

Task 1

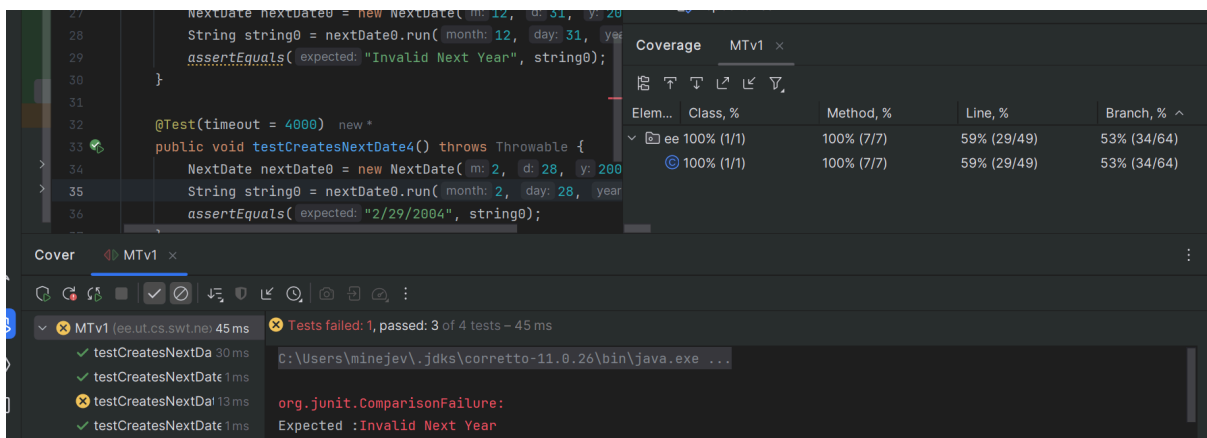
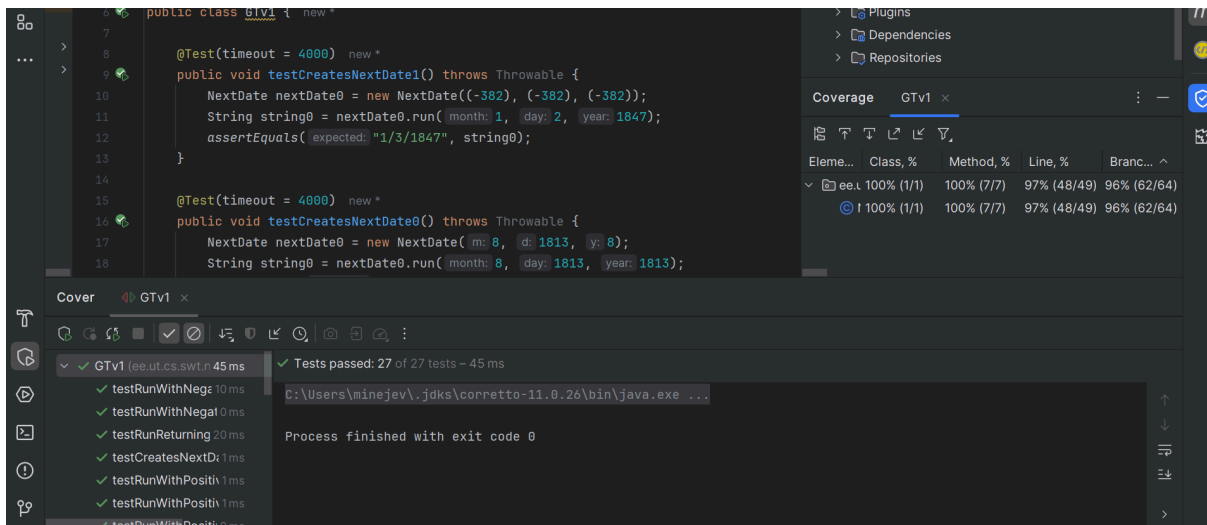


