SoftwareTesting.java

```
White-Box Testing {

    int courseNumber = 4;

}
```

SoftwareTesting.java

Cuprins {

}

WhiteBox.java

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

01 {

[@Test White-box]

String definition = “Testare bazată pe o analiză a structurii interne de componentă sau sistem.”

}

StatementCoverage.java

02 {

[@Test Statement Coverage]

String definition = “Procentul de instrucțiuni executabile care au fost exercitate printr-o suită de teste.”

}

StatementCoverage.java

```
1  class Statement {
2
3        O entitate într-un limbaj de programare, care este
4      de obicei cea mai mică unitate de execuție
5      indivizibilă.
6  }
7
8
9  class Main {
10
11         // Un exemplu de statement
12         System.out.println(“Hello World!”);
13
14 }
```

StatementCoverage.java

Statement coverage ‘Exercitiu’ {

Să luăm în considerare fragmentul de cod de mai jos:

```
public void main(String[] args) {
    int Stock = 0;

    if (Stock < 1) {
        System.out.println("Error message");
    }
}
```

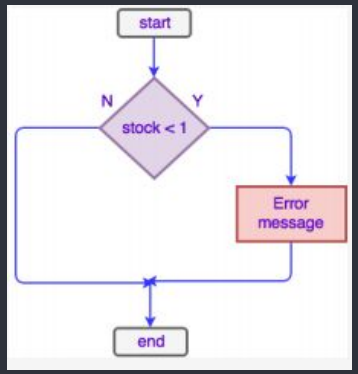
Statement coverage  100%

}

StatementCoverage.java

Statement coverage ‘Exercitiu’ {

```java
public void main(String[] args) {
    int Stock = 0;

    if (Stock < 1) {
        System.out.println("Error message");
    }
}
```



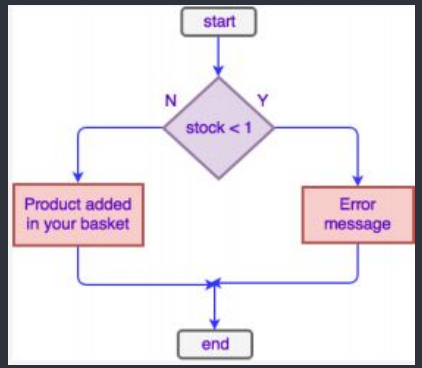
Statement coverage 100%

}

StatementCoverage.java

Statement coverage ‘Exercitiu’ {

```java
public void main(String[] args) {
    int Stock = 0;

    if (Stock < 1) {
        System.out.println("Error message");
    } else {
        System.out.println("Product added to basket");
    }
}
```



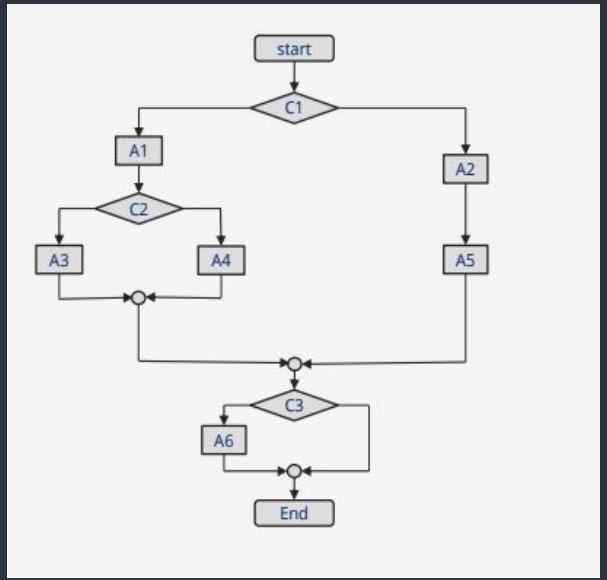
Statement coverage 100%

}

StatementCoverage.java

Statement coverage ‘Exercitiu’ {

```java
public void main(String[] args) {
    int sum = 22;
    int number = 2;

    double average = 0.0;

    boolean condition = false;

    if (sum < 10) {
        sum = number + sum;
        number++;
        condition = true;
    } else {
        average = sum / number;
        if (average < 20) {
            System.out.println("Average is smaller than 20");
            condition = true;
        } else {
            System.out.println("Average is bigger than 20");
            condition = false;
        }
    }
    if (!condition) {
        System.out.println("Condition not fullfilled");
    }
}
```



