

# Testivetoinen ohjelmistokehitys

# Ohjelman luominen pienin askelin

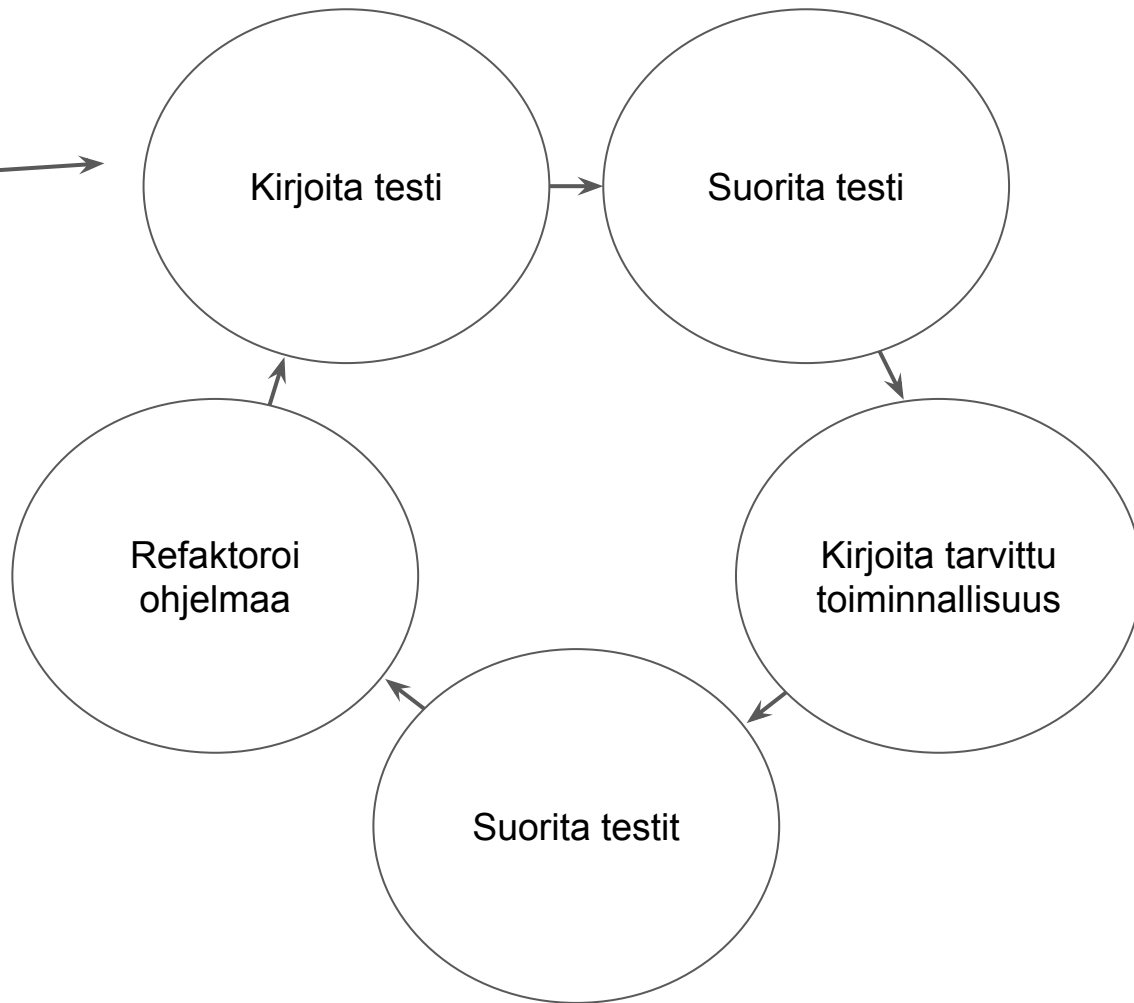
1. Kirjoita testi, joka testaa ohjelmalle myöhemmin lisättävää toiminnallisuutta.
2. Suorita testi. Testin ei tule mennä läpi.
  - Mikäli testi menee läpi, siirry kohtaan 1.
3. Kirjoita ohjelmaan testin läpäisemiseen tarvittava toiminnallisuus.
4. Suorita testit.
  - Mikäli testit eivät mene läpi, siirry kohtaan 3 ja täydennä toiminnallisuutta.
5. Refaktoroi.
  - Mikäli ohjelma on valmis, lopeta.
  - Muuten, siirry kohtaan 1.

# Ohjelman luominen pienin askelin

1. Kirjoita testi, joka testaa ohjelmalle myöhemmin lisättävää toiminnallisuutta.
2. Suorita testi. Testin ei tule mennä läpi.
  - Mikäli testi menee läpi, siirry kohtaan 1.
3. Kirjoita ohjelmaan testin läpäisemiseen tarvittava toiminnallisuus.
4. Suorita testit.
  - Mikäli testit eivät mene läpi, siirry kohtaan 3 ja täydennä toiminnallisuutta.
5. Refaktoroi.
  - Mikäli ohjelma on valmis, lopeta.
  - Muuten, siirry kohtaan 1.

*Refaktoroinnissa koodia siistitään säilyttäen samalla ohjelman toiminnallisuus. Siistiminen sisältää mm. luettavuuden parantamista sekä esimerkiksi ohjelman pilkkomista pienempiin metodeihin ja luokkiin.*

alku



# Hyötyjä

- Pakottaa miettimään toiminnallisuutta ennen ohjelman koodausta.
- Johtaa ylläpidettävään rakenteeseen, sillä ohjelmaa tehdään pienissä osissa refaktoroiden.
- Lopputuotoksessa testit, jonka ansiosta ohjelman jatkokehitys helpompaa: muutoksen yhteydessä voi tarkastaa menikö aiempi toiminnallisuus rikki.
- Vähemmän bugeja tuotannossa.