Table 1

Variable / Output	EC	Description	Covered by MTv1	Covered by GTv1
Month	M1	Valid month (1–12)	Yes	Yes
	M2	Invalid month (<1 or >12)	No	Yes
Day	D1	Valid day for given month and year	Yes	Yes
	D2	Invalid day (e.g., 2/30, negative)	No	Yes

Year	Y1	Valid year (e.g., 1801–2021)	Yes	Yes
	Y2	Invalid year (<1801 or >2021)	No	Yes
Output	O1	Valid next day in same month	Yes	Yes
	O2	Month change (e.g., 1/31 → 2/1)	Yes	Yes
	O3	Year change (e.g., 12/31 → 1/1)	Yes	Yes
	O4	"Invalid Input Date"	No	Yes
	O5	"Invalid Next Year"	No (test failed)	Yes
	O6	Invalid output (e.g., 12/32/1915)	No	Yes

Table 2

Aspect	MTv1	GTv1
Number of tests	4	27
Tests passed	3	27
Tests failed	1 (leap year test)	0
Input EC coverage	Only valid inputs	Valid and invalid inputs
Output EC coverage	O1, O2, O3	O1, O2, O3, O4, O5, O6
Detected failures	No actual detection (test failed due to wrong expectation)	Detected edge case outputs

Code coverage – Lines	59%	97%
Code coverage – Branches	53%	96%
Code coverage – Methods	100%	100%
EC coverage completeness	Partial	Comprehensive
Defect detection effectiveness	Low (missed logic)	High (validated corner cases)

