Test Driven Development

Author: Diego Barbieri 0001080333

Mock

Spezzata.java

```
package org.example;

import java.util.*;

public class Spezzata {
    ArrayList <Punto> punti;


    public Spezzata() {
        this.punti = new ArrayList<Punto>();
    }

    public void aggiungi(Punto punto) {
        punti.add(punto);
    }

    public double quantiPunti(){ return punti.size();}

    public double lunghezza() {
        double somma=0;
        for(int i=1; i<punti.size(); i++) {
            Punto p= punti.get(i-1);
            somma+=p.distanzaDa(punti.get(i));
        }
        return somma;
    }

    @Override
    public String toString() {
        List<String> strings = new ArrayList<>(punti.size());
        for (Object object : punti) {
            strings.add(Objects.toString(object, null));
        }
        return Arrays.toString(strings.toArray());
    }
}
```

SpezzataTest.java

```java
package org.example;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

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

class SpezzataTest {
    @BeforeAll
    static void init(){
        System.out.println("inizio test");
    }
    Spezzata s;

    @BeforeEach
    void setUp() {
        // clean the array
        this.s = new Spezzata();
    }

    @Test
    void aggiungi() {
        s.aggiungi(new Punto(1,1));
        assertEquals(1, s.quantiPunti());
        assertEquals("[(1.0, 1.0)]", s.toString());
    }

    @Test
    void quantiPunti() {
        assertEquals(0, s.quantiPunti());
    }

    @Test
    void lunghezza() {
        s.aggiungi(new Punto(1,1));
        s.aggiungi(new Punto(2,2));
        s.aggiungi(new Punto(3,3));
        assertEquals(2*Math.sqrt(2), s.lunghezza(), 0.01);
    }

    @Test
    void testToString() {
        assertEquals("[]", s.toString());
        s.aggiungi(new Punto(1,1));
        assertEquals("[(1.0, 1.0)]", s.toString());
```

```
    }

    @Test
    void exception() {

        assertThrows(NullPointerException.class, () -> {
            s = null;
            this.s.lunghezza();
        });
    }
}
```

Considerazioni Personali

Cosa ne penso?

A mio parere, la lezione di oggi è stata illuminante! Sono sempre stato incuriosito dal campo della programmazione dei test automatici, ma per mancanza di tempo e a causa di alcuni stereotipi non mi sono mai cimentato pienamente. Dopo questa esperienza ho compreso l'importanza di dedicare tempo anche ai test (e al refactoring). Tra le considerazioni e preoccupazioni riguardanti questa tecnica di programmazione, ho riflettuto su diversi fattori: - partire dal pensare ai test, in alcuni contesti, possa essere limitante per il raggiungimento della soluzione ottimale o, comunque, più laborioso. - Il tempo utilizzato per testare è a tutti gli effetti un investimento - I test coprono solo la parte di codice per cui sono stati pensati, il che può portare a un codice **apparentemente** privo di bug. ## Pensate sia adatta al vostro modo di lavorare?

Penso di iniziare a utilizzarla regolarmente come buona pratica di programmazione, specialmente nella creazione di progetti complessi, dove una "semplice" modifica a codice obsoleto può risultare disastrosa. Dalla mia recente esperienza personale mi sono imbattuto, durante la realizzazione del progetto per il corso di tecnologie web, nel dover ricorreggere errori al codice già precedentemente affrontati. L'ausilio di uno strumento automatico di testing mi avrebbe permesso di individuare e risolvere il problema con estrema facilità, specialmente prima di effettuare *merge* nei rami di production. ## Pensate di usarla se non costretti dalle circostanze?

Come già accennato nel punto precedente, può risultare decisiva, in termini di tempo risparmiato nel debug, la presenza di controlli automatici su codice già scritto.

Modulo Stack scritto in TDD

https://github.com/Diebbo/stack-go-test