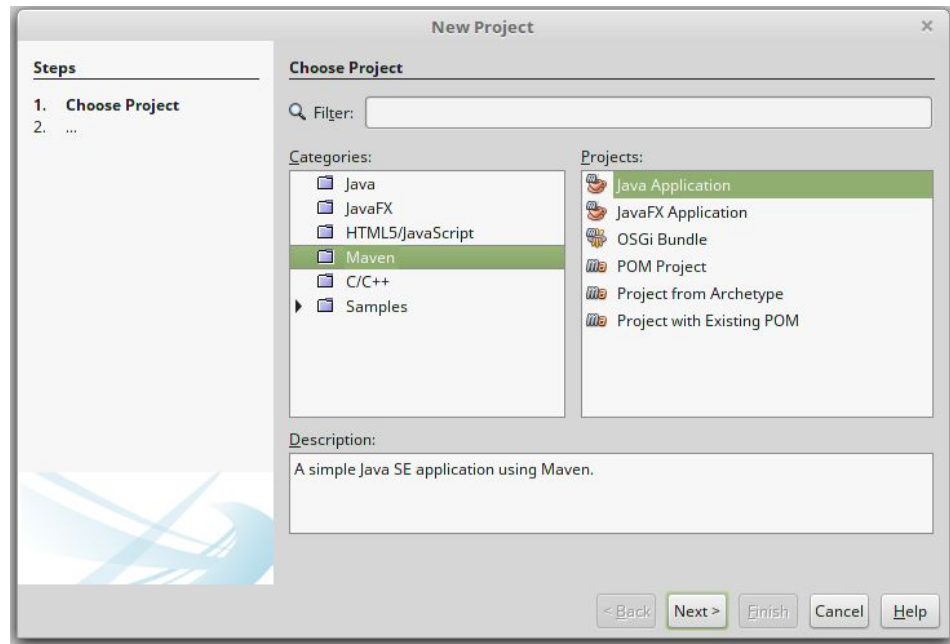
**Esimerkki: tehtävienhallinta**

# Osa 1. Projektin luominen ja JUnit-kirjaston käyttöönotto

Tarkastellaan testivetoista ohjelmistokehitystä tehtävien hallintaan tarkoitetun sovelluksen kannalta.

**Tehtävien hallintasovellukseen halutaan mahdollisuus tehtävien listaamiseen, lisäämiseen, tehdyksi merkkaamiseen sekä poistamiseen.**

Aloitetaan sovelluksen kehitys luomalla tyhjä projekti. Tämä tapahtuu valitsemalla NetBeansissa File -> New Project. Valitaan projektin kategoriaksi "Maven" ja projektin tyyppiä "Java Application".





Tämän jälkeen täytetään uuden projektin tiedot. Asetetaan projektin nimeksi “tehtavienhallinta”. Projektin sijainti kertoo missä päin tiedostojärjestelmää projektin tiedostot sijaitsevat.

Pidetään kohdassa Package-oleva kenttä tyhjänä.

**New Java Application**

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name: tehtavienhallinta

Project Location: /home/avihavai/NetBeansProjects

Project Folder: /home/avihavai/NetBeansProjects/tehtavienhallinta

Artifact Id: tehtavienhallinta

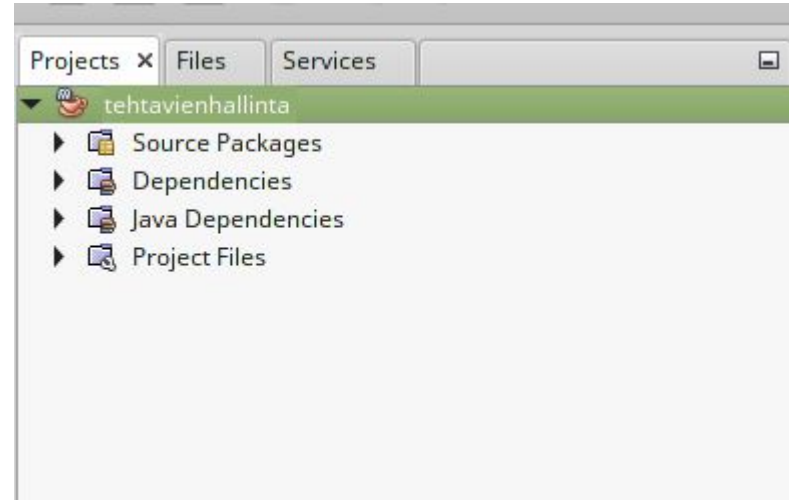
Group Id: hajautustaulu

Version: 1.0-SNAPSHOT

Package:  (Optional)

< Back Next > Finish Cancel Help

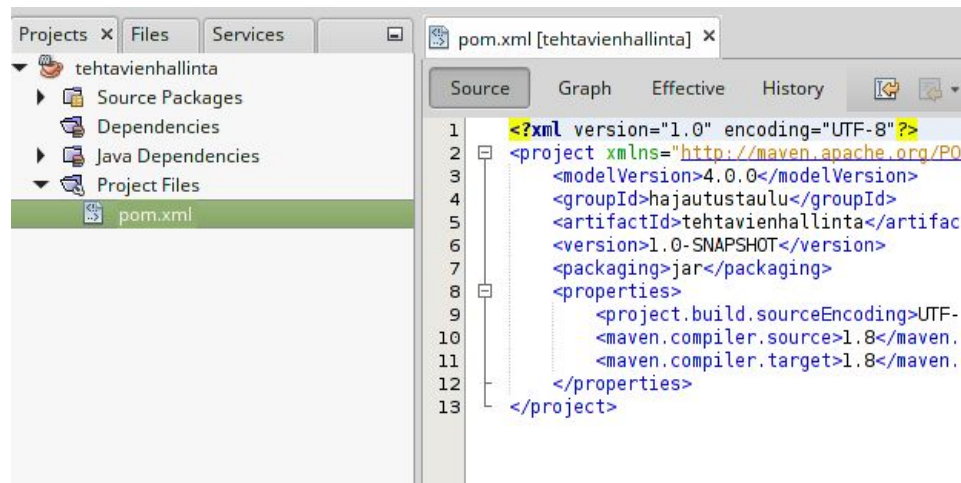
Kun painamme Finish, uusi projekti on luotu.  
Projekti näkyy NetBeansissa vasemmalla  
laidalla.



