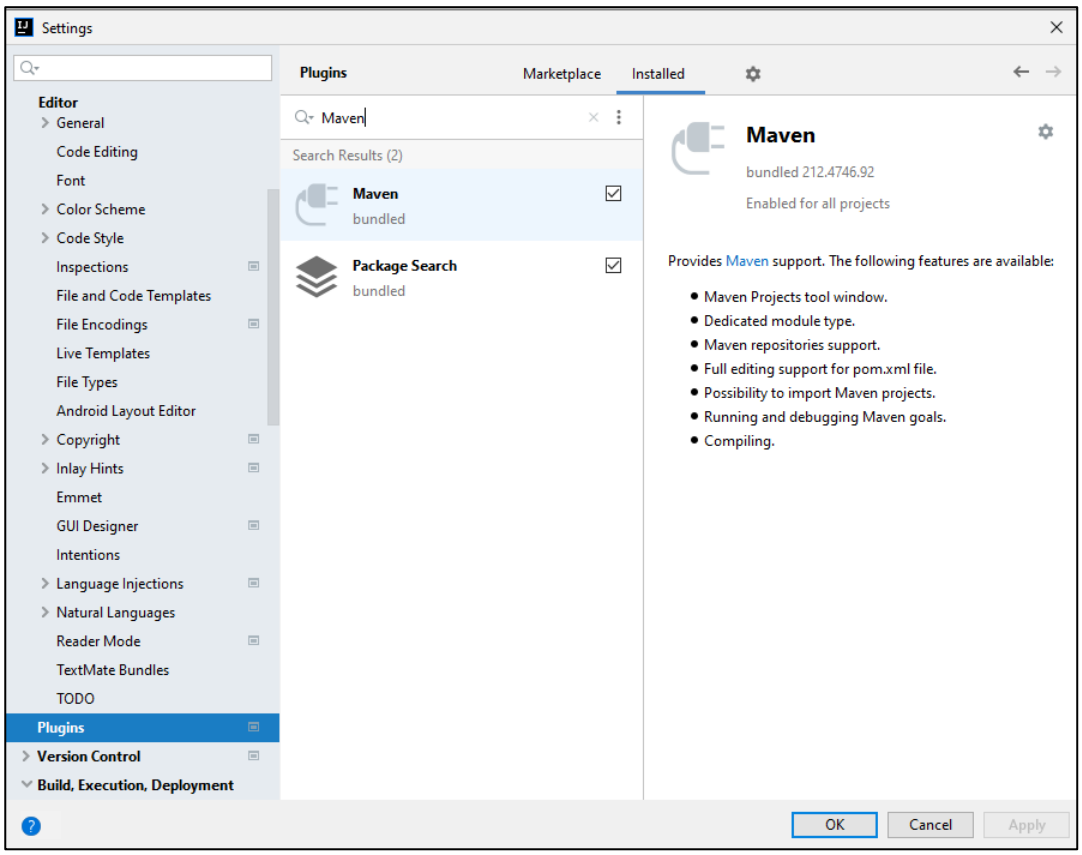
Lab: Unit Testing

This document defines the lab for "Java Advanced" course @ Software University.

