeSet from HashSet: " + treeSet);

Map<String, Integer> map = **new** HashMap<>();

map.put("Apple", 10);

map.put("Banana", 20);

System.***out***.println("\nMap Values: " + CollectionProgram.*getValues*(map));

System.***out***.println("\nMap Keys: " + CollectionProgram.*getKeys*(map));

}

}

**package** test.java;

**import** org.junit.jupiter.api.\*;

**import** main.java.CollectionProgram;

**import** java.util.\*;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**public** **class** CollectionProgramTest {

@Test

**public** **void** testShuffleArrayList() {

List<String> list = **new** ArrayList<>(Arrays.*asList*("A", "B", "C", "D"));

List<String> original = **new** ArrayList<>(list);

CollectionProgram.*shuffleArrayList*(list);

*assertEquals*(4, list.size());

*assertTrue*(original.containsAll(list));

}

@Test

**public** **void** testExtractPortion() {

List<String> list = Arrays.*asList*("A", "B", "C", "D");

List<String> portion = CollectionProgram.*extractPortion*(list, 1, 3);

*assertEquals*(Arrays.*asList*("B", "C"), portion);

}

@Test

**public** **void** testIncreaseArrayListSize() {

List<String> list = **new** ArrayList<>(Arrays.*asList*("A", "B"));

CollectionProgram.*increaseArrayListSize*(list, 5);

*assertEquals*(5, list.size());

}

@Test

**public** **void** testReplaceSecondElement() {

List<String> list = **new** ArrayList<>(Arrays.*asList*("A", "B", "C"));

CollectionProgram.*replaceSecondElement*(list, "Z");

*assertEquals*("Z", list.get(1));

}

@Test

**public** **void** testAppendToLinkedList() {

LinkedList<String> list = **new** LinkedList<>(Arrays.*asList*("A", "B"));

CollectionProgram.*appendToLinkedList*(list, "C");

*assertEquals*("C", list.getLast());

}

@Test

**public** **void** testInsertAtFront() {

LinkedList<String> list = **new** LinkedList<>(Arrays.*asList*("B", "C"));

CollectionProgram.*insertAtFront*(list, "A");

*assertEquals*("A", list.getFirst());

}

@Test

**public** **void** testCloneHashSet() {

HashSet<String> set = **new** HashSet<>(Arrays.*asList*("X", "Y"));

HashSet<String> cloned = CollectionProgram.*cloneHashSet*(set);

*assertEquals*(set, cloned);

}

@Test

**public** **void** testConvertToTreeSet() {

HashSet<String> set = **new** HashSet<>(Arrays.*asList*("Z", "Y", "X"));

TreeSet<String> treeSet = CollectionProgram.*convertToTreeSet*(set);

*assertEquals*(**new** TreeSet<>(set), treeSet);

}

@Test

**public** **void** testGetValues() {

Map<String, Integer> map = **new** HashMap<>();

map.put("One", 1);

map.put("Two", 2);

Collection<?> values = CollectionProgram.*getValues*(map);

*assertTrue*(values.contains(1));

*assertTrue*(values.contains(2));

}

@Test

**public** **void** testGetKeys() {

Map<String, Integer> map = **new** HashMap<>();

map.put("One", 1);

map.put("Two", 2);

Set<String> keys = CollectionProgram.*getKeys*(map);

*assertTrue*(keys.contains("One"));

*assertTrue*(keys.contains("Two"));

}

}

**A simple Java integration testing example.**

We’ll test the integration of two components:

· A UserRepository that stores users in a file.

· A UserService that uses this repository.

The integration test will check that UserService and UserRepository work correctly together (integration testing = testing the interaction between real components).