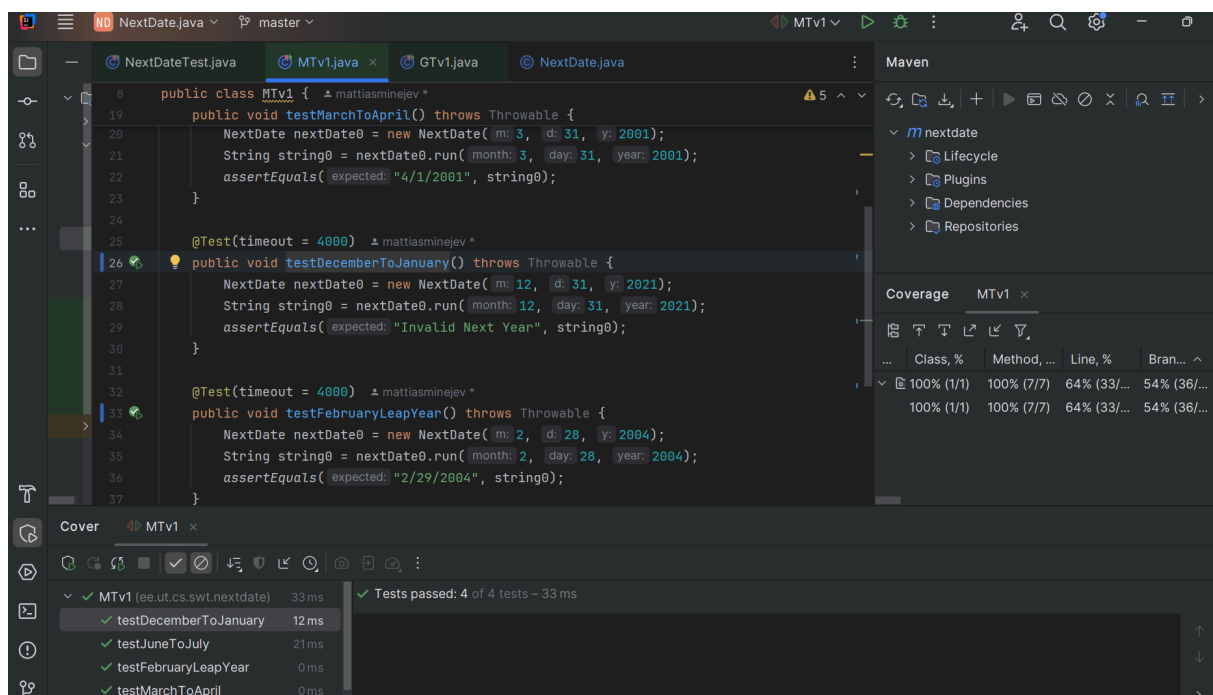
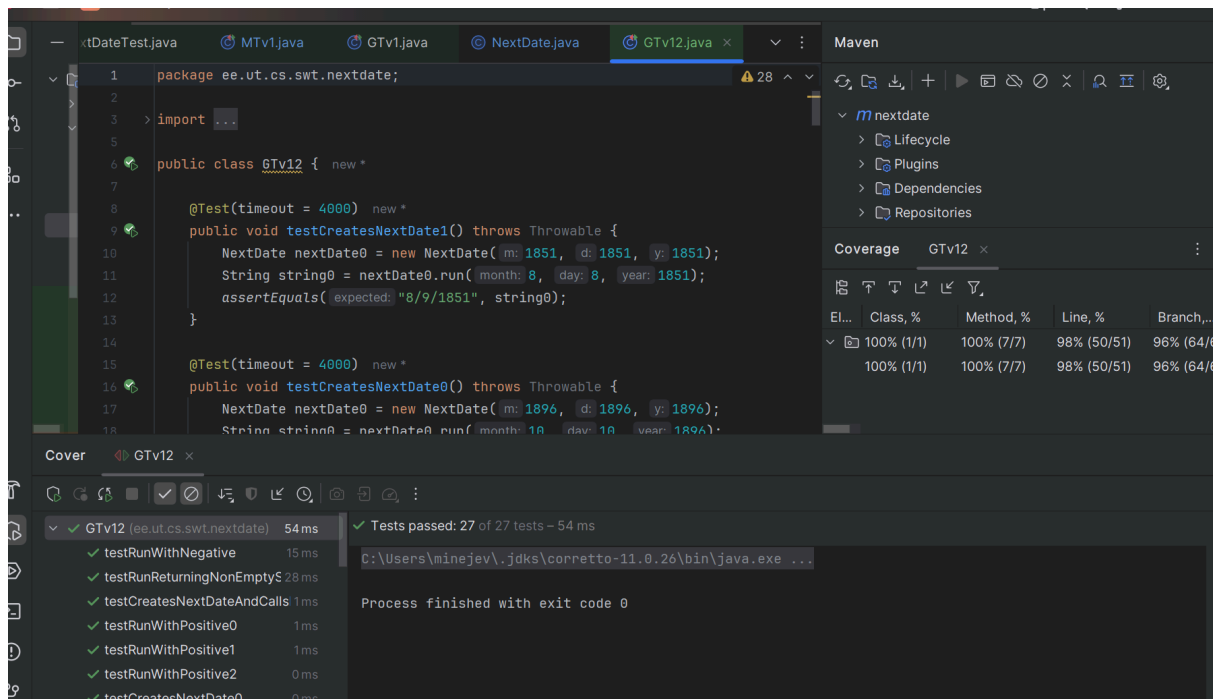
Task 2

The screenshot displays the IntelliJ IDEA IDE with the following components:

- Code Editor:** Shows the `GTV1` class with methods `testCreatesNextDate8`, `testCreatesNextDate10`, and `testCreatesNextDate12`. The code includes assertions and date calculations.
- Maven Tab:** Shows the project structure with coverage for `nextdate`, `Lifecycle`, `Plugins`, and `Dependencies`.
- Coverage Tab:** Displays a table with coverage metrics for the `GTV1` class.

El...	Class	%	Method	%	Line	%	Branch	%
100%	100%	100%	100%	100%	96%	95%	96%	95%
- Cover Tab:** Shows a list of tests and their results.