}

ISTQB.java

```
ISTQB Q1 {

}
```

Question #22 (1 Point)

Which one of the following is the description of statement coverage?

a) It is a metric, which is the percentage of test cases that have been executed
b) It is a metric, which is the percentage of statements in the source code that have been executed
c) It is a metric, which is the number of statements in the source code that have been executed by test cases that are passed
d) It is a metric, that gives a true/false confirmation if all statements are covered or not

Select ONE option.

ISTQB.java

ISTQB Q2 {

Question #27 (1 Point)

Which of the following descriptions of statement coverage is CORRECT?

a) Statement coverage is a measure of the number of lines of source code exercised by tests
b) Statement coverage is a measure of the proportion of executable statements in the source code exercised by tests
c) Statement coverage is a measure of the percentage of lines of source code (without comments) exercised by tests
d) Statement coverage is a measure of the number of executable statements in the source code exercised by tests

Select ONE option.

}

DecisionCoverage.java

03 {

[@Test Decision Coverage]

String definition = “Acoperirea deciziilor, legată de testarea ramurilor, este evaluarea procentului deciziei (de exemplu, opțiunile adevărate și false ale unei declarații IF) care a fost exercitat de o suită de teste”

}

DecisionCoverage.java

Decision Coverage() {

// Testarea deciziilor este o formă de testare a fluxului de control, deoarece urmează un flux specific de control prin punctele de decizie

// Acoperirea deciziilor este mai puternică decât acoperirea declarației

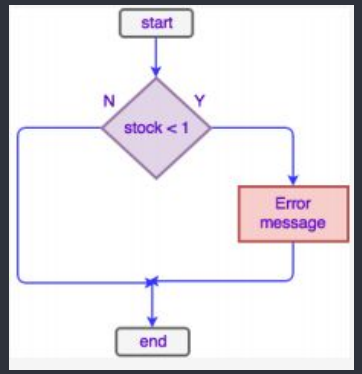
// Acoperirea 100% a deciziilor garantează acoperirea declarației 100%, dar nu vice versa

}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {

```java
public void main(String[] args) {
    int Stock = 0;

    if (Stock < 1) {
        System.out.println("Error message");
    }
}
```



Statement coverage 100%

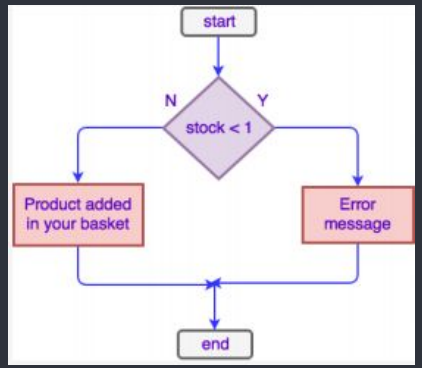
Decision coverage 50%

}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {

```java
public void main(String[] args) {
    int Stock = 0;

    if (Stock < 1) {
        System.out.println("Error message");
    } else {
        System.out.println("Product added to basket");
    }
}
```



