Testare si Verificare

- Laborator -

Ana Cristina Țurlea
ana.turlea@fmi.unibuc.ro

Organizare

- Laborator la 2 săptămâni;
- Nota: proiect individual 50% din nota finală.
- Prezentările de proiecte pot începe din săptămâna a 4a.
- Cerințe proiect: - dl Prof
- Programare Prezentări săptămânile 4-10: https://doodle.com/poll/shegv86yhqf5cihh
- Link document https://goo.gl/fT2EAB
- Email: ana.turlea@fmi.unibuc.ro

Testare software

- Test - execuția unui program pentru un set de date de intrare (ales convenabil), pentru a verifica dacă rezultatul obținut este corect.
- Test case - set de date de intrare + date de iesire.
- Test suite - set de ‘test cases’
- JUnit - testing framework;
- Testele se scriu în clase speciale (folosite doar pentru testare).
- Pentru a defini o metodă de tip test se folosește adnotarea @Test;
    - Aceasta executa porțiunea de cod testată;
    - Poate folosi *assert* pentru a verifica dacă rezultatul obținut este același cu rezultatul asteptat.

JUnit

- Instalare: https://github.com/junit-team/junit4/wiki/Download-and-Install
- Maven:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Exemplu

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

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

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

JUnit4 - Adnotări

import org.junit.*	Importul necesar pentru a folosi adnotările.
@Test	Identifică o metodă ca fiind o metodă de test.
@Before	Indică faptul ca metoda se execută înainte de fiecare test. Este folosit pentru a pregăti mediul de testare (de ex citirea datelor de intrare sau inițializarea claselor).
@After	Se execută după fiecare test. Folosit pentru a curăța mediul de testare.
@BeforeClass	Se execută o singură dată, înainte de rularea tuturor testelor. Poate fi folosit, de exemplu, pentru conectarea la baza de date.
@AfterClass	Se execută o singură dată, după ce toate testele au fost rulate. Poate fi folosit, de exemplu, pentru deconectarea de la baza de date.
@Test (expected = Exception.class)	Acest test va pica dacă metoda nu aruncă respectiva excepție.
@Test(timeout=100)	Acest test va pica dacă metoda rulează mai mult de 100 millisecunde.

Clasa Assert

fail([message])	Metoda va eșua cu mesajul respectiv.
assertTrue([message,] boolean condition)	Verifică dacă condiția booleană este true.
assertFalse([message,] boolean condition)	Verifică dacă condiția booleană este false.
assertEquals([message,] expected, actual)	Verifică dacă două valori sunt egale. Pentru elemente de tip array este verificată referința obiectelor si nu conținutul.
assertNull([message,] object)	Verifică dacă obiectul este null.
assertNotNull([message,] object)	Verifică dacă obiectul nu este null.

Proiect lab - cerinta

Sa se scrie un program Java, precum si cerintele (specificatia) acestuia.

1. Pe baza cerintelor programului, sa se genereze date de test folosind a) equivalence partitioning si b) boundary value analysis. Sa se implementeze testele obtinute folosind JUnit.
2. Sa se transforme programul intr-un graf orientat si, pe baza acestuia, sa se genereze date de test care realizeaza acoperire la nivel de a) instructiune, b) decizie, c) conditie d) modified condition/decision. Sa se implementeze testele obtinute folosind JUnit.
3. Sa se calculeze complexitatea programului folosind formula lui McCabe, precum si numarul de circuite independente.
4. Sa se propuna un set de teste pentru o acoperire la nivel de cale si sa se implementeze testele rezultate in JUnit.
5. Sa se foloseasca un generator de mutanti (e.g. PIT, MuJava/Muclipse) pentru a crea mutanti pentru programul dat. Sa se ruleze seturile de test de la punctele 1), 2) si 4) pe mutantii generati si sa se explice rezultatele.
6. Sa se creeze teste suplimentare pentru a omora 2 dintre mutantii neechivalenti ramasi in viata.

Exemple

1. Se da un număr n (1 <= n < 10); un sir de n numere naturale si o cifra c. Se cere sa se afiseze al c-lea număr din sir ce respecta condiția ca este palindrom prim.