Kun painamme Finish, uusi projekti on luotu. Projekti näkyy NetBeansissa vasemmalla laidalla.

Lisätään seuraavaksi yksikkötestien kirjoittamiseen käytetty JUnit-kirjasto (tämä on valmiina TMC:stä ladattavissa tehtäväpohjissa).

Klikataan kansio Project Files auki ja tuplaklikataan tiedostoa pom.xml. Tiedosto pom.xml kertoo projektimme rakenteesta sekä sen käyttämistä kirjastoista -- tähän tutustutaan tarkemmin Ohjelmoinnin jatkokurssilla.

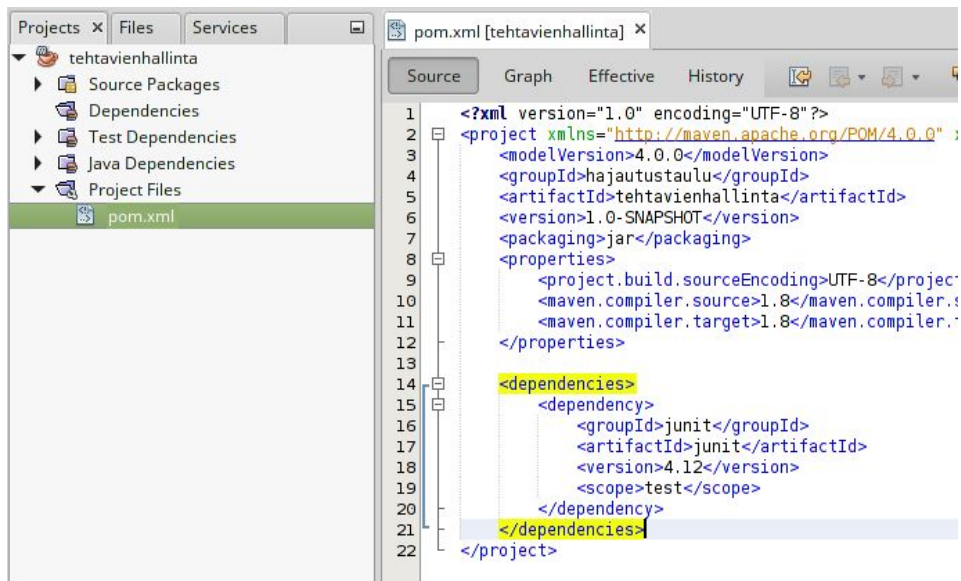


Kopioidaan JUnit-kirjasto ennen riviä `</project>`:

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

Kun yllä oleva sisältö on kopioitu pom.xml-tiedostoon (ennen riviä `</project>`), tallennetaan tiedosto.

Tämä tuo ohjelman käyttöön JUnit-kirjaston ja mahdollistaa yksikkötestien kirjoittamisen.

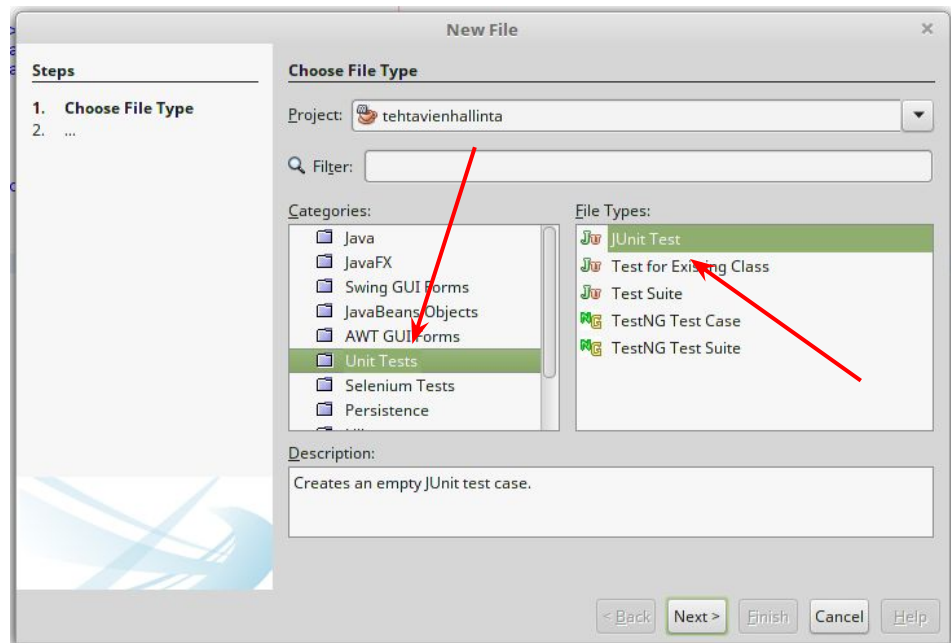


## Osa 2. Yksikkötestiluokan luominen

Luodaan ensimmäinen yksikkötesti.  
Yksikkötestien luominen onnistuu klikkaamalla projektia oikealla hiirennapilla ja valitsemalla new -> Other.

Tämä avaa näkyville uuden tiedoston luomiseen käytetyn valikon. Valitse kategoriaksi “Unit Tests” ja luotavan tiedoston tyypiksi “JUnit Test”.

Paina tämän jälkeen “Next”.