Test Name	Result	Duration
testRunWithNegative	Failed	76 ms
testRunWithNegativeAndNegr	Failed	13 ms
testRunReturningNonEmptyS	Failed	48 ms
testCreatesNextDateAndCalls	Failed	4 ms
testRunWithPositive0	Passed	0 ms
testRunWithPositive1	Passed	0 ms
testRunWithPositive2	Passed	0 ms
testCreatesNextDate0	Passed	0 ms
testCreatesNextDate1	Passed	0 ms
testCreatesNextDate2	Passed	1 ms
- Error Message:** `org.junit.ComparisonFailure: Expected :Invalid Next Year Actual :Invalid Input Date`



GTV1 made tests with wrong NextDate code therefore also the test inputs and expected outputs were wrong. Obviously GTv12 is better because it makes tests with corrected code.

Task3

The screenshot shows the IntelliJ IDEA IDE with the `GTV2.java` file open. The file is located in the package `ee.ut.cs` and contains the following code:

```
1 package ee.ut.cs;
2
3 import ...
4
5 public class GTv2 {
6     // ...
7
8     @Test(timeout = 4000)
9     public void testCreatesNDv2() {
10         NDv2 nDv2_0 = new NDv2();
11         String string0 = nDv2_0.run();
12         assertEquals("expected", string0, "7/1/2001");
13     }
14
15     @Test(timeout = 4000)
16     public void testCreatesNDv21() {
17         NDv2 nDv2_0 = new NDv2();
18         String string0 = nDv2_0.run();
19         assertEquals("expected", string0, "7/1/2001");
20     }
21 }
```

The Coverage window shows the following data:

Element	Class, %	Method, %	Line, %	Branch, %
ee	50% (1/2)	50% (7/14)	49% (50/102)	48% (64/132)
ee.ut.cs	0% (0/1)	0% (0/7)	0% (0/51)	0% (0/66)
ee.ut.cs.GTv2	100% (1/1)	100% (7/7)	98% (50/51)	96% (64/66)

The Coverage window also shows the following data:

Element	Class, %	Method, %	Line, %	Branch, %
ee	50% (1/2)	50% (7/14)	49% (50/102)	48% (64/132)
ee.ut.cs	0% (0/1)	0% (0/7)	0% (0/51)	0% (0/66)
ee.ut.cs.GTv2	100% (1/1)	100% (7/7)	98% (50/51)	96% (64/66)

The screenshot shows the IntelliJ IDEA IDE with the `MTv1.java` file open. The file is located in the package `ee.ut.cs.swt.nextdate` and contains the following code:

```
1 package ee.ut.cs.swt.nextdate;
2
3 import ...
4
5 public class MTv1 {
6     // ...
7
8     @Test(timeout = 4000)
9     public void testJuneToJuly() throws Throwable {
10         NextDate nextDate0 = new NextDate(m: 6, d: 30, y: 2001);
11         String string0 = nextDate0.run(month: 6, day: 30, year: 2001);
12         assertEquals("expected", string0, "7/1/2001");
13     }
14
15     @Test(timeout = 4000)
16     public void testDecemberToJanuary() {
17         NextDate nextDate0 = new NextDate(m: 12, d: 31, y: 2001);
18         String string0 = nextDate0.run(month: 12, day: 31, year: 2001);
19         assertEquals("expected", string0, "1/1/2002");
20     }
21
22     @Test(timeout = 4000)
23     public void testFebruaryLeapYear() {
24         NextDate nextDate0 = new NextDate(m: 2, d: 29, y: 2001);
25         String string0 = nextDate0.run(month: 2, day: 29, year: 2001);
26         assertEquals("expected", string0, "3/1/2001");
27     }
28
29     @Test(timeout = 4000)
30     public void testMarchToApril() {
31         NextDate nextDate0 = new NextDate(m: 3, d: 31, y: 2001);
32         String string0 = nextDate0.run(month: 3, day: 31, year: 2001);
33         assertEquals("expected", string0, "4/1/2001");
34     }
35 }
```

