1. Write a JUnit program to demonstrate lifecycle methods.

```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class LifecycleExampleTest {
    @BeforeClass
    public static void setUpClass() {
        System.out.println("BeforeClass: Setting up class-level resources");
    }
      @AfterClass
    public static void tearDownClass() {
        System.out.println("AfterClass: Cleaning up class-level resources");
    }
        @Before
    public void setUp() {
        System.out.println("Before: Setting up test-level resources");
    }
      @After
    public void tearDown() {
        System.out.println("After: Cleaning up test-level resources");
    }
    @Test
    public void testAddition() {
```

```
        System.out.println("Test: Performing addition test");
        assertEquals(4, 2 + 2);
    }
    @Test
    public void testSubtraction() {
        System.out.println("Test: Performing subtraction test");
        assertEquals(2, 4 - 2);
    }
}
```

2. Write a JUnit program to demonstrate assertions.

```
import org.junit.Test;
import static org.junit.Assert.*;
public class AssertionExampleTest {
@Test
 public void testAssertEquals() {
assertEquals("String comparison", "Hello", "Hello");
assertEquals("Integer comparison", 2, 1 + 1);
}
   @Test
public void testAssertNotEquals() {
  assertNotEquals("String comparison", "Hello", "World");
 assertNotEquals("Integer comparison", 3, 1 + 1);
  }
@Test
 public void testAssertTrueFalse() {
assertTrue("True assertion", 10 > 5);
```

```
    assertFalse("False assertion", 4 < 2);
   }
   @Test
   public void testAssertNullNotNull() {
   String nullString = null;
    assertNull("Should be null", nullString);
   String notNullString = "Hello";
    assertNotNull("Should not be null", notNullString);
    }
    @Test
    public void testAssertArrayEquals() {
      int[] expectedArray = {1, 2, 3};
      int[] actualArray = {1, 2, 3};

    assertArrayEquals("Array comparison", expectedArray, actualArray);
  }
}
```

3. Write a JUnit program to demonstrate how tests are disabled.

```
import org.junit.Test;
import org.junit.Ignore;
import static org.junit.Assert.*;
public class DisabledTestExample {
 @Test
   public void testEnabled() {
      System.out.println("This test is enabled and should run.");
      assertTrue(true);
```

```
    }
    @Ignore
    @Test
    public void testDisabled() {
        System.out.println("This test is disabled and should not run.");
        assertTrue(false);
disabled
    }
    @Test
    @Ignore("Reason for disabling this test")
    public void testDisabledWithReason() {
        System.out.println("This test is disabled with a reason and should not run.");
        assertTrue(false);    }

    @Test
    @Disabled
    public void testDisabledJUnit5() {
        System.out.println("This test is disabled in JUnit 5 and should not run.");
        assertTrue(false); disabled
    }
}
```

4. Write a JUnit program to demonstrate assumptions in JUnit.

```
import org.junit.Test;
import static org.junit.Assume.*;
```

```
public class AssumptionExampleTest {
   @Test
   public void testAssumptionPass() {
      assumeTrue("Assumption is true, so the test should run", true);
      System.out.println("Test passed!");
   }
   @Test
   public void testAssumptionFail() {
      assumeTrue("Assumption is false, so the test should be skipped", false);
      System.out.println("This line won't be executed because the assumption
failed.");
   }
   @Test
   public void testAssumptionWithMessage() {
      String environment = System.getProperty("env");
      assumeNotNull("Assumption is based on a system property", environment);
      System.out.println("Test passed with environment: " + environment);
   }
   @Test
   public void testAssumptionWithMultipleConditions() {
      String environment = System.getProperty("env");
      assumeTrue("Assumption is based on multiple conditions",
            environment != null && environment.equals("test"));
      System.out.println("Test passed with environment: " + environment);
    }
}
```