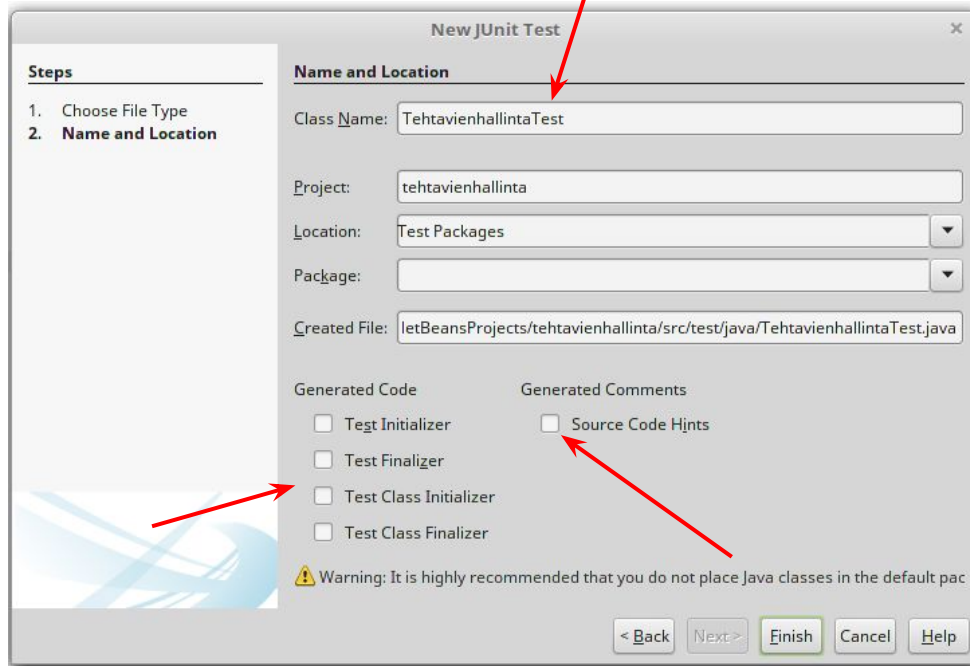


Eteen aukeaa yksikkötestitiedoston luomiseen käytettävä apuri.

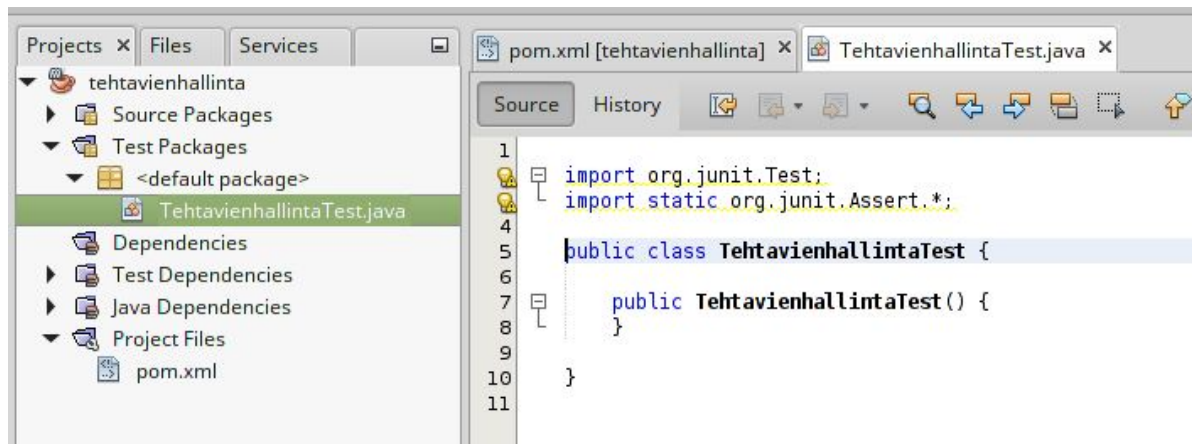
Asetetaan luokan nimeksi “TehtavienhallintaTest” ja valitaan ettei testiluokkaan tehdä koodia.

Huom! Varmista, että luokan nimi päättyy sanalla “Test”.

Paina lopulta Finish.



Nyt projektin kansioon “Test Packages” on ilmestynyt luokka “TehtavienhallintaTest”.





## Osa 3. Ensimmäinen yksikkötesti

Luodaan ensimmäinen testi. Testissä määritellään luokka Tehtavienhallinta, ja oletetaan, että luokalla on metodi tehtavalista, joka palauttaa tehtävälisan.

Testimetodi assertEquals saa parametrinaan kaksi arvoa -- ensimmäinen on odotettu arvo ja toinen on ohjelman palauttama arvo.

Tässä metodia käytetään tehtävälisan koon tarkastamiseen uuden tehtävälisan luomisen yhteydessä: uuden listan tulee olla tyhjä.