---

Project Structure

/project

├── User.java

├── UserRepository.java

├── UserService.java

├── IntegrationTest.java

---

1. User.java

public class User {

private String id;

private String name;

public User(String id, String name) {

this.id = id;

this.name = name;

}

// Getters

public String getId() { return id; }

public String getName() { return name; }

}

---

2. UserRepository.java

import java.io.\*;

import java.util.\*;

public class UserRepository {

private File storageFile;

public UserRepository(String filePath) {

this.storageFile = new File(filePath);

}

public void save(User user) throws IOException {

try (FileWriter writer = new FileWriter(storageFile, true)) {

writer.write(user.getId() + "," + user.getName() + "\n");

}

}

public List<User> findAll() throws IOException {

List<User> users = new ArrayList<>();

if (!storageFile.exists()) return users;

try (BufferedReader reader = new BufferedReader(new FileReader(storageFile))) {

String line;

while ((line = reader.readLine()) != null) {

String[] parts = line.split(",");

users.add(new User(parts[0], parts[1]));

}

}

return users;

}

public void clear() throws IOException {

new FileWriter(storageFile).close(); // clear contents

}

}

---

3. UserService.java

import java.io.IOException;

import java.util.List;

public class UserService {

private UserRepository repository;

public UserService(UserRepository repository) {

this.repository = repository;

}

public void registerUser(String id, String name) throws IOException {

User user = new User(id, name);

repository.save(user);

}

public List<User> getAllUsers() throws IOException {

return repository.findAll();

}

}

---

4. IntegrationTest.java (INTEGRATION TEST)

import java.io.IOException;

import java.util.List;

public class IntegrationTest {

public static void main(String[] args) throws IOException {

// Setup

String testFilePath = "test\_users.txt";

UserRepository repo = new UserRepository(testFilePath);

repo.clear(); // make sure the file is empty before testing

UserService service = new UserService(repo);

// Test

service.registerUser("1", "ABC");

service.registerUser("2", "DEF");

List<User> users = service.getAllUsers();

assert users.size() == 2 : "Expected 2 users";

assert users.get(0).getName().equals("ABC");

assert users.get(1).getName().equals("DEF");

System.out.println("Integration test passed!");

// Clean up

repo.clear();

}

}

---

How to Run

Compile and run with:

javac \*.java

java IntegrationTest

**Project – 3**

Design a system where students can register for courses. It should manage students and courses using ArrayList, ensure a student can't register for the same course twice, and throw exceptions for invalid operations. Use interfaces and abstract classes to organize the structure.

**Requirements:**

1. **Student and Course classes** – store data.
2. **RegistrationManager** – handles registrations.
3. **Interface (Registrable)** – defines methods like registerCourse(), dropCourse().
4. **Abstract Class (Person)** – base class for Student.
5. **Custom Exceptions**:
   * CourseAlreadyRegisteredException
   * CourseNotFoundException

**MCQ – Test Plan and Design**

**What is the primary objective of test planning?**

A. To write code  
B. To ensure test cases pass  
C. **To define the scope, approach, resources, and schedule of testing**D. To create user manuals

**2. Who is typically responsible for creating a test plan?**

A. Developer  
B. End-user  
C**.** Project manager  
D. **Test lead or test manager**

**3. Which of the following is NOT typically included in a test plan?**

A. Test objectives  
B. Test environment details  
C. **Marketing strategies**  
D. Resource planning

**4. What is a test case?**

A. A description of how to fix a bug  
B. **A set of conditions or inputs used to test a feature**  
C. A report of test results  
D. A program written for testing

**5. What is the importance of test design techniques?**

A. To create documentation  
B. To increase budget  
C. **To derive effective test cases**D. To avoid user feedback

**6. Which document defines what to test, how to test, and when to test?**

A. Test case  
B. Test report  
C. **Test plan**  
D. Test log

**7. Equivalence Partitioning and Boundary Value Analysis are types of:**

A. Static testing  
B. Test reporting techniques  
C. **Black-box test design techniques**  
D. White-box test design techniques

**8. In test planning, what does "entry criteria" mean?**

A. **Conditions under which testing stops**  
B. Metrics collected during testing  
C. Conditions that must be met to begin testing  
D. Test closure criteria

