Unit testing con JUnit

2 Aprile 2019

Scaglione San Pietro

RESPONSABILI

Giovanni Meroni
Amarildo Likmeta

TUTOR

Marco Bacis
Valentina Deda

Test di Unità

- Per “test di unità” si intende un frammento di codice volto a verificare una specifica “unità di lavoro” del sistema, accertandosi che una singola assunzione sul comportamento di tale unità venga rispettata
  - Un’unità di lavoro può essere un singolo metodo, una singola classe, oppure più classi a patto che lavorino insieme in modo coordinato.
- Se due classi non hanno relazioni dirette tra loro, non ha senso raggrupparle in una singola unità di lavoro
- La percentuale di codice che viene testato è chiamata copertura del test
- Il test di unità non è indicato per testare interfacce utente complesse o l’interazione tra componenti.
  - Per questo tipo di verifiche si usa il test di integrazione

Proprietà del test di unità

- Un test di unità dovrebbe:
- Essere completamente automatizzato
- Utilizzare come input valori noti a priori
- Verificare che l'output corrisponda ad un valore atteso
- Non presentare side effects
- Non essere influenzato, nè influenzare, l'ordine di esecuzione degli altri test
- Verificare una singola funzionalità logica dell'applicazione
- Prevedere due test per ogni singola funzionalità
- Prevedere un test positivo, che ne verifichi il comportamento con dati validi
- Prevedere un test negativo, che ne verifichi il comportamento con dati non validi

JUnit

- Framework per realizzare test di unità su applicazioni Java
- Offre le seguenti funzionalità:
  - Fixtures: routine per il setup dell'ambiente di test e la pulizia a test completato
  - Test suites: il codice di verifica effettivo
  - Test runners: il codice per l'esecuzione del test
  - Altre classi di supporto (import)

JUnit

- Fixtures:
  - Annotazione @Test, indica che il metodo associato è un test
  - Metodo setUp() o annotazione @Before, eseguito prima di ogni test
  - Metodo tearDown() o annotazione @After, eseguito dopo ogni test
  - Metodo setUpClass() o annotazione @BeforeClass, eseguito prima di ogni metodo della classe di test
  - Metodo tearDownClass() o annotazione @AfterClass, eseguito dopo ogni metodo della classe di test
- Test suites:
  - Annotazione @RunWith(Suite.class), definisce test suite correlate
  - Annotazione @SuiteClasses(TestClass1.class, …), definisce quali classi di test devono essere eseguite insieme

JUnit

- Test runners: il codice per l'esecuzione del test
  - Classe Result: contiene il risultato del test
  - Metodo Result.getFailures() restituisce cosa non è stato superato dal test
  - Metodo Result.wasSuccessful() verifica se il test è stato superato (pienamente)
- Altre classi di supporto (import):
  - Classi Assert: verificano se i risultati sono corretti
  - TestCase: permette di raggruppare le fixtures, utilizzandole così per più classi di test
  - TestResult: permette di raccogliere e gestire i risultati del test mentre questo è in esecuzione

Metodi classi Assert

- fail(): fa fallire sempre il test (utile quando si verificano eccezioni anomale)
- assertTrue(parametro): verifica che il parametro passato sia vero
- assertFalse(parametro): verifica che il parametro passato sia falso
- assertEquals(valore, parametro): verifica che il parametro passato sia uguale al valore indicato
- assertNotEquals(valore, parametro): verifica che il parametro passato non sia uguale al valore indicato
- assertArrayEquals(arrayAtteso, arrayPassato): verifica che l'array passato contenga gli stessi valori dell'array atteso
- assertNull(parametro): verifica che il parametro non sia inizializzato
- assertNotNull(parametro): verifica che il parametro sia inizializzato

Esempio classe di test

```
package it.polimi.ingsw;
import static org.junit.Assert.*;
import org.junit.Test;

public class TestWalletClass {
	@Test
	public void testMultiplication() {
		Wallet wallet= null;
		try {
			wallet = new Wallet(5);
		} catch (NotEnoughMoneyException e) {
			fail();
		}
		wallet.times(2);
		assertEquals(10, wallet.getAmount());
	}
}
```