Decision coverage 100%

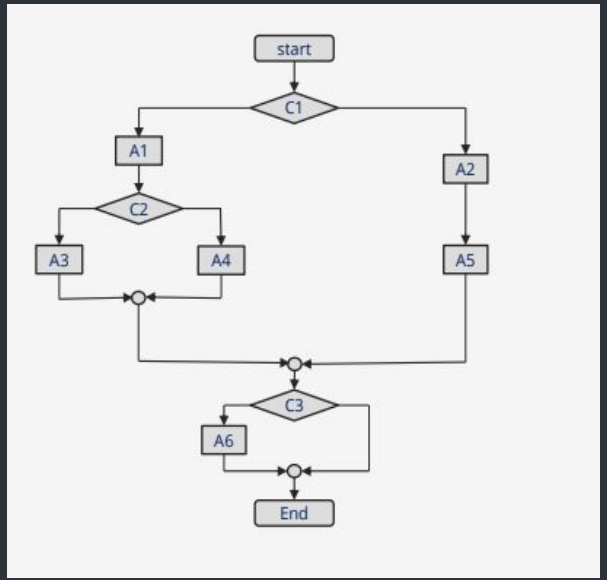
Statement coverage 100%

}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {

```java
public void main(String[] args) {
    int sum = 22;
    int number = 2;

    double average = 0.0;

    boolean condition = false;

    if (sum < 10) {
        sum = number + sum;
        number++;
        condition = true;
    } else {
        average = sum / number;
        if (average < 20) {
            System.out.println("Average is smaller than 20");
            condition = true;
        } else {
            System.out.println("Average is bigger than 20");
            condition = false;
        }
    }
    if (!condition) {
        System.out.println("Condition not fullfilled");
    }
}
```



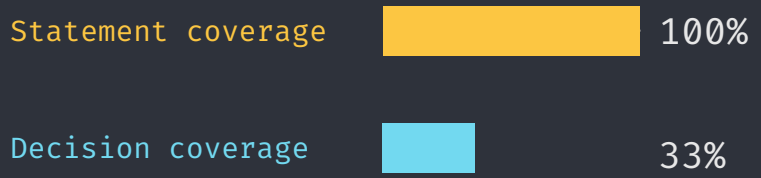
}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {

```java
if (x > y) {

    if (x > z) {

        System.out.print("Message");
    }

}
```

1 TC



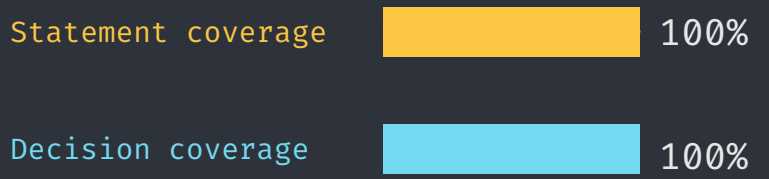
}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {

```java
if (x > y) {

    if (x > z) {

        System.out.print("Message");
    }

}
```

3 TC



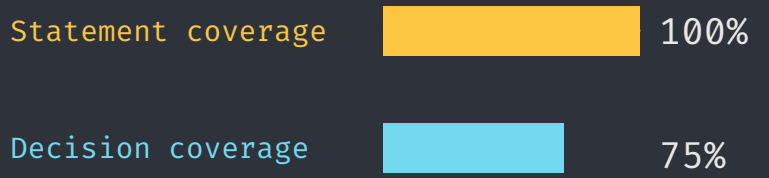
}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {

```java
if (x > y) {

    if (x > z) {

        System.out.print("Message");
    } else {
        System.out.println("Other Message");
    }

}
```

2 TC



}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {

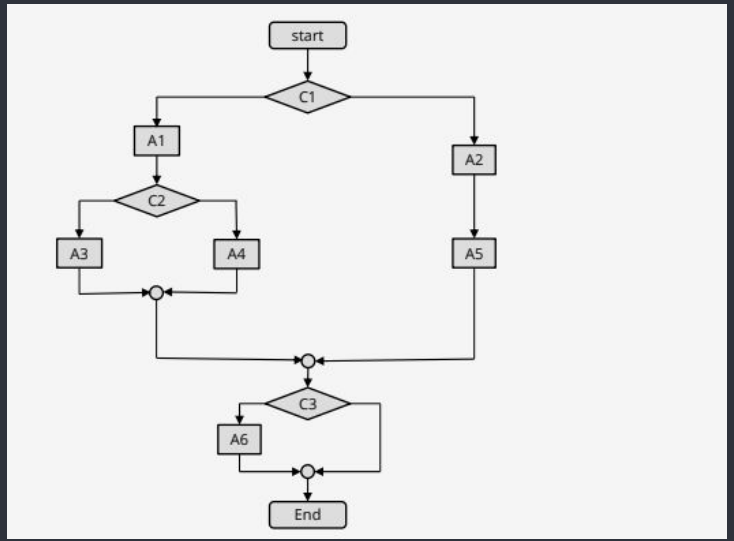
```java
if (x > y) {

    if (x > z) {

        System.out.print("Message");
    } else {
        System.out.println("Other Message");
    }

}
```