5. Write a JUnit program to demonstrate test interfaces and default methods in JUnit.

```
import org.junit.jupiter.api.Test;
interface MathOperationsTest {

    @Test
    default void testAddition() {
        System.out.println("Testing addition");
        int result = add(2, 3);
        assert result == 5 : "Addition failed!";
    }

    @Test
    default void testSubtraction() {
        System.out.println("Testing subtraction");
        int result = subtract(5, 3);
        assert result == 2 : "Subtraction failed!";
    }
    int add(int a, int b);

    int subtract(int a, int b);
}
class MathOperations implements MathOperationsTest {

    @Override
    public int add(int a, int b) {
```

```
        return a + b;
    }
    @Override
    public int subtract(int a, int b) {
        return a - b;
    }
}
class AnotherMathOperations implements MathOperationsTest {
    @Override
    public int add(int a, int b) {
        return b - a; // Incorrect implementation for testing purposes
    }

    @Override
    public int subtract(int a, int b) {
        return a * b; // Incorrect implementation for testing purposes
    }
}
public class TestInterfaceExample {
    public static void main(String[] args) {
        System.out.println("Running tests for MathOperations");
        MathOperations mathOperations = new MathOperations();
        mathOperations.testAddition();
        mathOperations.testSubtraction();
         System.out.println("\nRunning tests for AnotherMathOperations");
        AnotherMathOperations anotherMathOperations = new
AnotherMathOperations();
```

```
        anotherMathOperations.testAddition();
        anotherMathOperations.testSubtraction();
    }
}
```

6. Write a JUnit program to demonstrate how tests are repeated in JUnit.

```
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.RepetitionInfo;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class RepeatedTestExample {
    @RepeatedTest(3)
    void testAddition(RepetitionInfo repetitionInfo) {
        int result = add(2, 3);
        assertEquals(5, result, "Addition failed in repetition " +
repetitionInfo.getCurrentRepetition());
    }
    @RepeatedTest(value = 5, name = "{displayName} - Repetition
{currentRepetition}/{totalRepetitions}")
    void testSubtraction() {
        int result = subtract(5, 3);
        assertEquals(2, result, "Subtraction failed");
    }
    int add(int a, int b) {
        return a + b;
    }
    int subtract(int a, int b) {
```

```
        return a - b;
    }
}
```

7. Write a JUnit program to demonstrate how dynamic tests are created in JUnit.

```
import org.junit.jupiter.api.DynamicContainer;
import org.junit.jupiter.api.DynamicTest;
import org.junit.jupiter.api.TestFactory;
import java.util.Arrays;
import java.util.Collection;
import java.util.stream.Stream;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class DynamicTestExample {
    @TestFactory
    Collection<DynamicTest> dynamicTestsFromCollection() {
        return Arrays.asList(
                DynamicTest.dynamicTest("Addition Test", () -> assertEquals(5, add(2,
3))),
                DynamicTest.dynamicTest("Subtraction Test", () -> assertEquals(2,
subtract(5, 3)))
        );
    }

    @TestFactory
    Stream<DynamicTest> dynamicTestsFromStream() {
```

```
        return Stream.of(
                DynamicTest.dynamicTest("Multiplication Test", () -> assertEquals(12,
multiply(4, 3))),
                DynamicTest.dynamicTest("Division Test", () -> assertEquals(4,
divide(12, 3)))
        );
    }

    @TestFactory
    Stream<DynamicContainer> dynamicContainer() {
        return Stream.of(
                DynamicContainer.dynamicContainer("Math Operations",
                        Stream.of(
                                DynamicTest.dynamicTest("Addition Test", () -> assertEquals(7,
add(4, 3))),
                                DynamicTest.dynamicTest("Subtraction Test", () ->
assertEquals(2, subtract(5, 3)))
                        )),
                DynamicContainer.dynamicContainer("More Math Operations",
                        Stream.of(
                                DynamicTest.dynamicTest("Multiplication Test", () ->
assertEquals(20, multiply(4, 5))),
                                DynamicTest.dynamicTest("Division Test", () -> assertEquals(3,
divide(15, 5)))
                        ))
        );
    }
    int add(int a, int b) {
        return a + b;
```

```
    }
    int subtract(int a, int b) {
        return a - b;
    }
    int multiply(int a, int b) {
        return a * b;
    }
    int divide(int a, int b) {
        return a / b;
    }
}
```

8. Write a JUnit program to demonstrate how parameterized tests are created in JUnit.

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class ParameterizedTestExample {
    @ParameterizedTest
    @CsvSource({ "2, 3, 5", "5, 3, 8", "0, 0, 0", "-5, 5, 0" })
    void testAddition(int a, int b, int expectedSum) {
        int result = add(a, b);
        assertEquals(expectedSum, result, () -> "Sum of " + a + " and " + b + " should
be " + expectedSum);
    }
```

