Assignment 1: Login Validation

Main Java Program:

```
public class LoginService {
    public boolean login(String username, String password) {
        if(username.equals("admin") && password.equals("admin123")) {
            return true;
        }
        return false;
    }
}
```

JUnit Test Class:

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

public class LoginServiceTest {

    @Test
    public void testValidLogin() {
        LoginService ls = new LoginService();
        assertTrue(ls.login("admin", "admin123"));
    }

    @Test
    public void testInvalidLogin() {
        LoginService ls = new LoginService();
        assertFalse(ls.login("user", "pass"));
    }
}
```

Output:

```
JUnit Result:
testValidLogin PASSED
testInvalidLogin PASSED
```

Assignment 2: Age Validation

Main Java Program:

```
public class AgeValidator {
    public boolean isEligible(int age) {
        if(age >= 18) {
            return true;
        }
        return false;
    }
}
```

JUnit Test Class:

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

public class AgeValidatorTest {

    @Test
    public void testEligibleAge() {
        AgeValidator av = new AgeValidator();
        assertTrue(av.isEligible(20));
    }

    @Test
    public void testNotEligibleAge() {
        AgeValidator av = new AgeValidator();
        assertFalse(av.isEligible(15));
    }
}
```

Output:

```
JUnit Result:
testEligibleAge PASSED
testNotEligibleAge PASSED
```

Assignment 3: Addition Test

Main Java Program:

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

JUnit Test Class:

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

public class CalculatorTest {

    @Test
    public void testAddition() {
        Calculator c = new Calculator();
        assertEquals(10, c.add(5, 5));
    }
}
```

Output:

```
JUnit Result:
testAddition PASSED
```

Assignment 4: Division Exception Test

Main Java Program:

```
public class Divider {
    public int divide(int a, int b) {
        return a / b;
    }
}
```

JUnit Test Class:

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

public class DividerTest {

    @Test(expected = ArithmeticException.class)
    public void testDivideByZero() {
        Divider d = new Divider();
        d.divide(10, 0);
    }
}
```

Output:

```
JUnit Result:
testDivideByZero PASSED
```

Assignment 5 – Exception Handling with finally

```
Main Program:
package com.test.unit_testing;
public class FileFinallyDemo {
public static void main(String[] args) {
try {
System.out.println("File opened");
} catch (Exception e) {
System.out.println("Exception occurred");
} finally {
System.out.println("File closed");
}
}
}
Output:
File opened
File closed
```

Assignment 6 – Nested Try Catch

```
Main Program:
package com.test.unit_testing;
public class NestedTryDemo {
public static void main(String[] args) {
try {
try {
int a = 10 / 0;
} catch (ArithmeticException e) {
System.out.println("Inner catch: ArithmeticException");
}
} catch (Exception e) {
System.out.println("Outer catch");
}
}
}
Output:
Inner catch: ArithmeticException
```

Assignment 7 – User Input Validation Using JUnit

```
Main Class:
package com.test.unit_testing;
public class AgeValidator {
public boolean validateAge(int age) {
if (age < 18) {
throw new IllegalArgumentException("Age not eligible");
}
return true;
}
}
JUnit Test:
package com.test.unit_testing;
import static org.junit.Assert.*;
import org.junit.Test;
public class AgeValidatorTest {
```

```
@Test
public void testValidAge() {
AgeValidator av = new AgeValidator();
assertTrue(av.validateAge(21));
}
@Test(expected = IllegalArgumentException.class)
public void testInvalidAge() {
AgeValidator av = new AgeValidator();
av.validateAge(15);
}
}
JUnit Output:
GREEN BAR – All tests passed
```

Assignment 8 – Custom Exception with JUnit

```
Custom Exception:
package com.test.unit_testing;
public class InvalidBalanceException extends Exception {
public InvalidBalanceException(String message) {
super(message);
}
}
Main Class:
package com.test.unit_testing;
public class BankService {
public void withdraw(int balance) throws InvalidBalanceException {
if (balance < 500) {
throw new InvalidBalanceException("Low Balance");
}
}
}
JUnit Test:
package com.test.unit_testing;
import org.junit.Test;
public class BankServiceTest {
@Test(expected = InvalidBalanceException.class)
public void testLowBalance() throws InvalidBalanceException {
BankService bs = new BankService();
bs.withdraw(100);
}
}
JUnit Output:
GREEN BAR – Test passed
```

Assignment 9 – Exception Propagation

```
Program:
package com.test.unit_testing;
public class ExceptionPropagationDemo {
public void divide() throws ArithmeticException {
int a = 10 / 0;
}
}
package com.test.unit_testing;
public class ExceptionPropagationMain {
public static void main(String[] args) {
try {
ExceptionPropagationDemo d = new ExceptionPropagationDemo();
d.divide();
} catch (ArithmeticException e) {
System.out.println("Exception propagated and handled");
}
}
}
Output:
Exception propagated and handled
```

Assignment 10 – Re-Throwing an Exception

```
Program:
package com.test.unit_testing;
public class RethrowService {
public void process() {
try {
int a = 10 / 0;
} catch (ArithmeticException e) {
System.out.println("Logging exception");
throw e;
}
}
}
package com.test.unit_testing;
public class RethrowMain {
public static void main(String[] args) {
try {
new RethrowService().process();
} catch (Exception e) {
System.out.println("Exception rethrown and caught");
}
}
}
Output:
Logging exception
Exception rethrown and caught
```

Assignment 11 – Exception Handling in Method Overriding

```
Program:
package com.test.unit_testing;
class Parent {
```

```
void show() throws ArithmeticException {
System.out.println("Parent method");
}
}
class Child extends Parent {
void show() {
System.out.println("Child method");
}
}
public class MethodOverrideExceptionDemo {
public static void main(String[] args) {
Parent p = new Child();
p.show();
}
}
Output:
Child method
```

Assignment 12 – Custom Validation Layer

```
Program:
package com.test.unit_testing;
class EmptyInputException extends Exception {
EmptyInputException(String msg) {
super(msg);
}
}
class InvalidLengthException extends Exception {
InvalidLengthException(String msg) {
super(msg);
}
}
class ValidationService {
void validate(String input) throws Exception {
if (input == null || input.isEmpty())
throw new EmptyInputException("Input is empty");
if (input.length() < 5)
throw new InvalidLengthException("Input too short");
}
}
public class ValidationMain {
public static void main(String[] args) {
try {
new ValidationService().validate("abc");
} catch (Exception e) {
System.out.println(e.getMessage());
}
}
}
Output:
Input too short
```

Assignment 13 – Generic Exception Handling

```
Program:
package com.test.unit_testing;
public class GenericExceptionDemo {
```

```
public static void main(String[] args) {
try {
int[] a = new int[2];
a[5] = 10;
} catch (Exception e) {
System.out.println("Generic exception handled");
}
}
}
Output:
Generic exception handled
```

Assignment 14 – Try with Multiple Exceptions

```
Program:
package com.test.unit_testing;
public class MultiCatchDemo {
public static void main(String[] args) {
try {
int a = 10 / 0;
} catch (ArithmeticException | NullPointerException e) {
System.out.println("Multiple exception handled");
}
}
}
Output:
Multiple exception handled
```

Assignment 15 – Finally Execution

```
Program:
package com.test.unit_testing;
public class FinallyAlwaysDemo {
public static void main(String[] args) {
try {
int a = 10 / 5;
} finally {
System.out.println("Finally block executed");
}
}
}
Output:
Finally block executed
```