3 TC

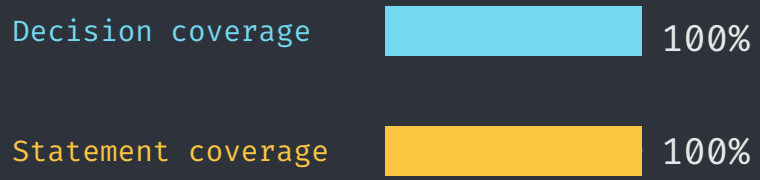
}

DecisionCoverage.java

Decision coverage ‘Exercitiu’ {



}



// 100% Decision coverage asigură 100% statement coverage

ISTQB.java

ISTQB Q3 {

Question #21 (1 Point)
The following statement refers to decision coverage:

"When the code contains only a single 'if' statement and no loops or CASE statements, and its execution is not nested within the test, any single test case we run will result in 50% decision coverage."

Which of the following statement is correct?

a) The statement is true. Any single test case provides 100% statement coverage and therefore 50% decision coverage
b) The statement is true. Any single test case would cause the outcome of the "if" statement to be either true or false
c) The statement is false. A single test case can only guarantee 25% decision coverage in this case
d) The statement is false. The statement is too broad. It may be correct or not, depending on the tested software

Select ONE option.

}

ISTQB.java

```
ISTQB Q4 {

}
```

Question #23 (1 Point)

Which statement about the relationship between statement coverage and decision coverage is true?

a) 100% decision coverage also guarantees 100% statement coverage
b) 100% statement coverage also guarantees 100% decision coverage
c) 50% decision coverage also guarantees 50% statement coverage
d) Decision coverage can never reach 100%

Select ONE option.

ISTQB.java

ISTQB Q5 {

Question #28 (1 Point)

Which of the following descriptions of decision coverage is CORRECT?

a) Decision coverage is a measure of the percentage of possible paths through the source code exercised by tests
b) Decision coverage is a measure of the percentage of business flows through the component exercised by tests
c) Decision coverage is a measure of the 'if' statements in the code that are exercised with both the true and false outcomes
d) Decision coverage is a measure of the proportion of decision outcomes in the source code exercised by tests

Select ONE option.

}

PathCoverage.java

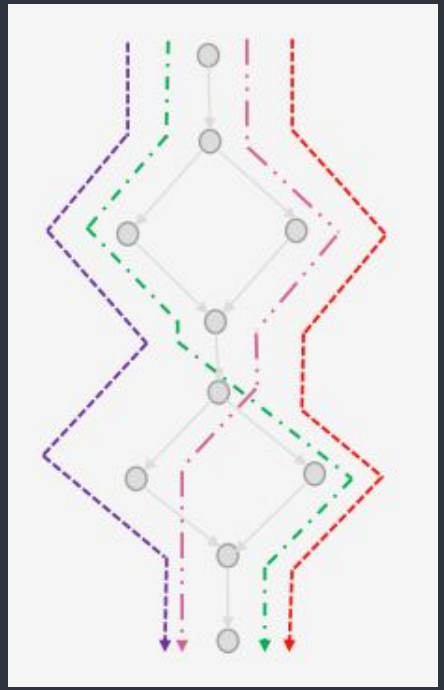
04 {

[@Test Path Coverage]

String definition = “Se concentrează pe executarea tuturor căilor posibile prin program.”