**9. Which is NOT a common test planning activity?**

A. Identifying test deliverables  
B. **Designing architecture**  
C. Defining test scope  
D. Scheduling testing tasks

**10. Which of the following is considered a test deliverable?**

A. Software requirements document  
B. Project proposal  
C. **Test summary report**  
D. Marketing analysis

**11. What is "test environment" in test planning?**

A. Location of testers  
B. **A controlled setting** **to execute tests**  
C. Project budget  
D. Database schema

**12. Which test design technique focuses on testing the paths in the code?**

A. Equivalence Partitioning  
B. Decision Table  
C. **Statement Coverage**  
D. Boundary Value Analysis

**13. What is risk-based testing?**

A. Testing without documentation  
B. **Testing focused on high-risk areas**  
C. Testing all features equally  
D. Ignoring minor bugs

**14. Which one of these is not a black-box test design technique?**

A. Decision Table Testing  
B. State Transition Testing  
C. Boundary Value Analysis  
D. **Control Flow Testing**

**15. Test schedule in the test plan helps to:**

A. Avoid testing  
B. Determine who writes the code  
C. **Track testing progress and deadlines**D. Track user feedback

**16. What is the purpose of a test summary report?**

A. To define the business model  
B. To evaluate the performance of developers  
C. **To summarize testing activities and results**  
D. To list all bugs

**17. Which phase comes immediately after test design?**

A. Test closure  
B. **Test execution**  
C. Test planning  
D. Requirement analysis

**18. What is the use of a Requirement Traceability Matrix (RTM) in test planning?**

A. To track team performance  
B**. To link test cases with requirements**  
C. To describe test environment  
D. To define coding standards

**19. Which one is NOT a purpose of test design?**

A. Deriving test cases  
B. Identifying test data  
C. **Estimating test effort**  
D. Identifying test conditions

**20. Which of the following is a component of test design specification?**

A. Budget plan  
B**. Test items and features to be tested**  
C. Marketing strategy  
D. Maintenance schedule

**MCQ – Test Execution**

**What is test execution in software testing?**

A. Writing test plans  
B. Running test cases and recording results  
C. Designing test cases  
D. Writing code for features

**2. Which of the following activities is part of test execution?**

A. Creating test plan  
B. Writing user stories  
C. Executing test cases  
D. Gathering requirements

**3. Before test execution begins, what should be ready?**

A. Test plan only  
B. Test report  
C. Test cases, test data, and test environment  
D. Code documentation

**4. What is the result of a test execution?**

A. A defect log  
B. Test plan  
C. Project scope  
D. Coding manual

**5. What is the main goal of test execution?**

A. To estimate project budget  
B. To validate software works as expected  
C. To improve UI design  
D. To assign team members

**6. During test execution, what happens when a test fails?**

A. It is ignored  
B. It is skipped  
C. It is logged as a defect or bug  
D. The project is canceled

**7. What is test log in the context of test execution?**

A. Log of user feedback  
B. Log of source code  
C. A record of test execution details  
D. List of team members

**8. Who typically executes test cases during test execution?**

A. Developers  
B. Business analysts  
C. Testers or QA engineers  
D. Customers

**9. Which of the following tools is commonly used for test execution management?**

A. MS Word  
B. Selenium IDE  
C. JIRA/Xray  
D. GitHub

**10. What is a blocked test case?**

A. A test case with high priority  
B. A test case with no expected result  
C. A test case that cannot be executed due to an unresolved issue  
D. A test case that passed

**11. What does a 'Pass' result indicate in test execution?**

A. The requirement has changed  
B. The test case was skipped  
C. The software behaved as expected  
D. There is a bug

**12. What happens if a defect is found during test execution?**

A. It is ignored  
B. It is fixed immediately and the test continues  
C. It is reported to the defect tracking system  
D. The tester modifies the requirement