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

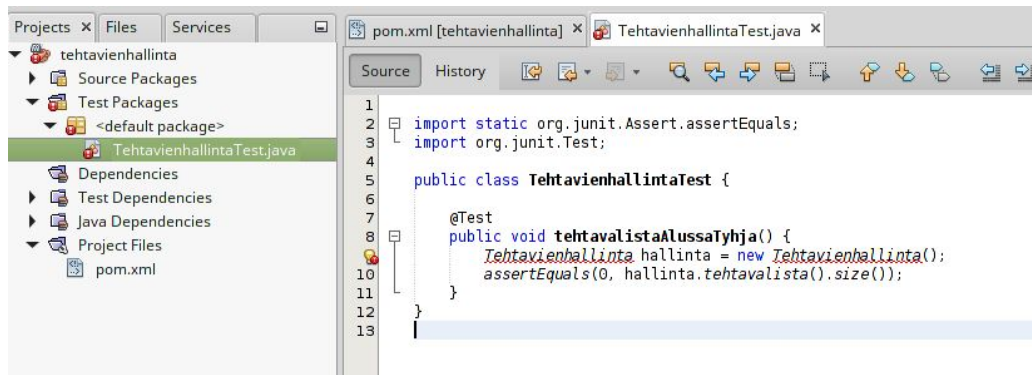
public class TehtavienhallintaTest {

    @Test
    public void tehtavalistaAlussaTyhja() {
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        assertEquals(0, hallinta.tehtavalista().size());
    }
}
```

Testi on luokassa TehtavienhallintaTest. Kun testi on luotu, nähdään suoraan ettei se mene läpi sillä testin testaama luokka Tehtavienhallinta puuttuu.

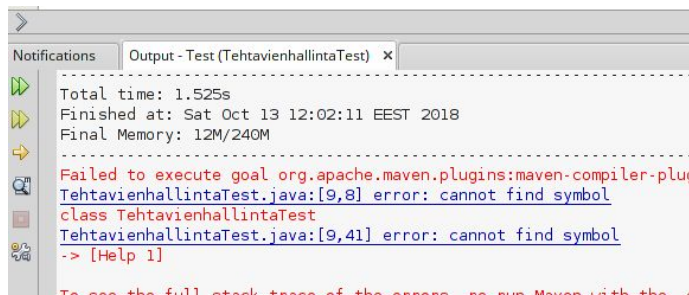
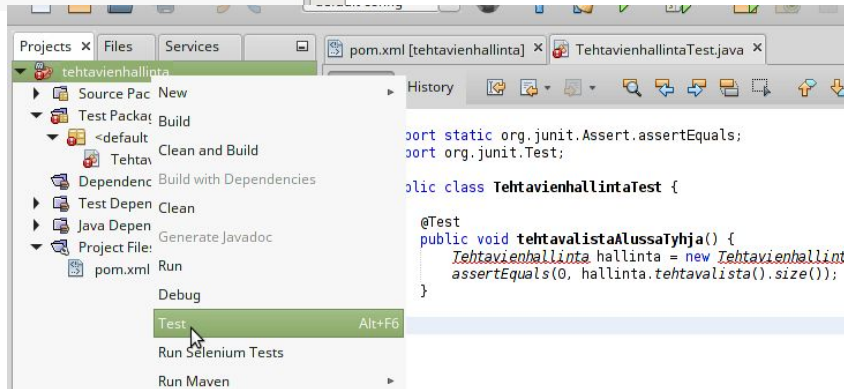
Suoritetaan testit silti. Tämä tapahtuu klikkaamalla projektia hiiren oikealla napilla ja valitsemalla "Test"

Näemme virheen "Failed to execute..."



The screenshot shows an IDE window with two tabs: 'pom.xml [tehtavienhallinta]' and 'TehtavienhallintaTest.java'. The left sidebar shows the project structure with 'TehtavienhallintaTest.java' selected under 'Test Packages'. The main editor shows the following code:

```
1
2 import static org.junit.Assert.assertEquals;
3 import org.junit.Test;
4
5 public class TehtavienhallintaTest {
6
7     @Test
8     public void tehtavalistaAlussaTyhja() {
9         Tehtavienhallinta hallinta = new Tehtavienhallinta();
10        assertEquals(0, hallinta.tehtavalista().size());
11    }
12
13 }
```



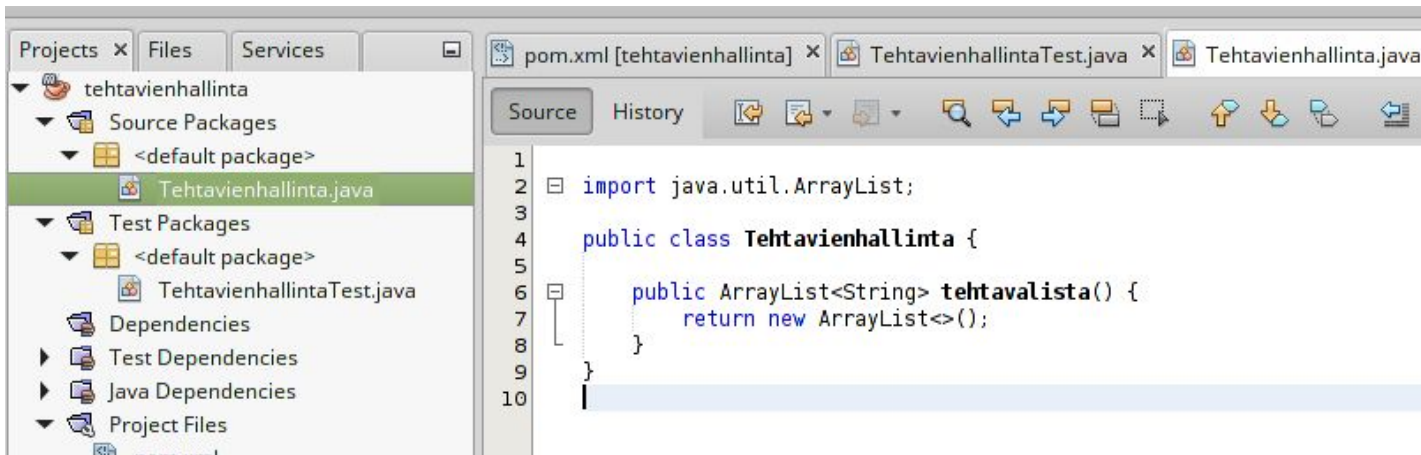
## Osa 4. Yksikkötestin vaatimusten täyttävän toiminnallisuuden toteutus

Luodaan luokka Tehtavienhallinta (luokka lisätään kansioon *Source Packages*) ja lisätään luokalle listan palauttava metodi tehtavalista.

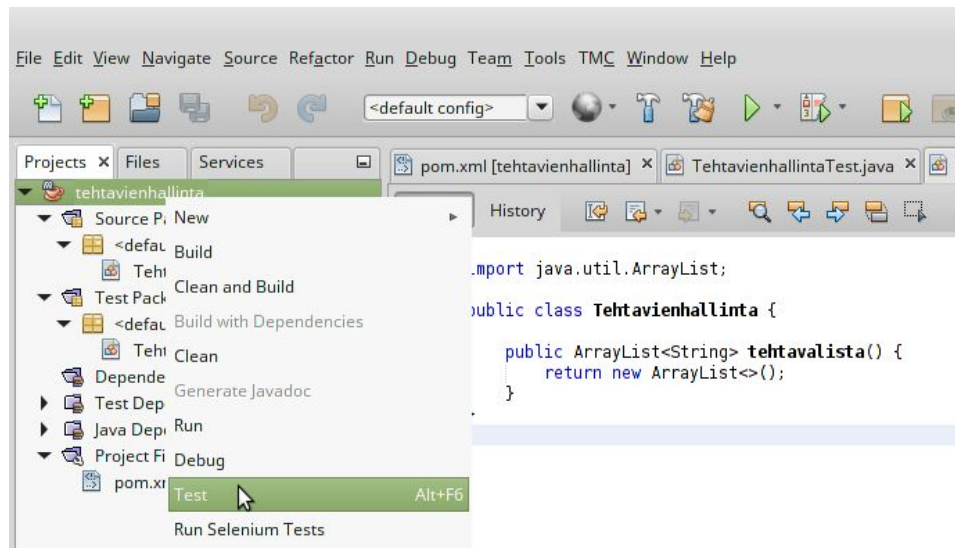
```
import java.util.ArrayList;

