Jegyzőkönyv

Szoftvertesztelés

GEIAL31H-BL

Féléves beadandó

Készítette:
Burka Erik
BFATS0

Feladat leírás:

A választott témaköröm az egységteszt bemutatása java környezetben. Először az egységteszt irodalmi bemutatása majd IntelliJ fejlesztői környezetben egy java project környezeten belüli bemutatása.

Egységteszt:

„A számítógép-programozásban az egységtesztelés a szoftvertesztelésnek egy olyan módszere, amelynek során a forráskód egységeit (egy vagy több számítógépes program modul készletet) a kapcsolódó vezérlő adatokkal, a felhasználási-és a működtető eljárásokkal együtt tesztelik annak meghatározására, hogy azok elérik-e kitűzött céljukat. Az egységtesztek általában automatizált tesztek, amelyeket a szoftverfejlesztők írnak és futtatnak annak biztosítása érdekében, hogy egy adott alkalmazás adott szakasza (úgynevezett "egység") megfeleljen a kialakításának és az elvárt, kívánt módon viselkedjen. A procedurális programozásban az egység lehet egy teljes modul, de túlnyomórészt egyéni függvény vagy eljárás. Objektumorientált programozás során az egység gyakran egy teljes felület/interfész, például egy osztály, de lehet egyedi metódus is. Az esetlegesen felmerülő problémák elkülönítése érdekében minden tesztesetet egymástól függetlenül, külön kell tesztelni.
Előnye:

Az egységtesztelés a szoftverfejlesztési folyamat korai szakaszában találja meg a problémákat. Ez egyaránt magában foglalja mind a programozó végrehajtásának bugjait, mind az egység specifikációjának hibáit vagy annak hiányzó részeit. Az alapos, részletes tesztkészlet megírásának folyamata arra készteti a fejlesztőt, hogy gondolja át a bemeneteket, a kimeneteket és a hibafeltételeket, ezáltal így pontosabban meghatározza az egység kívánt viselkedését. A hibakeresés költsége a kódolás megkezdése vagy a kód első írása előtt jóval alacsonyabb, mint a hiba későbbi felismerésének, azonosításának és kijavításának költsége. Az egységteszt lehetővé teszi a programozó számára a kódrefaktorálást vagy a rendszerkönyvtárak egy későbbi időpontban történő frissítését és a modul továbbra is helyes működéséről való megbizonyosodást (például regressziós teszteléskor). Az egységteszt olyan jellemzőket testesít meg, amelyek kritikusak az egység sikeréhez. Ezek a jellemzők jelezhetik az egység megfelelő / nem megfelelő használatát, továbbá a negatív viselkedéseket, amelyek az egység által „csapdába” (trapped by the unit) estek. Egy egységteszt önmagában dokumentálja ezeket a kritikus jellemzőket, habár sok szoftverfejlesztő környezet nem kizárólag arra szolgál, hogy a fejlesztés alatt álló terméket dokumentálja a kód alapján.

Korlátok és hátrányok:

A tesztelés nem fog minden hibát elfogni a programban, mert nem tud minden végrehajtási útvonalat kiértékelni. Ez a döntési probléma (a megállási probléma (halting problem) szülő-halmaza, azaz, hogy az adott program befejezi-e a futását vagy örökre fut,) eldönthetetlen, mivel lehetnek olyan kódrészek, amikre nem írható olyan ellenőrző algoritmus, amely igaz-hamis válasszal szolgálna a kimenetében. Ugyanez igaz az egységtesztelésre is. Ezenkívül, a definíció szerint az egységteszt csak maguknak az egységeknek a funkcionalitását teszteli. Ebből kifolyólag nem fog elkapni integrációs hibákat vagy tágabb, rendszerszintű hibákat (például több egységen végrehajtott függvények vagy nem funkcionális tesztterületek, például a teljesítmény) Az egységtesztelést más szoftvertesztelési tevékenységekkel összefüggésben kell elvégezni, mivel ezek csak bizonyos hibák jelenlétét vagy hiányát mutatják; nem tudják bizonyítani az összes hiba hiányát. Ahhoz, hogy garantálva legyen a végrehajtás minden útjának és minden lehetséges bemenetnek a helyes viselkedése és biztosítva legyen a hibák hiánya, más technikákra is szükség van, nevezetesen formális módszerek alkalmazására annak igazolására, hogy a szoftverkomponensnek nincs váratlan viselkedése. A szoftvertesztelés egy kombinatorikus probléma. Például, minden Boolean döntési nyilatkozatnak legalább két tesztre van szüksége: egy „igaz” és egy „hamis” kimenetelűre. Ennek eredményeként a programozóknak minden megírt kódsorra 3–5 sornyi tesztkódra van szükségük. Ez nyilvánvalóan időt vesz igénybe és esetleg nem éri meg a belefektetett erőfeszítést.” [1]