**13. Which document is updated during test execution?**

A. Requirement document  
B. User guide  
C. Test case execution report or log  
D. Gantt chart

**14. Test execution can be done:**

A. Only manually  
B. Only by developers  
C. Both manually and using automation tools  
D. Only at the end of the project

**15. What does "Actual Result" refer to in test execution?**

A. What the developer expected  
B. The output observed during execution  
C. The original requirement  
D. The previous test result

**16. A test case is considered completed when:**

A. It is skipped  
B. It is not assigned  
C. It has been executed and result (Pass/Fail/Blocked) is recorded  
D. The project is completed

**17. What is a test incident report?**

A. A report for budgeting  
B. A report of encountered problems during testing  
C. A list of new features  
D. A training document

**18. When is re-testing performed during test execution?**

A. When the requirement changes  
B. After a test passes  
C. After a defect is fixed  
D. When a test is blocked

**19. What is the difference between re-testing and regression testing?**

A. Re-testing checks new features; regression checks old ones  
B. Re-testing verifies bug fixes; regression ensures no new bugs  
C. They are the same  
D. Regression is only manual

**20. What is test data in test execution?**

A. Data used by developers  
B. Production data  
C. Input values used in test cases  
D. User credentials

**MCQ – Defect Management in Software Testing**

**What is a defect in software testing?**

A. A change request  
B. A mismatch between expected and actual results  
C. A feature enhancement  
D. A user complaint

**2. What is the main goal of defect management?**

A. To increase code complexity  
B. To manage user expectations  
C. To detect, document, track, and resolve defects  
D. To delete bugs quickly

**3. Which phase involves fixing the defect?**

A. Test execution  
B. Defect logging  
C. Defect resolution  
D. Test planning

**4. What is the first step in defect management?**

A. Defect fixing  
B. Defect closure  
C. Defect detection  
D. Defect validation

**5. Which of the following is NOT a typical status of a defect?**

A. Open  
B. In Review  
C. Resolved  
D. Promoted

**6. Who is responsible for logging a defect?**

A. Project Manager  
B. Developer  
C. Tester  
D. Client

**7. What is a defect lifecycle?**

A. A software development model  
B. The sequence of states a defect goes through  
C. A backup plan  
D. A testing tool

**8. Which tool is commonly used for defect tracking?**

A. Microsoft Excel  
B. JIRA  
C. Slack  
D. Notepad

**9. What does defect severity refer to?**

A. The likelihood of a defect occurring  
B. The number of users affected  
C. The impact of the defect on the system  
D. How often the defect appears

**10. What does defect priority indicate?**

A. The type of defect  
B. How quickly the defect should be fixed  
C. The status of the defect  
D. Who raised the defect

**11. Who usually sets the priority of a defect?**

A. Tester  
B. Developer  
C. Project manager or product owner  
D. End user

**12. A defect is marked as "Closed" when:**

A. It is detected  
B. It is assigned to a developer  
C. It is verified as fixed by the tester  
D. The project ends

**13. What does "Reopen" status mean in defect management?**

A. The defect was wrongly closed  
B. The defect is not reproducible  
C. The defect is invalid  
D. The defect was deleted

**14. What is a “Duplicate” defect?**

A. A defect that occurs more than once  
B. A defect with the same name  
C. A defect already reported by someone else  
D. A defect without severity

**15. What is the difference between a bug and a defect?**

A. Bugs are in hardware; defects in software  
B. There is no difference; the terms are often used interchangeably  
C. Defects are found by users only  
D. Bugs are always critical

**16. Which of the following is NOT a typical field in a defect report?**

A. Steps to reproduce  
B. Expected result  
C. Root cause analysis  
D. Marketing plan

**17. What is the purpose of a defect triage meeting?**

A. To discuss marketing strategies  
B. To plan the next sprint  
C. To prioritize and assign defects  
D. To develop new features

**18. Which of the following best describes “Regression Defect”?**

A. A new defect in a new feature  
B. A defect introduced by fixing another  
C. A critical defect  
D. A non-reproducible defect