The Coverage window shows the following data:

Element	Class, %	Method, %	Line, %	Branch, %
ee	100% (1/1)	100% (7/7)	64% (33/51)	54% (36/66)
ee.ut.cs	100% (1/1)	100% (7/7)	64% (33/51)	54% (36/66)
ee.ut.cs.swt.nextdate	100% (1/1)	100% (7/7)	64% (33/51)	54% (36/66)

The Coverage window also shows the following data:

Element	Class, %	Method, %	Line, %	Branch, %
ee	100% (1/1)	100% (7/7)	64% (33/51)	54% (36/66)
ee.ut.cs	100% (1/1)	100% (7/7)	64% (33/51)	54% (36/66)
ee.ut.cs.swt.nextdate	100% (1/1)	100% (7/7)	64% (33/51)	54% (36/66)

Due to `isLeapYear` already uncommented and working, both test files worked to perfection and all tests passed.

Task4.1

The screenshot displays an IDE with two panels. The top panel shows the source code for `GTV2.java` with two test methods: `testCreatesNDv220` and `testCreatesNDv25`. The bottom panel shows the test results for `MTV2`, indicating that 1 test failed and 25 tests passed. The failed test is `testCreatesNDv220`, which failed due to a `org.junit.ComparisonFailure` where the expected value was `Invalid Input Date` and the actual value was `2/29/1900`. The error message includes a link to see the difference and the location of the failure in the code: `at ee.ut.cs.swt.nextdate.GTv2.testCreatesNDv220(GTv2.java:131)`.

The bottom panel also shows the source code for `NDv2.java`, which contains the `run` method. The method logic is as follows:

```
public String run(int month, int day, int year) {
    if (isLeapYear(year)) { //AND a leap year - reset the day to 1, mont
        tomorrowDay = 1;
        tomorrowMonth = 3;
    }
    else
        return "Invalid Input Date";
    else if (day > 29) //invalid input as February will never have more than
        return "Invalid Input Date";
    //return the string representing the nextDate, in the form MM/DD/YY
    return tomorrowMonth + "/" + tomorrowDay + "/" + tomorrowYear;
}
```

The test results for `MTV1` show that all 4 tests passed, with a total execution time of 27 ms. The tests are:

- `testDecember1` (0 ms)
- `testJuneToJu` (20 ms)
- `testFebruaryLe` (0 ms)
- `testMarchToAp` (1 ms)

The performance section indicates that the process finished with exit code 0 and was terminated.

One test failed. it failed due to the method not being able to bring out error: "Invalid Input Date".

Task 4.2

The screenshot shows an IDE with the file `MTv1.java` open. The code defines a class `MTv1` with two test methods: `testJuneToJuly()` and `testMarchToApril()`. The `testJuneToJuly()` method creates a `NextDate` object for June 30, 2001, and expects the next date to be July 1, 2001. The `testMarchToApril()` method is also present but its implementation is partially obscured. The coverage report on the right shows 50% class coverage, 50% method coverage, 32% line coverage, and 27% branch coverage. The bottom panel shows the test results for `MTv1`, indicating that 4 out of 4 tests passed in 46 ms.

```
package ee.ut.cs.swt.nextdate;

import ...

public class MTv1 {

    @Test(timeout = 4000)
    public void testJuneToJuly() throws Throwable {
        NextDate nextDate0 = new NextDate(m: 6, d: 30, y: 2001);
        String string0 = nextDate0.run(month: 6, day: 30, year: 2001);
        assertEquals(expected: "7/1/2001", string0);
    }

    @Test(timeout = 4000)
    public void testMarchToApril() throws Throwable {
        NextDate nextDate0 = new NextDate(m: 3, d: 31, y: 2001);
    }
}
```

El...	Class, %	Method, %	Line, %	Branch, %
✓	50% (1/2)	50% (7/14)	32% (33/102)	27% (36/132)
✓	0% (0/1)	0% (0/7)	0% (0/51)	0% (0/66)
✓	100% (1/1)	100% (7/7)	64% (33/51)	54% (36/66)