JUnit:

A JUnit egy egységteszt-keretrendszere Java programozási nyelvhez. A fejlesztés során nem csak a kódunkat, hanem magát a tesztet is párhuzamosan fejleszteni kell. A JUnit segít a megírt tesztek csoportos formában való futtatására. A project úgynevezett „release” akkor hibátlan, ha minden teszt hiba nélkül lefut. „A JUnit az egységteszt-keretrendszerek családjába tartozik, melyet összességében xUnit-nak hívunk, amely eredeztethető a SUnitból. JUnit keretrendszer fizikailag egy JAR fájlba van csomagolva.
A keretrendszer osztályai következő csomag alatt található:

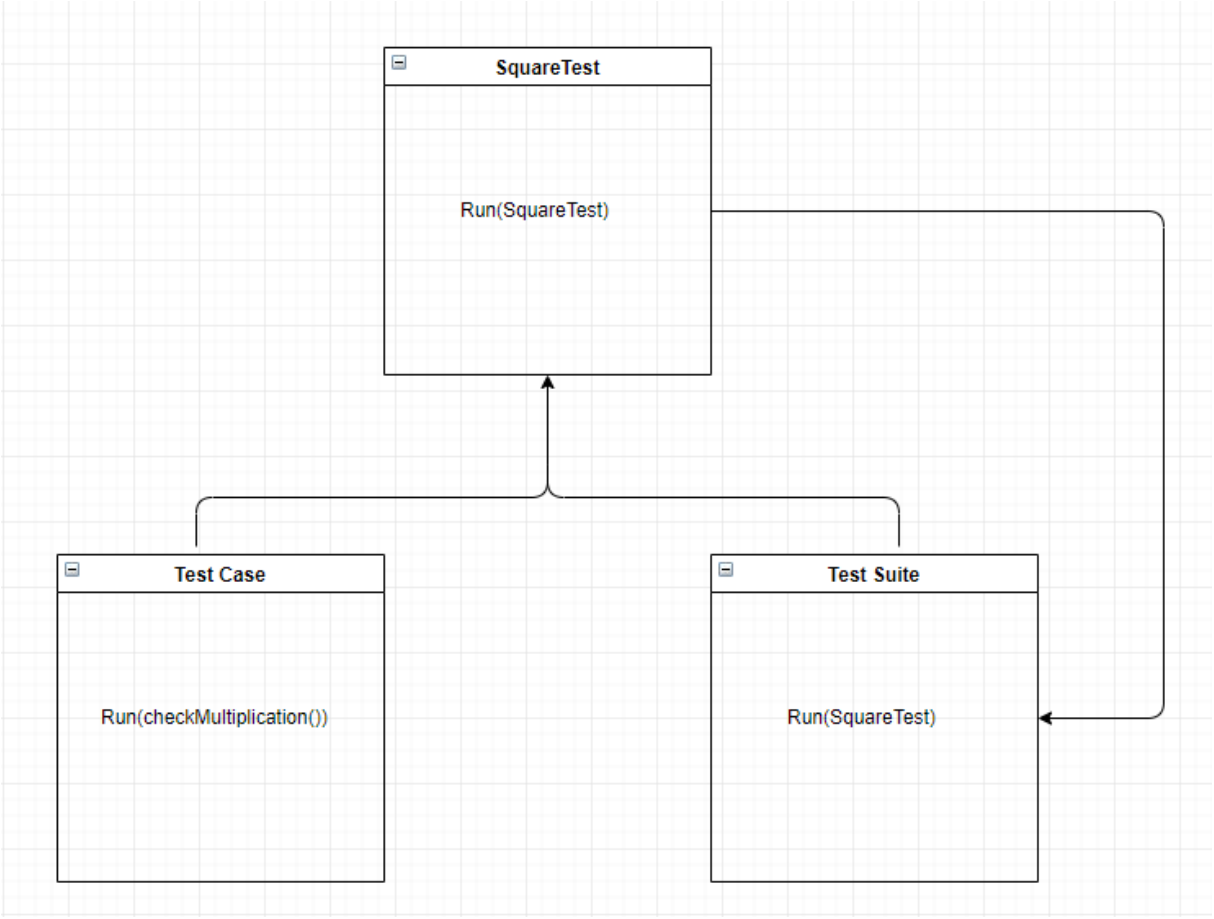
- JUnit 3.8-as ill. korábbi verzióiban a junit.framework alatt találhatók
- JUnit 4-es ill. későbbi verzióiban org.junit alatt találhatók”. [2]