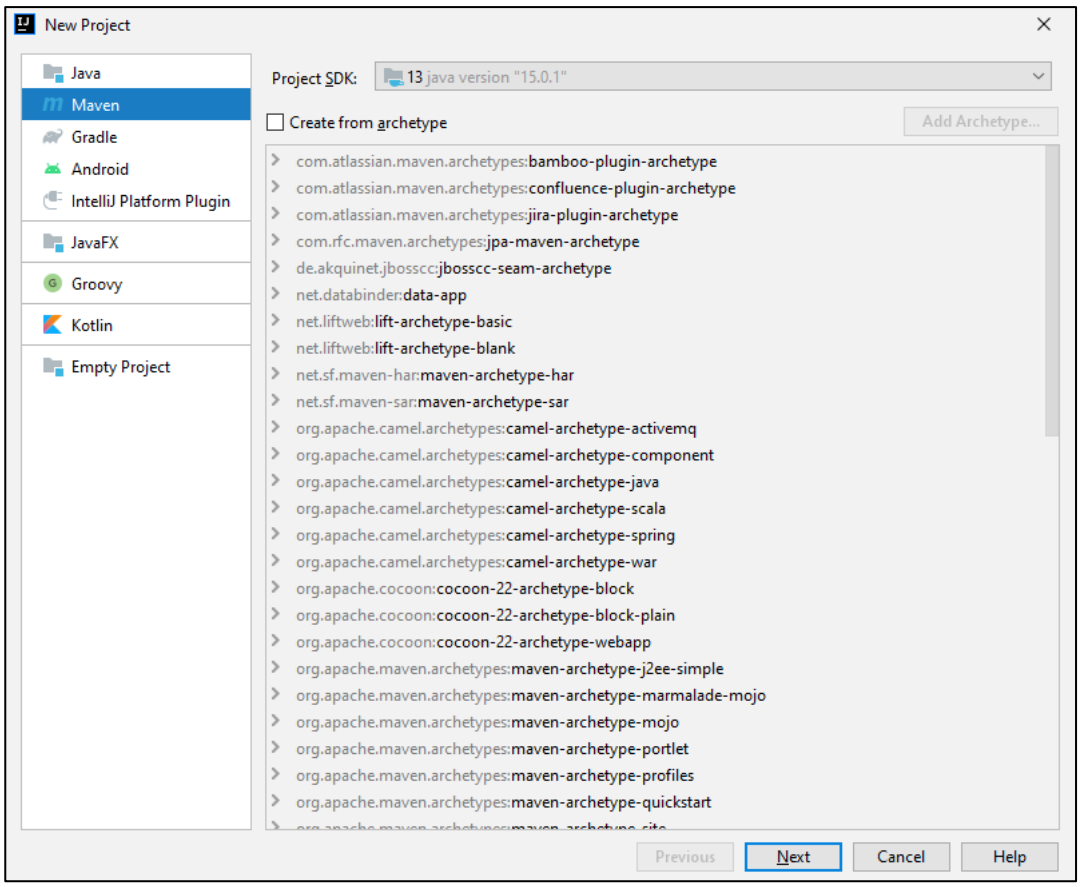
Part I: Unit Testing Basics

1. Create Maven Project

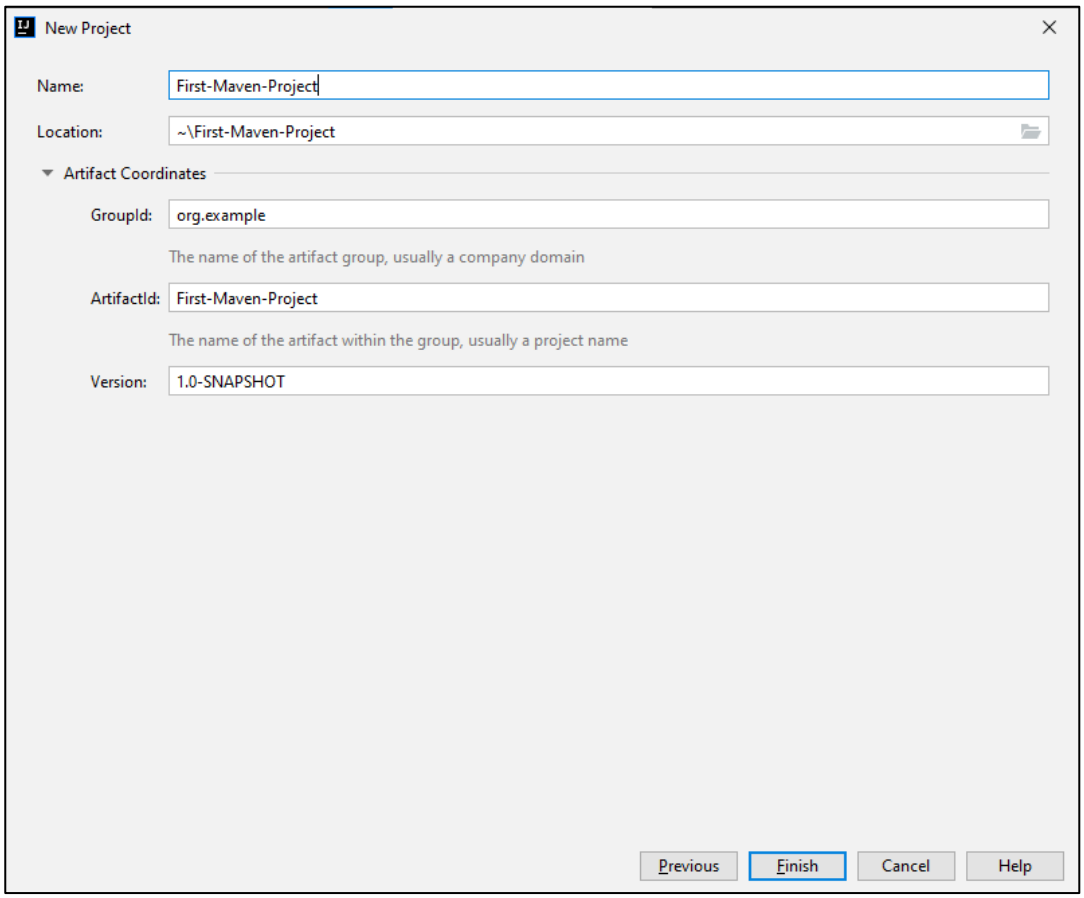
Maven is build automation tool that takes care of dependencies for your project. Before you can make one, make sure that you enable the plugin in IntelliJ **`[File → Settings → Plugins → Maven]`**



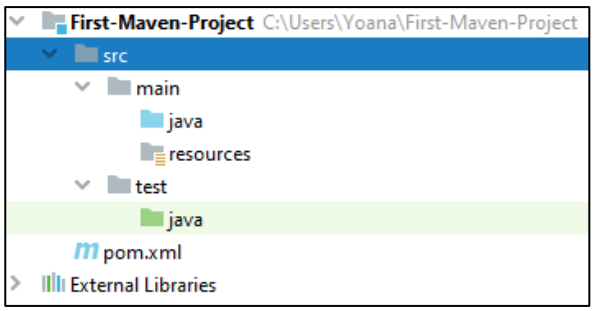
Now, you can create a Maven project.



