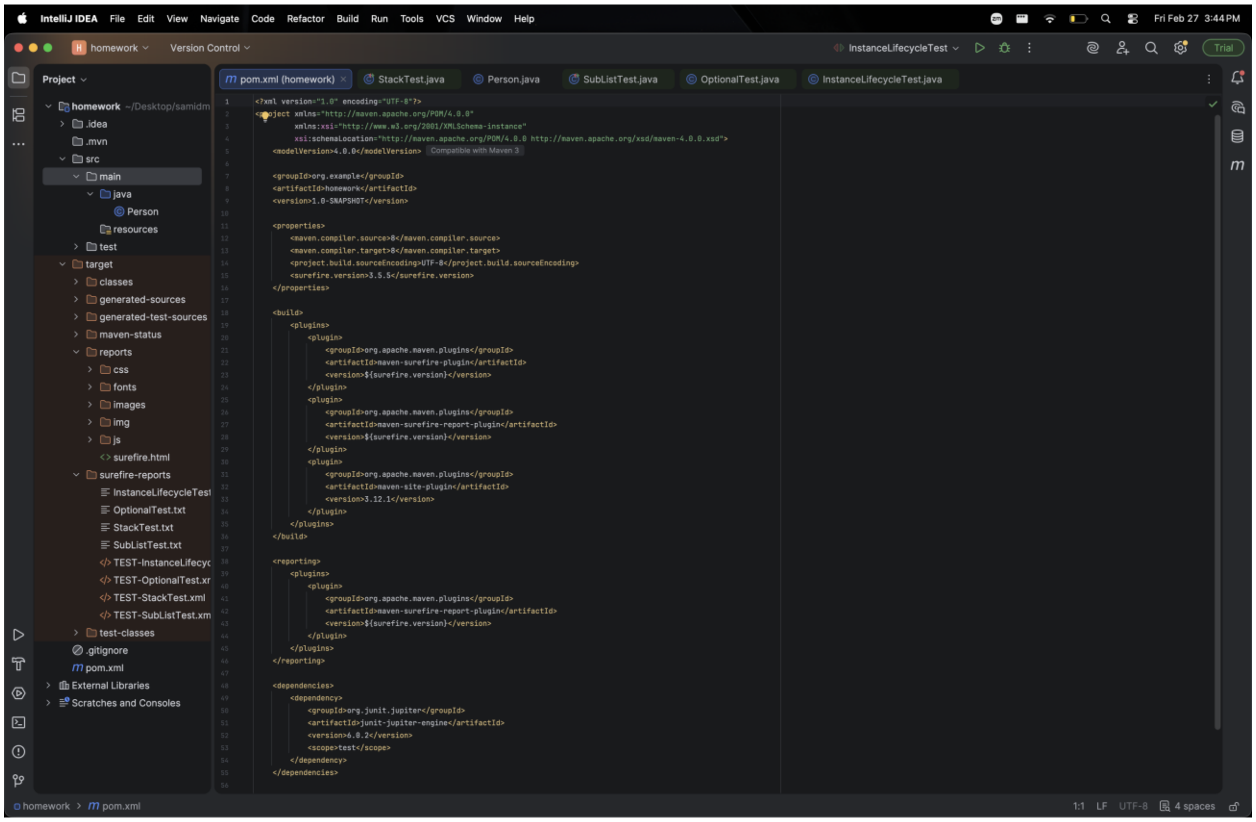
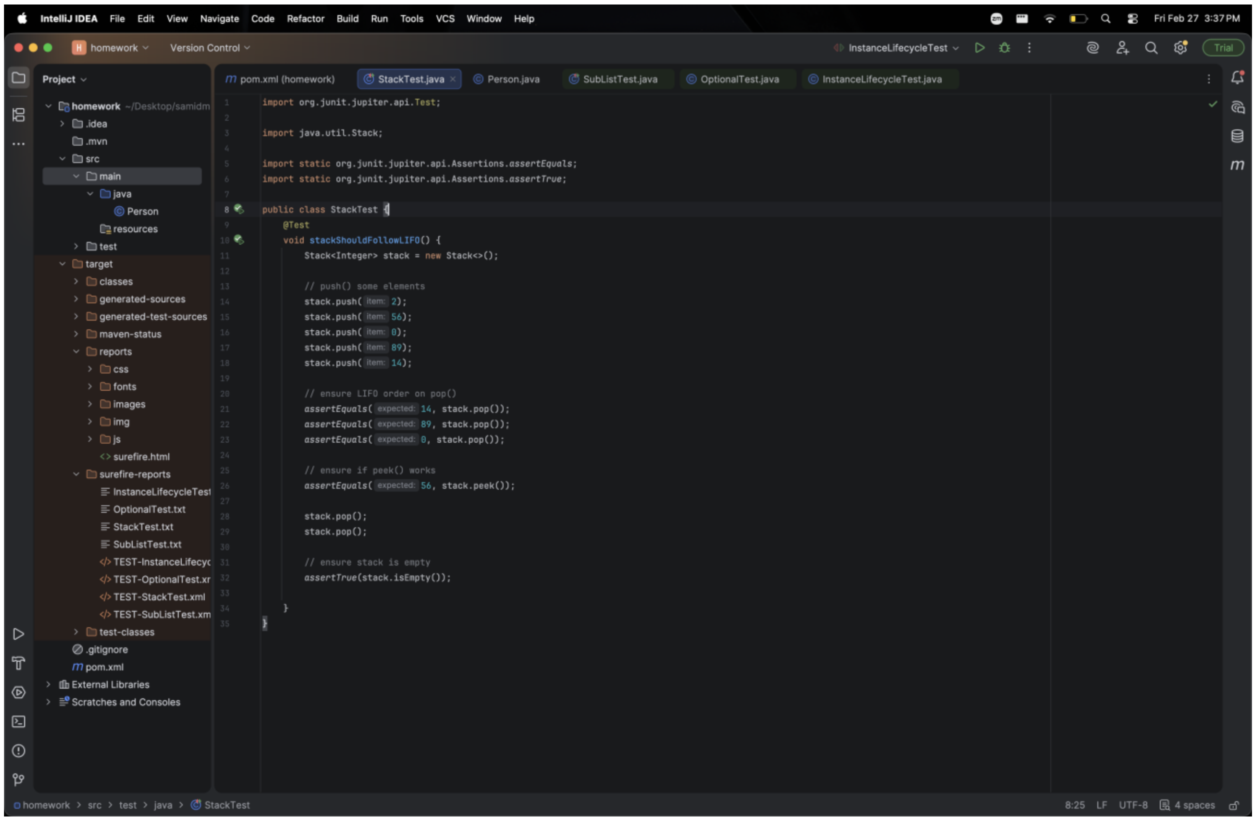
Week 2 – Day 4 Assignment (February 26th, 2026)