Permette di definire le assezioni

Definisce testMultiplication come test

Se si arriva a questo punto, il test deve sempre fallire

Verifica che il risultato sia uguale al valore indicato

Test di unità: consigli pratici

- Mantenere le classi di test chiaramente separate dalle classi dell'applicazione
- Non fare setup del test all'interno del costruttore della classe di test
- L'ordine di esecuzione dei test non deve influenzare i test
- Non scrivere test case con side effects
- Utilizzare percorsi relativi (p.e., .\supportData.txt) per caricare file di dati
- Includere nel test tutti i dati necessari per la sua esecuzione
- Dare nomi significativi ai test:
- Le classi test dovrebbero iniziare tutte col prefisso Test (p.e., TestRoom.java)
- I metodi delle classi test dovrebbero far capire cosa viene testato (p.e., testRollDice)

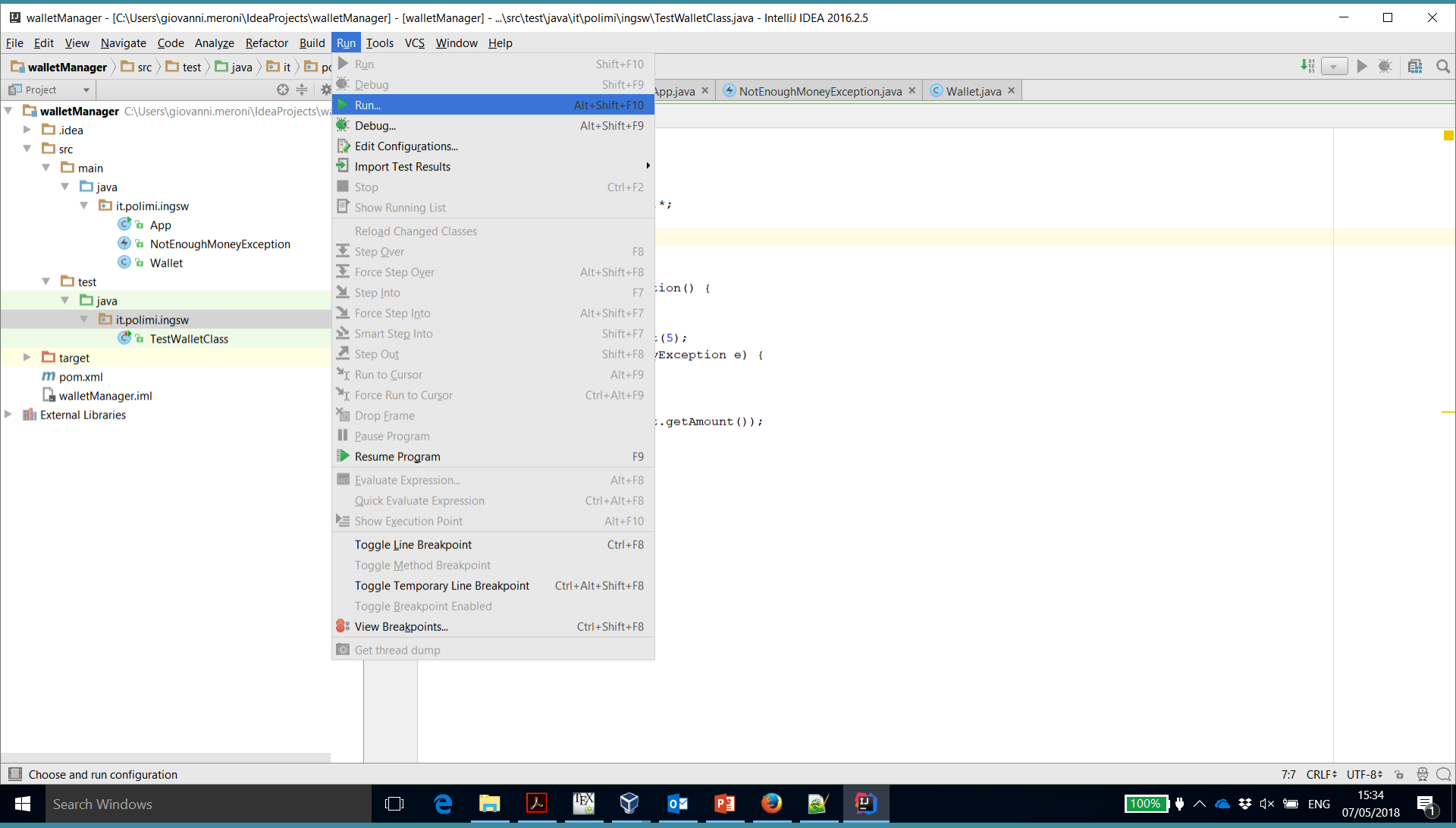
Test di unità: consigli pratici

- Utilizzare i metodi assert e fail in modo corretto
- Commentare i test con JavaDoc (più avanti in questo laboratorio)
- Scrivere test semplici e rapidi
- **Non** testare le interfacce utente (GUI, CLI, etc…)
- **Non** testare le comunicazioni di rete
- Testare metodi privati per via indiretta testando I metodi pubblici che ne fanno uso (p.e., se il codice del metodo pubblico foo() chiama il metodo privato bar(), testando foo() è possible per via indiretta testare anche bar()
- Quanto più il comportamento di un frammento di codice risulta visibile dall'esterno, tanto più andrà testato approfonditamente