„Ellenkezőleg az előző JUnit verziókhoz képest a JUnit 5 3 különböző al moduleból tevődik össze. JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage. JUnit Platform alapjául szolgál a tesztelés frameworknek a JVM-en belül. Meghatározza a TestEngine API fejlesztésést a tesztelés során. JUnit Jupiter kombinációja az új programozási – és bővítési modellnek tesztek és bővítmények írásához JUnit 5 alatt. JUnit Vintage lehetőséget nyújt JUnit 3 és JUnit 4 alapú tesztek futtatására. A JUnit 5 Java 8 vagy attól nagyobb verziójú projektekben használható.

Annotációk	Leírás
@Test	Azonosítja, hogy egy metódus teszt metódus. JUnit 4-el ellentétben itt a @Test annotáció nem deklarál semmilyen attribútumot.
@RepeatedTest	Azonosítja, hogy a metódus egy teszt metódus többszörös ismétlésre.
@BeforeEach	Azonosítja, hogy a metódus mindenképpen végrehajtódik minden egyes @Test, @RepeatedTest előtt ugyan azon classon belül.
@AfterEach	Azonosítja, hogy a metódus mindenképpen végrehajtódik minden egyes @Test, @ReapetedTest után ugyan azon classon belül.

[3]
”

UML diagram:



Gyakorlati feladat:

A felhasználótól bekérek két adatot (Téglatest A és B oldalát) majd ezen adatok megadása után kiszámolom a terület nagyságát.

Forráskód:

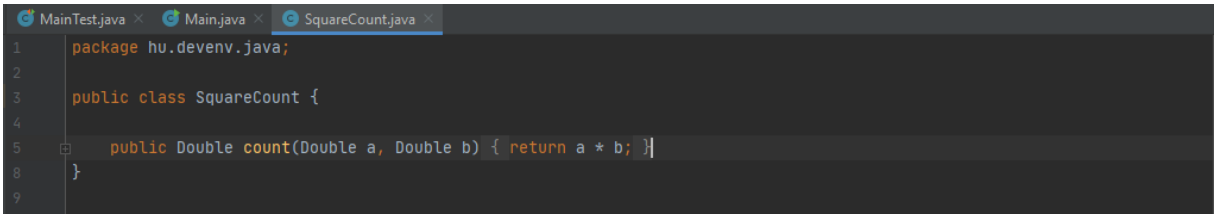
```java
package hu.devenv.java;

import ...

public class Main {

    public static void main(String[] args) {
        System.out.print("Negyzet A oldala: ");
        Scanner in = new Scanner(System.in);
        Double a = in.nextDouble();
        System.out.print("\nNegyzet B oldala: ");
        Double b = in.nextDouble();

        SquareCount sc = new SquareCount();
        System.out.println("Negyzet terulete: " + Math.round(sc.count(a, b)));
    }
}
```



```java
package hu.devenv.java;

public class SquareCount {

    public Double count(Double a, Double b) { return a * b; }
}
```

```java
package hu.devenv.test.java;

import hu.devenv.java.SquareCount;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

class MainTest {

    @Test
    public void checkMultiplication() {
        SquareCount test = new SquareCount();
        System.out.println("======TEST STARTED=======");
        Assertions.assertTrue(!Double.isNaN( v: 3.00), message: "!Double.isNaN(3.00) test success");
        Assertions.assertTrue(!Double.isNaN( v: 5.00), message: "!Double.isNaN(5.00) test success");
        Assertions.assertEquals( expected: 15.00, test.count( a: 3.00, b: 5.00), message: "test.count(3.00, 5.00) test success");
        System.out.println("======TEST ENDED=======");
    }
}
```

Források:

1: https://hu.wikipedia.org/wiki/Egys%C3%A9gtesztel%C3%A9s
2: https://en.wikipedia.org/wiki/JUnit
3: https://junit.org/junit5/docs/5.0.2/user-guide/index.pdf