```
    @ParameterizedTest
    @CsvSource({ "4, 2, 2", "9, 3, 6", "0, 5, -5", "-8, 4, -12" })
    void testSubtraction(int a, int b, int expectedDifference) {
        int result = subtract(a, b);
        assertEquals(expectedDifference, result, () -> "Difference of " + a + " and " + b
+ " should be " + expectedDifference);
    }

    @Test
    void regularTest() {
            assertEquals(6, add(2, 4));
    }
    int add(int a, int b) {
        return a + b;
    }
    int subtract(int a, int b) {
        return a - b;
}
```

9. Write a JUnit program to demonstrate how argument sources are used in in JUnit.

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.MethodSource;
```

```
import java.util.stream.Stream;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class ArgumentSourceExample {
   @ParameterizedTest
   @CsvSource({ "2, 3, 5", "5, 3, 8", "0, 0, 0", "-5, 5, 0" })
   void testAddition(int a, int b, int expectedSum) {
      int result = add(a, b);
      assertEquals(expectedSum, result, () -> "Sum of " + a + " and " + b + " should
be " + expectedSum);
   }

   @ParameterizedTest
   @MethodSource("subtractArguments")
   void testSubtraction(int a, int b, int expectedDifference) {
      int result = subtract(a, b);
      assertEquals(expectedDifference, result, () -> "Difference of " + a + " and " +
b + " should be " + expectedDifference);
   }
   static Stream<Arguments> subtractArguments() {
      return Stream.of(
            Arguments.of(4, 2, 2),
            Arguments.of(9, 3, 6),
            Arguments.of(0, 5, -5),
            Arguments.of(-8, 4, -12)
      );
   }
   int add(int a, int b) {
      return a + b;
```

```
    }
    int subtract(int a, int b) {
        return a - b;
    }
}
```

10. Write a JUnit program to demonstrate argument conversion in JUnit.

```
import org.junit.jupiter.api.extension.ParameterContext;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.converter.ArgumentConversionException;
import org.junit.jupiter.params.converter.ConvertWith;
import org.junit.jupiter.params.converter.SimpleArgumentConverter;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.MethodSource;
import java.util.stream.Stream;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class ArgumentConversionExample {
  @ParameterizedTest
 @CsvSource({ "2, 3, 5", "5, 3, 8", "0, 0, 0", "-5, 5, 0" })
void testAddition(int a, int b, int expectedSum) {
int result = add(a, b);
        assertEquals(expectedSum, result, () -> "Sum of " + a + " and " + b + "
should be " + expectedSum);
}
```

```
    @ParameterizedTest
    @MethodSource("subtractArguments")
    void testSubtraction(@ConvertWith(MyConverter.class) CustomNumber a,
                         @ConvertWith(MyConverter.class) CustomNumber b,
                         int expectedDifference) {
        int result = subtract(a.getValue(), b.getValue());
        assertEquals(expectedDifference, result, () -> "Difference of " + a + " and " + b + "
should be " + expectedDifference);
    }
    static Stream<Arguments> subtractArguments() {
        return Stream.of(
                Arguments.of(new CustomNumber(4), new CustomNumber(2), 2),
                Arguments.of(new CustomNumber(9), new CustomNumber(3), 6),
                Arguments.of(new CustomNumber(0), new CustomNumber(5), -5),
                Arguments.of(new CustomNumber(-8), new CustomNumber(4), -12)
        );
    }
    int add(int a, int b) {
        return a + b;
    }
    int subtract(int a, int b) {
        return a - b;
    }
    static class MyConverter extends SimpleArgumentConverter {
        @Override
        protected Object convert(Object source, Class<?> targetType) throws
ArgumentConversionException {
            if (source instanceof String && targetType == CustomNumber.class) {
```

```
            return new CustomNumber(Integer.parseInt((String) source));
        }
        throw new ArgumentConversionException("Cannot convert source to target type");
    }
  }
  static class CustomNumber {
    private final int value;
    CustomNumber(int value) {
      this.value = value;
    }
    int getValue() {
      return value;
    }
    @Override
    public String toString() {
      return String.valueOf(value);
    }
  }
}
```

11 . Write a JUnit program to demonstrate extension points.