}

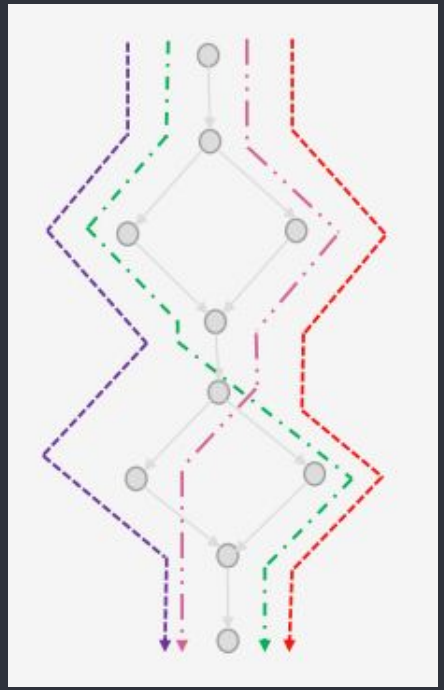
PathCoverage.java



2 TCs pentru statement coverage
- ROSU
- MOV

Statement coverage 100%

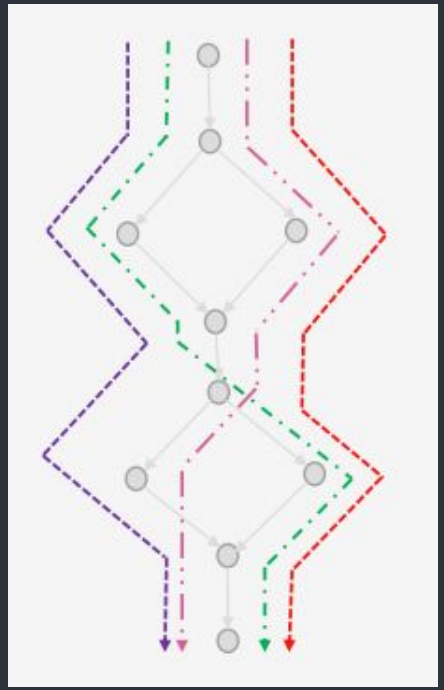
PathCoverage.java



2 TCs pentru statement coverage
- VERDE
- ROZ

Decision coverage 100%

PathCoverage.java

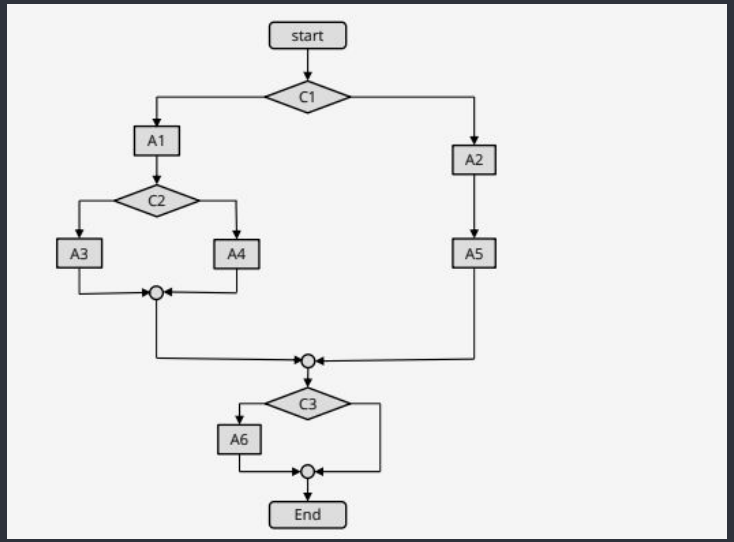


4 TCs pentru path coverage
- ROSU, MOV
- VERDE, ROZ

Path coverage 100%

PathCoverage.java

Path coverage ‘Exercitiu’ {



}