public class Tehtavienhallinta {

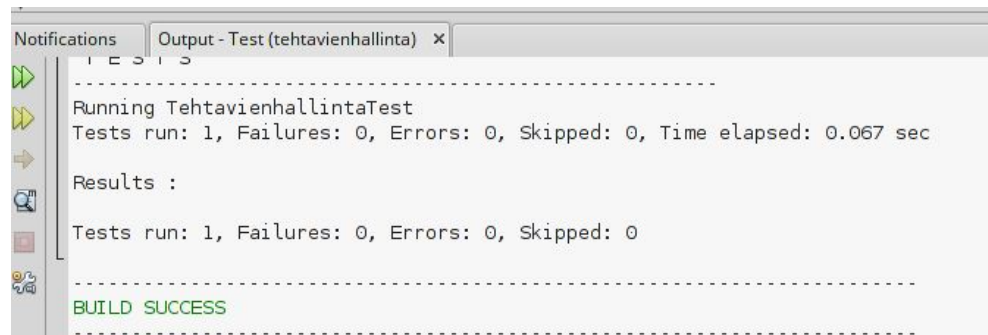
    public ArrayList<String> tehtavalista() {
        return new ArrayList<>();
    }
}
```



Suoritetaan projektiin liittyvät testit klikkaamalla projektia hiiren oikealla napilla ja valitsemalla "Test"



Projektiin liittyvät testit menevät läpi. Refaktoritavaa ei ole, joten jatkamme seuraavan testin kirjoitukseen.



## Osa 5. Toinen yksikkötesti ja siihen liittyvän toiminnallisuuden toteutus

Seuraavaksi luomme uuden testin, jossa tarkastellaan tehtävien lisäämiseen liittyvää toiminnallisuutta.

Testissä määritellään luokalle Tehtavienhallinta metodi lisää, joka lisää tehtävälisälle uuden tehtävän. Tehtävän lisäämisen onnistuminen tarkastetaan tehtavalista-metodin koon kasvamisen kautta.

Testi ei toimi lainkaan, sillä luokasta Tehtavienhallinta puuttuu lisää-metodi.

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

public class TehtavienhallintaTest {

    @Test
    public void tehtavalistaAlussaTyhja() {
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        assertEquals(0, hallinta.tehtavalista().size());
    }

    @Test
    public void tehtavanLisaaaminenKasvattaaListanKokoaYhdella() {
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        hallinta.lisaa("Kirjoita testi");
        assertEquals(1, hallinta.tehtavalista().size());
    }
}
```



Lisätään luokkaan Tehtavienhallinta metodi lisää ja suoritetaan testit. Metodi ei tässä vaiheessa tee vielä mitään.

Alussa lisäämämme testi menee läpi, mutta edellä lisätty testi ei.

Virheilmoitus on “Failed: expected: <1> but was: <0>” eli testi odotti arvoa 1 mutta sai arvon 0.

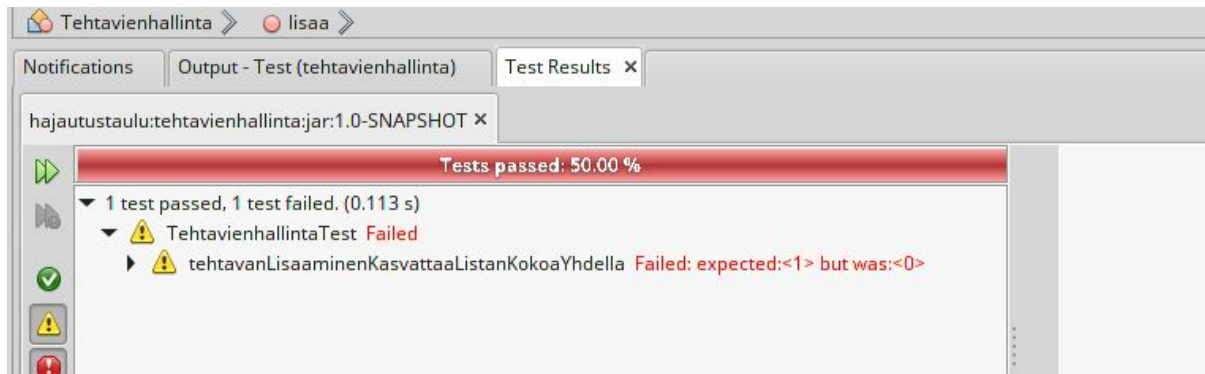
```
import java.util.ArrayList;

public class Tehtavienhallinta {

    public ArrayList<String> tehtavalista() {
        return new ArrayList<>();
    }

    public void lisää(String tehtava) {

    }
}
```



Muokataan luokan tehtävähallinta toiminnallisuutta siten, että luokalle luodaan oliomuuttujaksi tehtävät sisältävä lista.