2. Se citește de la tastatură un număr întreg şi pozitiv n, de maximum nouă cifre, și o cifra c. Să se verifice dacă numărul n este palindrom și, dacă este, să se verifice dacă cifra din centrul lui n este egală cu c, în cazul în care n are un număr impar de cifre, sau perechea de două cifre din centrul lui n este formată din cifra c, dacă n are un număr par de cifre.
   - Dacă n este număr palindrom și c este egal cu centrul lui n sau cu cifrele care alcătuiesc centrul lui n atunci rezultatul afișat este ”11”
   - Dacă n este număr palindrom, dar c nu este egal cu centrul lui n sau cu cifrele care alcătuiesc centrul lui n atunci rezultatul afișat este ”10”
   - Dacă n nu este număr palindrom atunci rezultatul afișat este "00"
   - Pentru cazurile de eroare se va afisa un mesaj corespunzator.

3. Se citesc doua cuvinte de la tastatură de lungime maxima 20, ce contin doar litere si cifre. Sa se determine dacă cele două cuvinte sunt anagrame. Cuvintele sunt considerate anagrame daca au același număr de litere și acestea apar de același număr de ori în ambele cuvinte.

4. Se dă un şir cu n (2 <= n <= 100) elemente, numere naturale, de maxim 5 cifre. Să se verifice dacă oglinditul primului element apare printre celelalte elemente ale șirului.

Exemple

5. Se citește un șir cu n (1 <= n <= 100) numere naturale, de maxim 9 cifre. Să se verifice dacă prin rearajarea elementelor șirului se poate obține un șir palindrom. Exemplu de sir palindrom: [1,2,3,4,5,4,3,2,1]
6. Se citește un număr natural n de maxim 9 cifre si un numar x (de maxim 2 cifre). Să se determine daca numărul de divizori ai oglinditului lui n este mai mare decat x.
7. Se citește un șir cu n (1 <= n <= 100) numere naturale, de maxim 9 cifre, nu neapărat distincte. Să se afișeze elementul care apare de cele mai multe ori în șir (in cazul in care exista mai multe elemente se va afisa primul gasit).
8. Se consideră două numere naturale x şi y (cu x < y) şi un şir A de n elemente numere întregi, 1 <= n <= 100. Se cere să se afişeze numărul elementelor perfecte din şir, care se află în intervalul [x, y]. Ex: Fie x=3, y=25, n=5 şi a=(10,6, 7,28, 4), atunci numărul nr=2, deoarece 6 şi 28 sunt perfecte (suma divizorilor mai mici decât numărul, este egală cu numărul).
9. Se consideră un şir A, de n elemente numere întregi. Se cere să se afişeze numărul elementelor cu proprietatea că suma cifrelor lor este număr prim. Dacă nu există nici un astfel de număr se va afişa un mesaj.
10. Se consideră un şir A de n elemente numere întregi (1 <= n <= 100), doua numere naturale x si y. Se cere să se verifice daca suma elementelor divizibile cu x ale şirului se afla in intervalul [-y,y].

Exemple

11. Se consideră un şir de n elemente numere naturale de cel puţin două cifre. Se cere să se verifice daca numarul de elemente ale şirului care au ultima cifră impară este mai mare decat n/2.

12. Se consideră două mulțimi, a și b, de numere naturale de maxim 5 cifre, a având n elemente, b având m elemente, 1 <= n,m <= 100. Se cere să se formeze al treilea vector (cu proprietate de mulțime, adică având elemente distincte) reprezentând reuniunea mulțimilor date.

13. Se consideră două mulțimi, a și b, de numere naturale de maxim 5 cifre, a având n elemente, b având m elemente, 1 <= n,m <= 100. Se cere să se formeze al treilea vector (cu proprietate de mulțime, adică având elemente distincte) reprezentând intersectia mulțimilor date.

14. Se consideră două mulțimi, a și b, de numere naturale de maxim 5 cifre, a având n elemente, b având m elemente, 1 <= n,m <= 100. Se cere să se formeze al treilea vector (cu proprietate de mulțime, adică având elemente distincte) reprezentând diferenta mulțimilor date.

15. Se da o matrice bidimensionala cu m linii si n coloane, 1 <= n,m <= 100, cu elemente intregi din intervalul [-500, 500]. Sa se verifice daca numarul de elemente pozitive este mai mare decat numarul de elemente negative.

To Be Added...

Bibliografie:

- JUnit tutorial: https://www.vogella.com/tutorials/JUnit/article.html