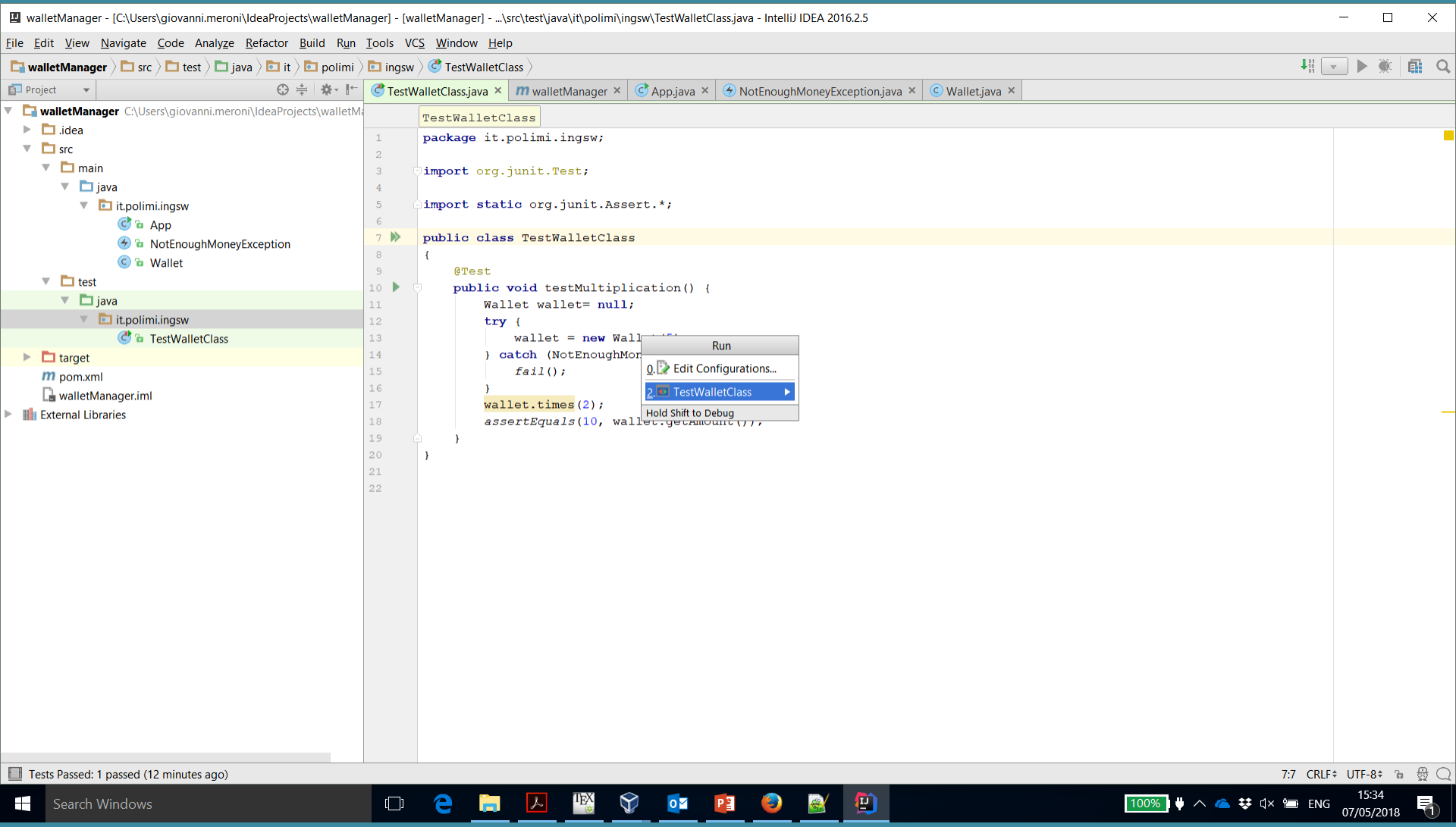
Integrazione JUnit – IntelliJ IDEA

- Mantenendo visualizzata una classe di test, selezionare Run → Run…
- Selezionare il nome della classe di Test
- Controllare nella sezione in basso che i test si comportino come previsto
- È possibile modificare la configurazione dei test da Run → Edit configurations
- Si può lanciare velocemente un singolo test facendo clic sul triangolo di fianco al nome del metodo e selezionando Run