Test I – pom.xml



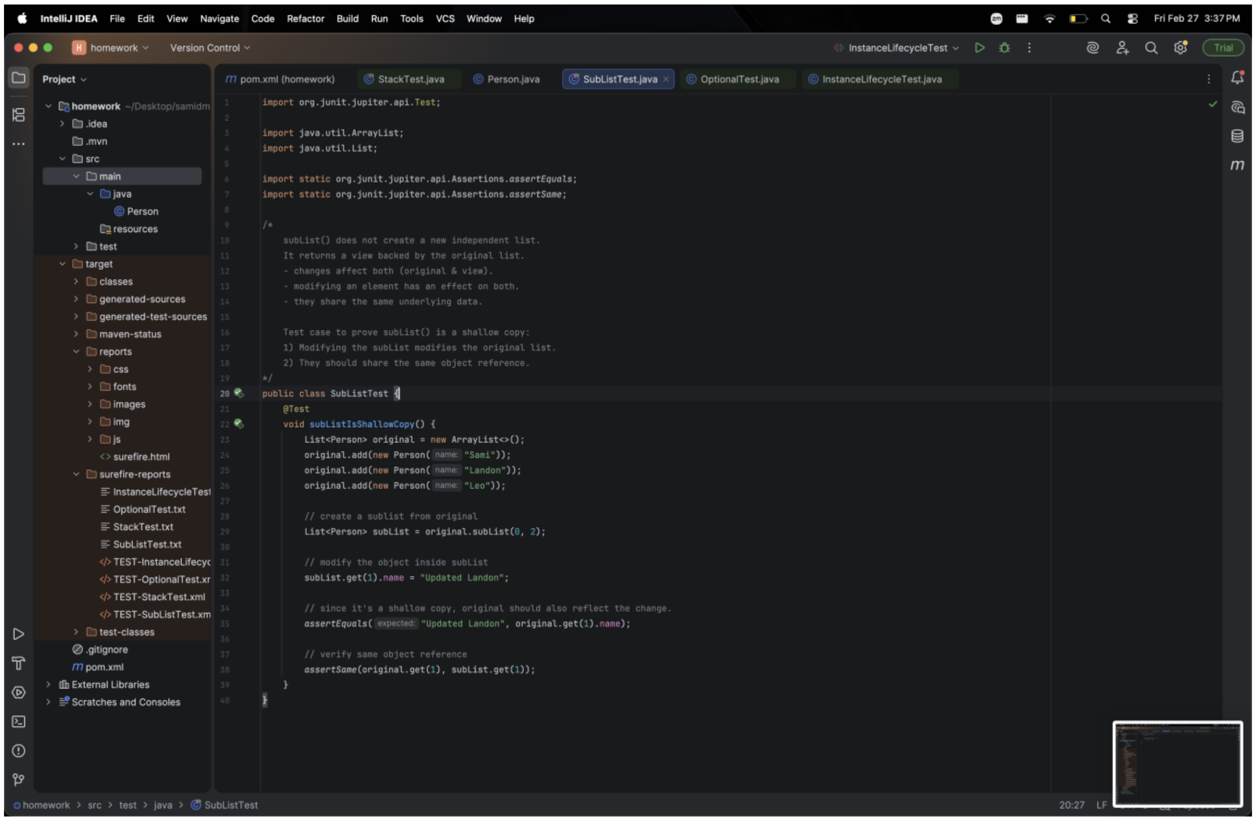
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>homework</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <surefire.version>3.5.5</surefire.version>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>${surefire.version}</version>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-report-plugin</artifactId>
                <version>${surefire.version}</version>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-site-plugin</artifactId>
                <version>3.12.1</version>
            </plugin>
        </plugins>
    </build>

    <reporting>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-report-plugin</artifactId>
                <version>${surefire.version}</version>
            </plugin>
        </plugins>
    </reporting>