Muokataan metodin lisää-toiminnallisuutta vain niin, että se läpäisee testin, mutta ei tee todellisuudessa haluttua asiaa.

Suoritetaan testit -- testit menevät läpi, joten voimme siirtyä seuraavan testin kirjoittamiseen.

```
import java.util.ArrayList;

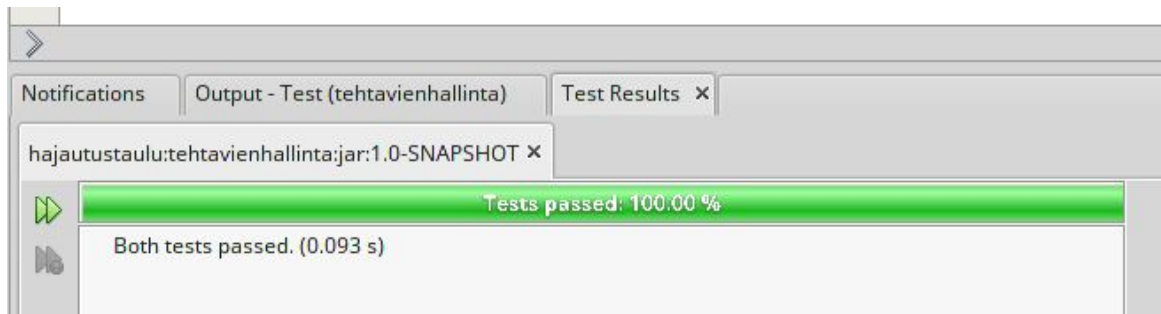
public class Tehtavienhallinta {

    private ArrayList<String> tehtavat;

    public Tehtavienhallinta() {
        this.tehtavat = new ArrayList<>();
    }

    public ArrayList<String> tehtavalista() {
        return this.tehtavat;
    }

    public void lisää(String tehtava) {
        this.tehtavat.add("Uusi");
    }
}
```



## Osa 6. Kolmas yksikkötesti ja siihen liittyvän toiminnallisuuden toteutus

Täydennetään testejä siten, että ne vaativat, että lisätyn tehtävän tulee olla listalla.

JUnit-kirjaston tarjoama metodi `assertTrue` vaatii, että sille parametrina annettu lauseke saa lopulta arvon `true`.

Kun ohjelmaan on lisätty uusi testi, testit eivät taaskaan mene läpi.

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

```
public class TehtavienhallintaTest {
```

```
    @Test
    public void tehtavalistaAlussaTyhja() {
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        assertEquals(0, hallinta.tehtavalista().size());
    }
```

```
    @Test
    public void tehtavanLisaminenKasvattaaListanKokoaYhdella() {
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        hallinta.lisaa("Kirjoita testi");
        assertEquals(1, hallinta.tehtavalista().size());
    }
```

```
    @Test
    public void lisattyTehtavaLoytyyTehtavalistalta() {
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        hallinta.lisaa("Kirjoita testi");
        assertTrue(hallinta.tehtavalista().contains("Kirjoita testi"));
    }
}
```

Noheva ohjelmoija muokkasi luokan Tehtavienhallinta toimintaa siten, että metodissa lisää lisättäisiin listalle aina merkkijono "Kirjoita testi".

Tämä johtaisi tilanteeseen, missä testit menisivät läpi, mutta sovellus ei vielä kukaan tarjoaisi toimivaa tehtävien lisäämistötoiminnallisuutta.

Muokataan luokkaa Tehtavienhallinta siten, että lisättävä tehtävä lisätään tehtävälistalle.

```
import java.util.ArrayList;

public class Tehtavienhallinta {

    private ArrayList<String> tehtavat;

    public Tehtavienhallinta() {
        this.tehtavat = new ArrayList<>();
    }

    public ArrayList<String> tehtavalista() {
        return this.tehtavat;
    }

    public void lisää(String tehtava) {
        this.tehtavat.add(tehtava);
    }
}
```

Huomaamme, että testiluokassa on toistoa -- hyvä hetki refaktoroinnille.

Siirretään Tehtävienhallinta testiluokan oliomuuttujaksi, ja alustetaan se jokaisen testin alussa.

Muuttujan alustaminen onnistuu luomalla testiluokkaan metodi alusta, missä muuttuja(t) alustetaan.

Metodille alusta annetaan annotaatio `@Before`, joka määrää että metodi suoritetaan ennen kunkin testimetodin suoritusta.

Testit menevät läpi refaktoroinnin jälkeenkin.

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

```
public class TehtavienhallintaTest {
```

```
    @Test
```

```
    public void tehtavalistaAlussaTyhja() {
```

```
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        assertEquals(0, hallinta.tehtavalista().size());
```

```
    }
```

```
    @Test
```

```
    public void tehtavanLisaminenKasvattaaListanKokoaYhdella() {
```

```
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        hallinta.lisaa("Kirjoita testi");
        assertEquals(1, hallinta.tehtavalista().size());
```

```
    }
```

```
    @Test
```

```
    public void lisattyTehtavaLoytyyTehtavalistalta() {
```

```
        Tehtavienhallinta hallinta = new Tehtavienhallinta();
        hallinta.lisaa("Kirjoita testi");
        assertTrue(hallinta.tehtavalista().contains("Kirjoita testi"));
```

```
    }
```

```
}
```

## Osa 7. Neljäs yksikkötesti ja siihen liittyvän toiminnallisuuden toteutus

Lisätään toiminnallisuus tehtävän tehdyksi merkkaamiseen.