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtensionContext;
import org.junit.jupiter.api.extension.ExtensionContext.Namespace;
import org.junit.jupiter.api.extension.ExtensionContext.Store;
import org.junit.jupiter.api.extension.ExtensionContext.Store.CloseableResource;
```

```
import org.junit.jupiter.api.extension.RegisterExtension;
import org.junit.jupiter.api.extension.Extension;
import org.junit.jupiter.api.extension.ExtensionContextException;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class ExtensionExample {
    @RegisterExtension
    static MyExtension myExtension = new MyExtension();
    @Test
    void testWithExtension() {
        int result = addWithExtension(2, 3);
        assertEquals(5, result, "Addition with extension failed");
    }
    int addWithExtension(int a, int b) {
        return myExtension.add(a, b);
    }
    static class MyExtension implements Extension {
        int add(int a, int b) {
            ExtensionContext context =
ExtensionContext.getRootStore().get(Namespace.create(getClass()));
            MyCloseableResource resource =
context.getStore(Namespace.create(getClass())).getOrComputeIfAbsent(
                    MyCloseableResource.class, k -> new MyCloseableResource(),
MyCloseableResource.class);
            return resource.add(a, b);
        }
    }
    static class MyCloseableResource implements CloseableResource {
        private int count = 0;
```

```
    int add(int a, int b) {
        count++;
        System.out.println("Count: " + count);
        return a + b;
    }
    @Override
    public void close() {
        System.out.println("Closing MyCloseableResource");
    }
  }
}
```

12. Write a JUnit program to demonstrate meta-annotation.

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.DisabledIfEnvironmentVariable;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class MetaAnnotationExample {
  @Test
  @CustomEnabledOnCI
  void testOnCI() {
    assertTrue(true, "Test passed");
  }

  @Test
  @CustomEnabledOnCI
  void anotherTestOnCI() {
    assertTrue(true, "Another test passed");
  }
```

```
    @Test
    @DisabledIfEnvironmentVariable(named = "CI", matches = "true")
    void testDisabledOnCI() {
            assertTrue(true, "Test passed");
    }
    @Target(ElementType.METHOD)
    @Retention(RetentionPolicy.RUNTIME)
    @CustomEnabledOnCI
    @DisabledIfEnvironmentVariable(named = "DATABASE_URL", matches = ".*")
    public @interface CustomEnabledOnDatabase {
    }
    @Test
    @CustomEnabledOnDatabase
    void testEnabledOnDatabase() {
           assertTrue(true, "Test passed");
    }
    @Target(ElementType.METHOD)
    @Retention(RetentionPolicy.RUNTIME)
    @DisabledIfEnvironmentVariable(named = "CI", matches = "false")
    public @interface CustomEnabledOnCI {
    }
}
```

13. Write a JUnit program to demonstrate how tests are run from the console.

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

public class MyTest {

    @Test
    void testAddition() {
        int result = add(2, 3);
        assertEquals(5, result, "Addition failed");
    }

    int add(int a, int b) {
        return a + b;
    }
}
```

14. Write a JUnit program to demonstrate how tests are run on Gradle.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class MyTest {
    @Test
    void testAddition() {
        int result = add(2, 3);
        assertEquals(5, result, "Addition failed");
    }
    int add(int a, int b) {
        return a + b;
    }
}
```

15. Write a JUnit program to demonstrate how tests are run on Maven.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class MyTest {
    @Test
    void testAddition() {
        int result = add(2, 3);
        assertEquals(5, result, "Addition failed");
    }
```

```
    int add(int a, int b) {
        return a + b;
    }
}
```

16 .Write a JUnit program to demonstrate how tests with tags are included or excluded.

```
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class TaggedTestExample {
    @Test
    @Tag("slow")
    void slowTest() {
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        assertEquals(2 + 2, 4, "Slow test failed");
    }
    @Test
    @Tag("fast")
    void fastTest() {
```

```
        assertEquals(3 * 4, 12, "Fast test failed");
    }

    @Test
    @Tag("integration")
    void integrationTest() {
        assertEquals("integration", "integration", "Integration test failed");
    }

    @Test
    @Tag("unit")
    void unitTest() {
        assertEquals("unit", "unit", "Unit test failed");
    }
}
```

17. Write a JUnit program to demonstrate code coverage.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class SampleTest {
    @Test
    void testAddition() {
        Calculator calculator = new Calculator();
        int result = calculator.add(2, 3);
        assertEquals(5, result, "Addition failed");
    }
```

```
    @Test
    void testSubtraction() {
        Calculator calculator = new Calculator();
        int result = calculator.subtract(5, 3);
        assertEquals(2, result, "Subtraction failed");
    }
}
```