    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>6.0.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

```

Test I – Stack LIFO

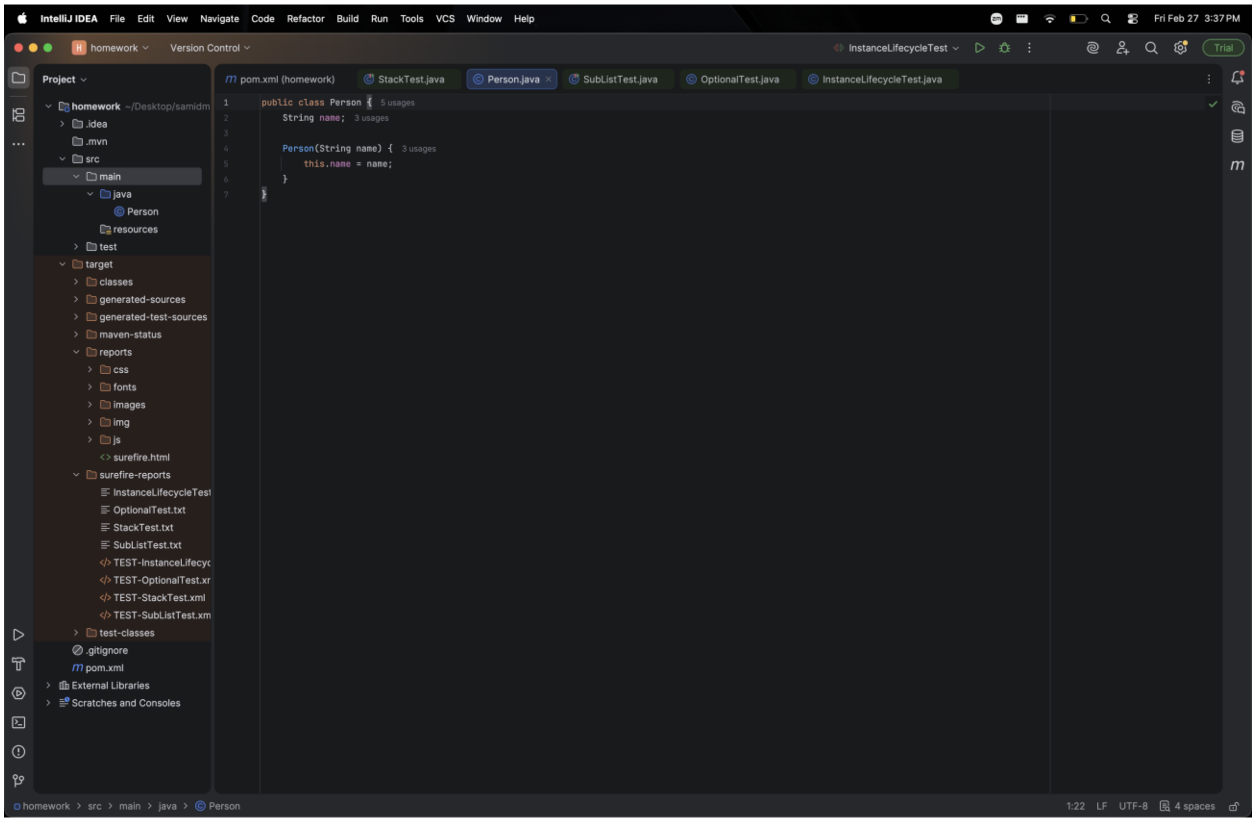


```java
import org.junit.jupiter.api.Test;

import java.util.Stack;

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

public class StackTest {
    @Test
    void stackShouldFollowLIFO() {
        Stack<Integer> stack = new Stack<>();

        // push() some elements
        stack.push( item: 2);
        stack.push( item: 56);
        stack.push( item: 0);
        stack.push( item: 89);
        stack.push( item: 14);

        // ensure LIFO order on pop()
        assertEquals( expected: 14, stack.pop());
        assertEquals( expected: 89, stack.pop());
        assertEquals( expected: 0, stack.pop());

        // ensure if peek() works
        assertEquals( expected: 56, stack.peek());

        stack.pop();
        stack.pop();

        // ensure stack is empty
        assertTrue(stack.isEmpty());