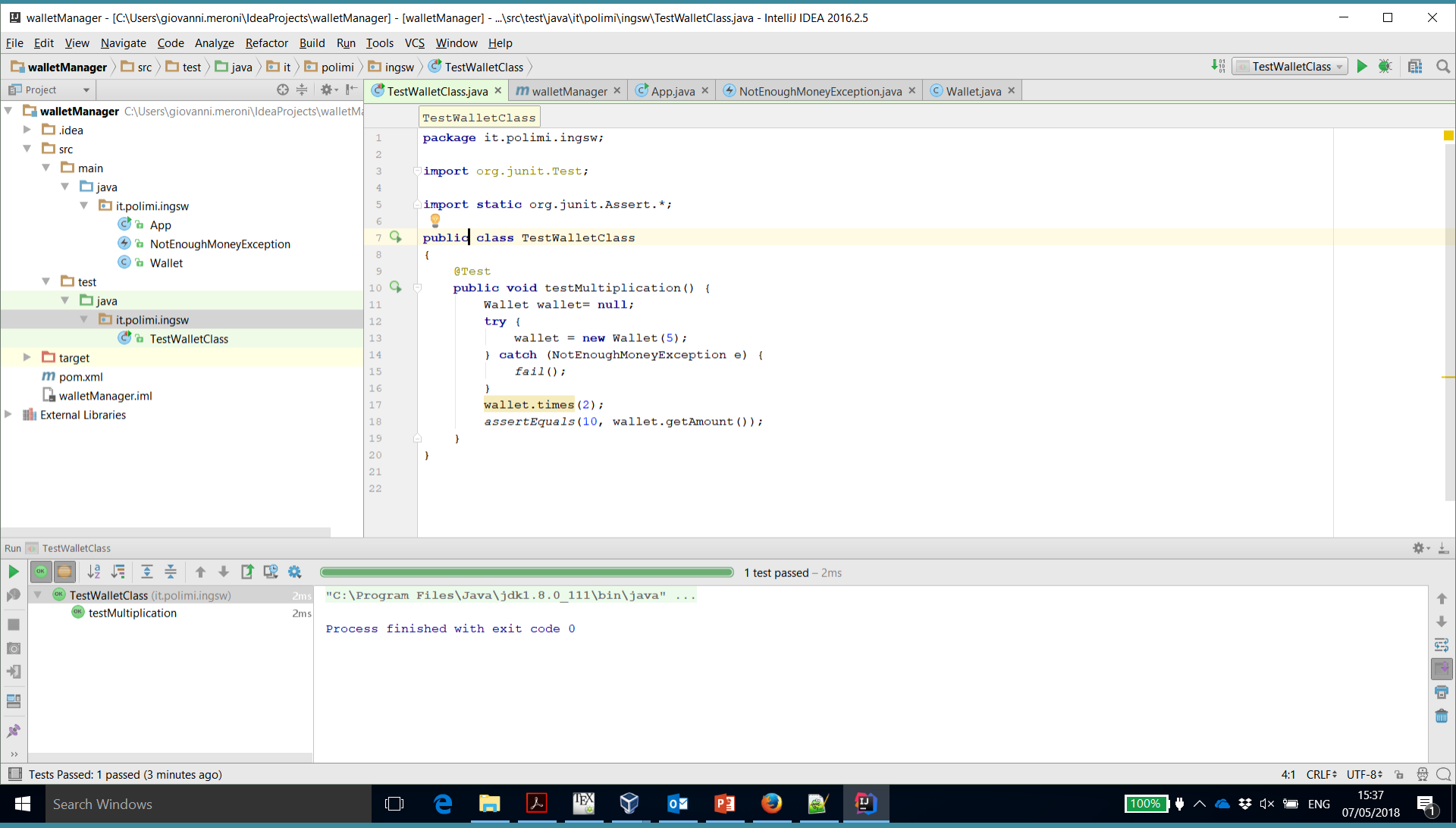
Integrazione JUnit – IntelliJ IDEA



Integrazione JUnit – IntelliJ IDEA



Integrazione JUnit – IntelliJ IDEA



Integrazione JUnit – IntelliJ IDEA