**19. When should a defect be deferred?**

A. When it is low priority and planned for a future release  
B. When the developer is unavailable  
C. When the build fails  
D. When the test case is incorrect

**20. What is the role of root cause analysis in defect management?**

A. To assign defects to testers  
B. To identify the source of the defect  
C. To automate test cases  
D. To document user feedback

**MCQ - JDBC**

1. **What does JDBC stand for?**  
   A) Java Distributed Connection  
   B) Java Database Connectivity  
   C) Java Data Communication  
   D) Java Dynamic Connectivity
2. **Which package contains JDBC classes and interfaces?**  
   A) java.jdbc  
   B) javax.sql  
   C) java.sql  
   D) org.sql
3. **Which interface is used to establish a connection with the database?**  
   A) Statement  
   B) DriverManager  
   C) Connection  
   D) ResultSet
4. **Which method is used to execute a SQL SELECT query?**  
   A) execute()  
   B) executeUpdate()  
   C) executeQuery()  
   D) runQuery()
5. **Which object is returned by the executeQuery() method?**  
   A) Connection  
   B) Statement  
   C) ResultSet  
   D) Result
6. **Which method is used to load a JDBC driver?**  
   A) Class.forName()  
   B) Driver.load()  
   C) DriverManager.loadDriver()  
   D) Connection.loadDriver()
7. **Which interface is used to execute SQL statements?**  
   A) Connection  
   B) Statement  
   C) ResultSet  
   D) PreparedStatement
8. **Which method is used to close a JDBC connection?**  
   A) end()  
   B) finish()  
   C) close()  
   D) disconnect()
9. **What does executeUpdate() return?**  
   A) A boolean value  
   B) A ResultSet  
   C) Number of rows affected  
   D) A SQL string
10. **Which type of driver is a thin driver?**  
    A) Type-1  
    B) Type-2  
    C) Type-3  
    D) Type-4
11. **Which of the following is not a valid JDBC driver type?**  
    A) Type-1  
    B) Type-2  
    C) Type-5  
    D) Type-4
12. **Which interface allows you to run a precompiled SQL statement?**  
    A) Statement  
    B) CallableStatement  
    C) PreparedStatement  
    D) SQLStatement
13. **Which object is used to call stored procedures?**  
    A) Statement  
    B) CallableStatement  
    C) PreparedStatement  
    D) ProcedureCall
14. **How do you prevent SQL injection in JDBC?**  
    A) Use Statement  
    B) Use PreparedStatement  
    C) Use dynamic SQL  
    D) Use ResultSet
15. **Which method of ResultSet moves the cursor to the next row?**  
    A) next()  
    B) previous()  
    C) first()  
    D) moveNext()