Kun ohjelmaan on lisätty uusi testi, testit eivät taaskaan mene läpi.

```
// ...
@Test
public void tehtavanVoiMerkkataTehdyksi() {
    hallinta.lisaa("Uusi tehtävä");
    hallinta.merkkaaTehdyksi("Uusi tehtävä");
    assertTrue(hallinta.onTehty("Uusi tehtävä"));
}
// ...
```



Toiminnallisuutta varten tulee lisätä luokkaan Tehtavienhallinta metodit merkkaaTehdyksi ja onTehty.

Testit eivät tarkasta metodien toimintaa tarkemmin, joten lisätään toiminnallisuus aluksi hyvin kevyenä.

Testit menevät läpi.

```
// ...  
public void merkkaaTehdyksi(String tehtava) {  
  
}  
  
public boolean onTehty(String tehtava) {  
    return true;  
}  
// ...
```

## Osa 8. Viides yksikkötesti ja siihen liittyvän toiminnallisuuden toteutus

Olemme tähän mennessä tarkistaneet, että haluttu toiminnallisuus on olemassa, mutta emme ole juurikaan tarkastaneet epätoivotun toiminnan poissaoloa.

Mikäli testejä kirjoitettaessa keskitytään halutun toiminnallisuuden olemassaoloon, testit saattavat jäädä ohjelman toiminnallisuutta hyvin vähän tarkastelevaksi.

Kirjoitetaan seuraavaksi testi, joka tarkastaa, että tehdyksi merkkaamaton testi ei ole tehty.

Testit eivät mene läpi.

```
// ...
@Test
public void tehdyksiMerkkaamatonEiOleTehty() {
    hallinta.lisaa("Uusi tehtava");
    hallinta.merkkaaTehdyksi("Uusi tehtava");
    assertFalse(hallinta.onTehty("Joku tehtava"));
}
// ...
```

Toteutetaan taas testit täyttävä toiminnallisuus. Joudumme tekemään hieman isomman muutoksen ohjelmaan: lisätään luokkaan erillinen lista tehtäville, jotka on merkattu tehdyiksi.

Testit menevät taas läpi.

Sovelluksessa on kuitenkin muutamia kysymysmerkkejä. Pitäisikö tehtävälisäyksessä palautetut tehtävät merkitä jollain tavalla tehdyksi? Voiko tehtävän, joka ei ole tehtävälisäyksellä tosiaankin merkattu tehdyksi?

```
import java.util.ArrayList;

public class Tehtavienhallinta {

    private ArrayList<String> tehtavat;
    private ArrayList<String> tehdytTehtavat;

    public Tehtavienhallinta() {
        this.tehtavat = new ArrayList<>();
        this.tehdytTehtavat = new ArrayList<>();
    }

    public List<String> tehtavalista() {
        return this.tehtavat;
    }

    public void lisaa(String tehtava) {
        this.tehtavat.add(tehtava);
    }

    public void merkkaaTehdyksi(String tehtava) {
        this.tehdytTehtavat.add(tehtava);
    }

    public boolean onTehdy(String tehtava) {
        return this.tehdytTehtavat.contains(tehtava);
    }
}
```

Tehdään ensimmäinen hieman laajempi ohjelman sisäisen rakenteen korjaus. Tehtävä on selkeästi käsite, joten sille kannattanee luoda oma erillinen luokka.

Luodaan luokka Tehtava. Luokalla Tehtava on nimi sekä tieto siitä, onko tehtävä tehty.

```
public class Tehtava {  
    private String nimi;  
    private boolean tehty;  
  
    public Tehtava(String nimi) {  
        this.nimi = nimi;  
        this.tehty = false;  
    }  
  
    public String getNimi() {  
        return nimi;  
    }  
  
    public void setTehty(boolean tehty) {  
        this.tehty = tehty;  
    }  
  
    public boolean onTehty() {  
        return tehty;  
    }  
}
```

Muokataan tämän jälkeen luokan Tehtavienhallinta rakennetta siten, että luokka tallentaa tehtävät merkkijonojen sijaan Tehtava-olioina.

Huomaa, että luokan metodien määrittelyt eivät muutu, mutta niiden sisäinen toteutus muuttuu.

Vaikka tehty muutos muutti luokan Tehtavienhallinta sisäistä toimintaa merkittävästi, testit toimivat yhä.

Testivetoinen ohjelmistokehitys jatkuisi samalla tavalla kunnes toivottu perustoiminnallisuus olisi paikallaan

```
import java.util.ArrayList;

public class Tehtavienhallinta {

    private ArrayList<Tehtava> tehtavat;

    public Tehtavienhallinta() {
        this.tehtavat = new ArrayList<>();
    }

    public ArrayList<String> tehtavalista() {
        ArrayList<String> palautettavat = new ArrayList<>();
        for (Tehtava tehtava: tehtavat) {
            palautettavat.add(tehtava.getNimi());
        }

        return palautettavat;
    }

    public void lisaa(String tehtava) {
        this.tehtavat.add(new Tehtava(tehtava));
    }

    public void merkkaaTehdyksi(String tehdyksiMerkattavaTehtava) {
        for (Tehtava tehtava: tehtavat) {
            if (tehtava.getNimi().equals(tehdyksiMerkattavaTehtava)) {
                tehtava.setTehty(true);
            }
        }
    }

    public boolean onTehty(String tarkistettavaTehtava) {
        for (Tehtava tehtava: tehtavat) {
            if (tehtava.getNimi().equals(tarkistettavaTehtava)) {
                return tehtava.onTehty();
            }
        }

        return false;
    }
}
```

# Yhteenveto

Testivetoisessa ohjelmistokehitysmenetelmässä ohjelman toiminnallisuus rakennetaan pieninä askelina siten, että ohjelmoija kirjoittaa aina ensin uutta toiminnallisuutta tarkastelevan testin, jota seuraa testin läpäisevän ohjelmakoodin kirjoitus.