    }
}
```

Test II – subList() Shallow Copy

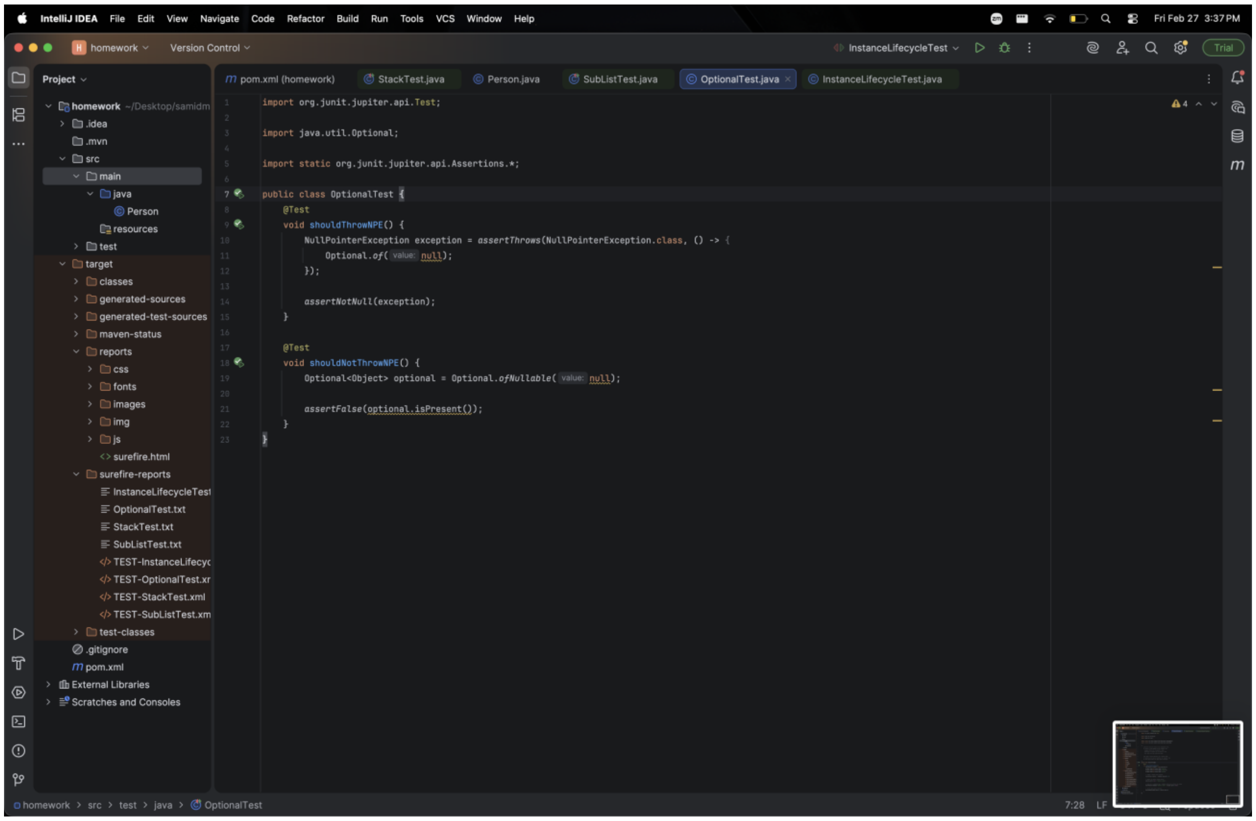


```java
import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.List;

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

/*
    subList() does not create a new independent list.
    It returns a view backed by the original list.
    - changes affect both (original & view).
    - modifying an element has an effect on both.
    - they share the same underlying data.

    Test case to prove subList() is a shallow copy:
    1) Modifying the subList modifies the original list.
    2) They should share the same object reference.
*/
public class SubListTest {
    @Test
    void subListIsShallowCopy() {
        List<Person> original = new ArrayList<>();
        original.add(new Person("Sami"));
        original.add(new Person("Landon"));
        original.add(new Person("Leo"));

        // create a sublist from original
        List<Person> subList = original.subList(0, 2);

        // modify the object inside subList
        subList.get(1).name = "Updated Landon";

        // since it's a shallow copy, original should also reflect the change.
        assertEquals("Updated Landon", original.get(1).name);