1. **What is the default ResultSet type?**  
   A) TYPE\_SCROLL\_INSENSITIVE  
   B) TYPE\_FORWARD\_ONLY  
   C) TYPE\_SCROLL\_SENSITIVE  
   D) TYPE\_DYNAMIC
2. **Which method is used to commit a transaction?**  
   A) transaction.commit()  
   B) commit()  
   C) Connection.commit()  
   D) Statement.commit()
3. **Which of the following disables auto-commit in JDBC?**  
   A) conn.disableAutoCommit()  
   B) conn.setAutoCommit(false)  
   C) conn.autoCommit(false)  
   D) conn.transaction(false)
4. **Which exception is thrown by JDBC methods?**  
   A) SQLException  
   B) JDBCException  
   C) IOException  
   D) DataAccessException
5. **Which interface is used to fetch metadata about the database?**  
   A) DatabaseInfo  
   B) ResultSet  
   C) DatabaseMetaData  
   D) DataInspector
6. **Which interface allows accessing metadata of a ResultSet?**  
   A) ResultMetaData  
   B) ResultSetMetaData  
   C) MetaData  
   D) DataResultMeta
7. **Which method is used to rollback a transaction?**  
   A) rollback()  
   B) cancel()  
   C) undo()  
   D) reverse()
8. **Which method of PreparedStatement is used to bind a string?**  
   A) setValue()  
   B) setText()  
   C) setString()  
   D) bindString()
9. **Which method returns the number of columns in a ResultSet?**  
   A) getColumnCount()  
   B) columnCount()  
   C) getCount()  
   D) getColumnNumber()
10. **Which method checks if the ResultSet is positioned after the last row?**  
    A) isLast()  
    B) afterLast()  
    C) isAfterLast()  
    D) last()
11. **What is the use of batch processing in JDBC?**  
    A) Improves UI performance  
    B) Reduces DB size  
    C) Reduces number of DB hits  
    D) None of the above
12. **Which method adds a query to the batch?**  
    A) addQuery()  
    B) executeBatch()  
    C) addBatch()  
    D) queueQuery()
13. **Which ResultSet type reflects changes made by others while scrolling?**  
    A) TYPE\_FORWARD\_ONLY  
    B) TYPE\_SCROLL\_SENSITIVE  
    C) TYPE\_SCROLL\_INSENSITIVE  
    D) TYPE\_DYNAMIC
14. **What is the return type of execute() method?**  
    A) boolean  
    B) ResultSet  
    C) int  
    D) void
15. **What does ResultSet.getInt("id") do?**  
    A) Returns a double  
    B) Returns an int from column "id"  
    C) Deletes the row  
    D) Returns row number

1. **Which method is best for executing DDL (like CREATE TABLE)?**  
   A) executeQuery()  
   B) executeUpdate()  
   C) execute()  
   D) runSQL()
2. **Which statement is used for dynamic queries with parameters?**  
   A) Statement  
   B) PreparedStatement  
   C) CallableStatement  
   D) StaticStatement
3. **To call a stored function that returns a value, you use:**  
   A) Statement  
   B) CallableStatement  
   C) PreparedStatement  
   D) None
4. **Which of the following ensures a valid JDBC driver is registered?**  
   A) DriverManager.loadDriver()  
   B) Class.forName("com.mysql.jdbc.Driver")  
   C) new Driver()  
   D) Connection.register()
5. **What will ResultSet.getString(1) return?**  
   A) First column value as a String  
   B) Row count  
   C) Column name  
   D) Error
6. **Which of the following drivers uses ODBC?**  
   A) Type 1  
   B) Type 2  
   C) Type 3  
   D) Type 4
7. **Which JDBC driver type is platform-independent and written in Java?**  
   A) Type 1  
   B) Type 2  
   C) Type 3  
   D) Type 4
8. **In JDBC, which of the following provides connection pooling?**  
   A) DriverManager  
   B) DataSource  
   C) Connection  
   D) ResultSet

1. **Which is more efficient for repeated SQL execution?**  
   A) Statement  
   B) PreparedStatement  
   C) ResultSet  
   D) Connection
2. **Which interface supports scrollable and updatable result sets?**  
   A) ResultSet  
   B) Statement  
   C) PreparedStatement  
   D) ScrollResultSet
3. **JDBC was developed by:**  
   A) Oracle  
   B) Sun Microsystems  
   C) Microsoft  
   D) IBM
4. **To execute multiple SQL queries at once, use:**  
   A) execute()  
   B) executeQuery()  
   C) executeBatch()  
   D) multiExecute()
5. **To get the generated primary key after insert:**  
   A) getGeneratedKey()  
   B) getAutoKey()  
   C) getKey()  
   D) getGeneratedKeys()
6. **JDBC API supports:**  
   A) Only MySQL  
   B) Only Oracle  
   C) Any relational DB with driver  
   D) Only JavaDB
7. **Which ResultSet type cannot move backward?**  
   A) TYPE\_SCROLL\_INSENSITIVE  
   B) TYPE\_SCROLL\_SENSITIVE  
   C) TYPE\_FORWARD\_ONLY  
   D) TYPE\_SCROLLABLE