Coverage: MTv1

Tests passed: 4 of 4 tests - 46 ms

Process finished with exit code 0

The screenshot shows an IDE with the file `GTv12.java` open. The code defines a class `GTv12` with two test methods: `testCreatesNextDate1()` and `testCreatesNextDate0()`. The `testCreatesNextDate1()` method creates a `NextDate` object for August 8, 1851, and expects the next date to be August 9, 1851. The `testCreatesNextDate0()` method creates a `NextDate` object for August 10, 1896, and expects the next date to be August 11, 1896. The coverage report on the right shows 50% class coverage, 50% method coverage, 49% line coverage, and 48% branch coverage. The bottom panel shows the test results for `GTv12`, indicating that 27 out of 27 tests passed in 42 ms.

```
package ee.ut.cs.swt.nextdate;

import ...

public class GTv12 {

    @Test(timeout = 4000)
    public void testCreatesNextDate1() throws Throwable {
        NextDate nextDate0 = new NextDate(m: 1851, d: 1851, y: 1851);
        String string0 = nextDate0.run(month: 8, day: 8, year: 1851);
        assertEquals(expected: "8/9/1851", string0);
    }

    @Test(timeout = 4000)
    public void testCreatesNextDate0() throws Throwable {
        NextDate nextDate0 = new NextDate(m: 1896, d: 1896, y: 1896);
        String string0 = nextDate0.run(month: 10, day: 10, year: 1896);
        assertEquals(expected: "10/11/1896", string0);
    }
}
```

El...	Class, %	Method, %	Line, %	Branch, %
✓	50% (1/2)	50% (7/14)	49% (50/102)	48% (64/132)
✓	0% (0/1)	0% (0/7)	0% (0/51)	0% (0/66)
✓	100% (1/1)	100% (7/7)	98% (50/51)	96% (64/66)

Coverage: GTv12

Tests passed: 27 of 27 tests - 42 ms

Process finished with exit code 0

Now the test works again. Nothing surprising.

Task 5

The screenshot displays an IDE with the following components:

- Code Editor:** Shows the `GTv1.java` file with two test methods:
 - `testCreatesShoppingCartAndCallsCancelCurrentPurchase()` (lines 14-18)
 - `testCreatesShoppingCartAndCallsSubmitCurrentPurchase()` (lines 21-24)
- Coverage View:** Located at the top right, it shows a table of coverage data:

Element	Class, ...	Method, ...	Line, %
ee.ut.math.tvt.salessysteme	100% (1...	100% (5/5)	81% (13/16)
ShoppingCart	100% (1...	100% (5/5)	81% (13/16)
- Test Results View:** Located at the bottom, it shows a list of tests and their execution times:
 - `testCreatesShoppy` 16 ms
 - `testCreatesShoppy` 1 ms
 - `testCancelCurrent` 0 ms
 - `testSubmitCurrent` 1 ms
 - `testSubmitCurrent` 8 ms
 - `testCreatesShoppy` 1 ms
- Error Details:** A red error message is displayed: `java.lang.NullPointerException: Cannot invoke "ee.ut.math.tvt.salessystem.dao.SalesSystemDAO.beginTransaction()"`. The stack trace points to `ShoppingCart.submitCurrentPurchase` (line 48) and `GTv1.testSubmitCurrentPurchaseThrowsNullPointerException` (line 85).

1. How strong is the generated test suite?

Päris tugev – katab 81% koodireadest ja 100% meetoditest.

2. How could you check the strength of the generated test suite?

Vaadates koodi katvust ja kontrollides, kas testid tuvastavad vigu (nt mutation testing).

3. What does the generated test suite say about the correctness of your program?

Põhiline loogika töötab, aga üks `NullPointerException` näitab, et kõik sõltuvused pole korrektselt algatatud.