        // verify same object reference
        assertSame(original.get(1), subList.get(1));
    }
}
```



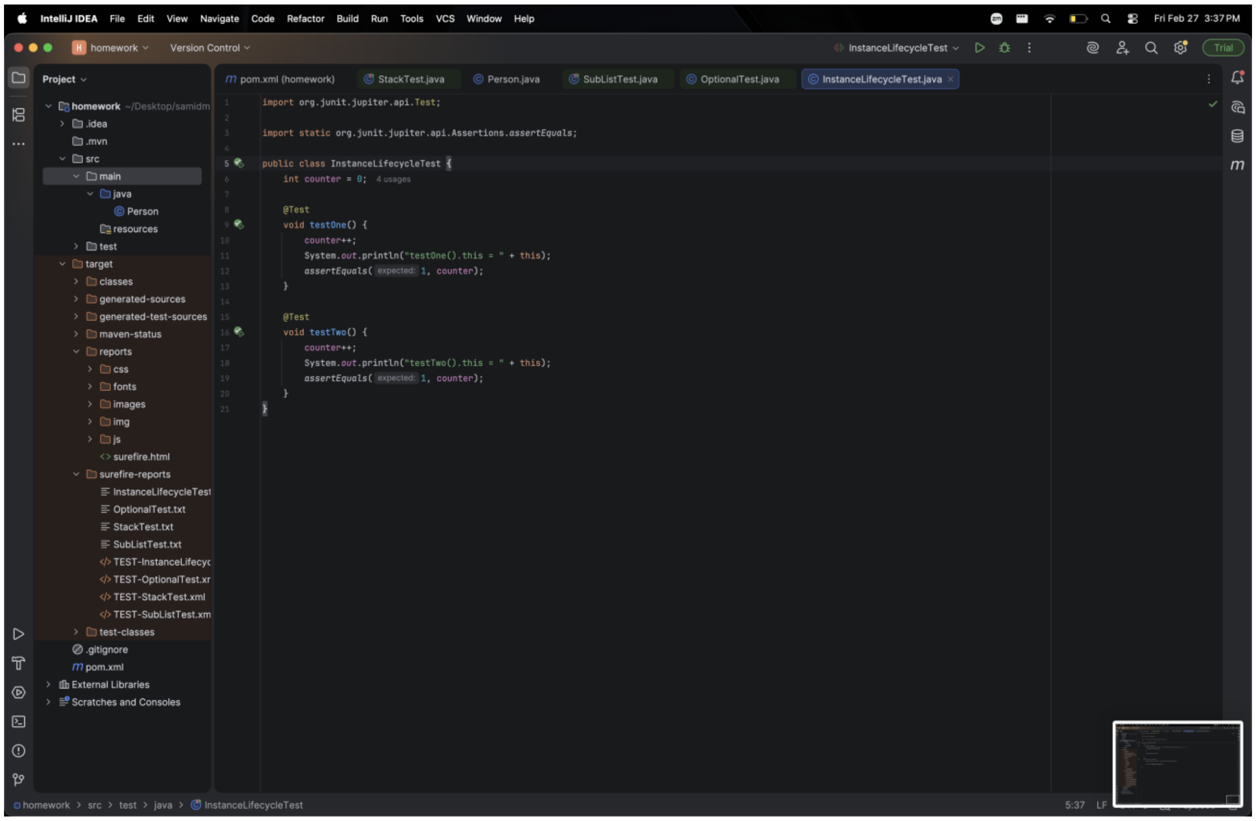
```java
public class Person {
    String name;

    Person(String name) {
        this.name = name;
    }
}
```

Test III – Optional.of() NULL



```java
import org.junit.jupiter.api.Test;

import java.util.Optional;

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

public class OptionalTest {
    @Test
    void shouldThrowNPE() {
        NullPointerException exception = assertThrows(NullPointerException.class, () -> {
            Optional.of(null);
        });

        assertNotNull(exception);
    }

    @Test
    void shouldNotThrowNPE() {
        Optional<Object> optional = Optional.ofNullable(null);

        assertFalse(optional.isPresent());
    }
}
```

Test IV – Test Instance Lifecycle Method



```java
import org.junit.jupiter.api.Test;

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

public class InstanceLifecycleTest {
    int counter = 0;

    @Test
    void testOne() {
        counter++;
        System.out.println("testOne().this = " + this);
        assertEquals(1, counter);
    }

    @Test
    void testTwo() {
        counter++;
        System.out.println("testTwo().this = " + this);
        assertEquals(1, counter);
    }
}
```