1. **Which statement is false about JDBC?**  
   A) JDBC connects Java to DB  
   B) JDBC can call stored procedures  
   C) JDBC does not support transactions  
   D) JDBC can use PreparedStatements
2. **What is the first step in JDBC?**  
   A) Create Statement  
   B) Load Driver  
   C) Execute Query  
   D) Close Connection
3. **Which is used to hold the results of SQL query?**  
   A) Statement  
   B) ResultSet  
   C) Connection  
   D) DataSet
4. **Which method of ResultSet checks for NULL?**  
   A) isNull()  
   B) wasNull()  
   C) checkNull()  
   D) hasNull()
5. **Which object in JDBC manages the set of drivers?**  
   A) Driver  
   B) DriverManager  
   C) Connection  
   D) Statement
6. **MCQ – File Handling**
7. **1. Which class is used to create a file in Java?**
8. a) FileOutputStream  
   b) FileReader  
   c) File  
   d) FileWriter
9. **2. What package contains the File class?**
10. a) java.io  
    b) java.util  
    c) java.file  
    d) java.lang
11. **3. What does the createNewFile() method of the File class return if the file already exists?**
12. a) true  
    b) false  
    c) null  
    d) Throws Exception
13. **4. Which method is used to write data to a file using FileWriter?**
14. a) put()  
    b) write()  
    c) append()  
    d) output()
15. **5. Which exception must be handled when performing file operations in Java?**
16. a) IOException  
    b) FileNotFoundException  
    c) RuntimeException  
    d) Both a and b
17. **6. What is the use of FileReader in Java?**
18. a) Write to a file  
    b) Delete a file  
    c) Read characters from a file  
    d) Rename a file
19. **7. Which method of File class is used to delete a file?**
20. a) remove()  
    b) deleteFile()  
    c) delete()  
    d) erase()
21. **8. What does BufferedReader improve?**
22. a) File writing speed  
    b) File structure  
    c) Reading performance  
    d) File formatting
23. **9. What will File.exists() return if the file does not exist?**
24. a) true  
    b) false  
    c) null  
    d) Exception
25. **10. Which class is used for writing primitive data types to a file?**
26. a) FileWriter  
    b) FileOutputStream  
    c) DataOutputStream  
    d) ObjectOutputStream
27. **11. Which class is used for reading serialized objects from a file?**
28. a) ObjectReader  
    b) DataInputStream  
    c) ObjectInputStream  
    d) FileInputStream
29. **12. Which of the following is used to read a line of text from a file?**
30. a) FileReader.read()  
    b) BufferedReader.readLine()  
    c) Scanner.nextFile()  
    d) FileInputStream.readLine()
31. **13. Which method can be used to create a directory using the File class?**
32. a) makeDirectory()  
    b) mkdir()  
    c) createDir()  
    d) newDirectory()
33. **14. What is the superclass of all classes handling file input and output?**
34. a) File  
    b) Reader  
    c) InputStream  
    d) Object
35. **15. Which class is used to read byte data from a file?**
36. a) FileReader  
    b) FileInputStream  
    c) BufferedReader  
    d) Scanner
37. **16. What does the flush() method in FileWriter or BufferedWriter do?**
38. a) Deletes file  
    b) Flushes the stream and forces any buffered output to be written  
    c) Reads the file  
    d) Encrypts data
39. **17. How can you check if a file is writable using File class?**
40. a) isWritable()  
    b) canWrite()  
    c) writeable()  
    d) isWritableFile()
41. **18. What does File.list() return?**
42. a) List of file sizes  
    b) Array of file names as strings  
    c) List of file objects  
    d) Nothing
43. **19. Which class is preferred for writing character data to a file?**
44. a) FileWriter  
    b) FileOutputStream  
    c) DataOutputStream  
    d) ObjectOutputStream
45. **20. Which method is used to close a file after use?**
46. a) terminate()  
    b) delete()  
    c) close()  
    d) end()