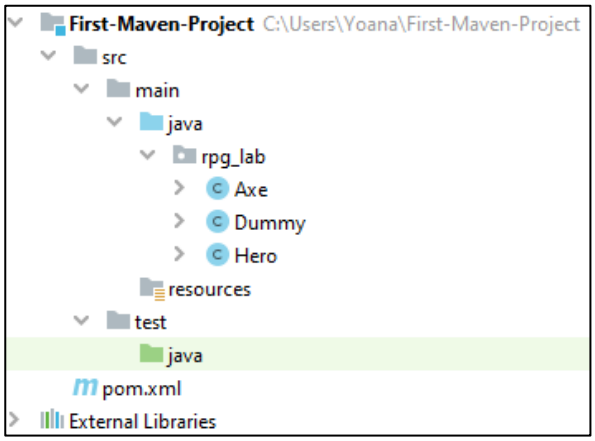
First, you give a name and location to your project. Group Id should be separated by dots, Artifact Id should be separated by hyphens:



If everything is ok, you should see the following project structure:



Copy the files provided and place them in a package inside **`src/main/java`** folder



2. Test Axe

In the **`test/java`** folder, create a package called **`rpg_lab`.**

Create a class **`AxeTest`.**

Create the following tests:

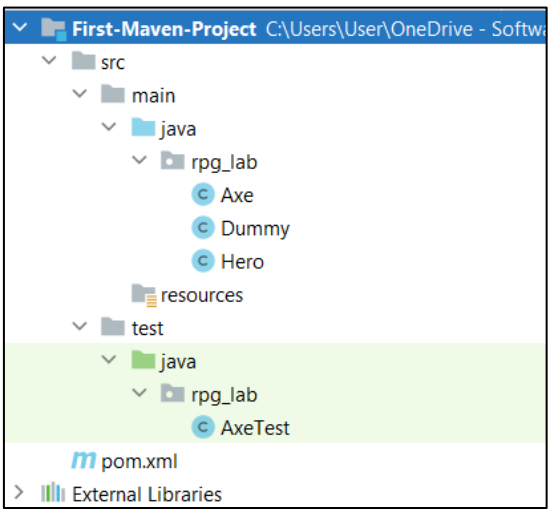
- Test if the weapon loses durability after each attack;
- Test attacking with a broken weapon.

Note

It is a good practice to name the folder in the test package the same as in the java package (rpg_lab).

Solution

Create the new package **`rpg_lab`** and inside create the class **`AxeTest:`**



Inside the class create your first test:

```
public class AxeTest {

    @Test
    public void weaponAttacksLosesDurability() {
        // Arrange

        //Act

        //Assert
    }
}
```

Arrange preconditions:

```
// Arrange
Axe axe = new Axe(10, 10);
Dummy dummy = new Dummy(10, 10);
```

Execute tested behaviour:

```
// Act
axe.attack(dummy);
```

Assert postconditions:

```
// Assert
Assert.assertEquals(9, axe.getDurabilityPoints());
```

Create your second test method:

```
@Test(expected = IllegalStateException.class) // Assert
public void brokenWeaponCantAttack() {
    // Arrange

    // Act
}
```

Arrange preconditions and test behaviour:

```
//ARRANGE
Axe axe = new Axe ( attack: 10,  durability: 1);
Dummy dummy = new Dummy( health: 20,  experience: 10);

//Act
axe.attack(dummy);
axe.attack(dummy);
```

3. Test Dummy

Create a class **DummyTest.**

Create the following tests:

- Dummy loses health if attacked.
- Dead Dummy throws an exception if attacked.
- Dead Dummy can give XP.
- Alive Dummy can't give XP.

Hints

Follow the logic of the previous problem.

4. Refactor Tests

Refactor the tests for **Axe** and **Dummy** classes.

Make sure that:

- **Names** of test methods are **descriptive.**
- You use **appropriate assertions** (assert equals vs assert true).
- You use **assertion messages.**
- There are **no magic numbers.**
- There is **no code duplication** (Don't Repeat Yourself).

Hints

Extract constants and private fields for **Axe** class:

```
private static final int AXE_ATTACK = 10;
private static final int AXE_DURABILITY = 1;
private static final int DUMMY_HEALTH = 20;
private static final int DUMMY_XP = 10;
private static final int EXPECTED_DURABILITY = AXE_DURABILITY - 1;

private Axe axe;
private Dummy dummy;
```

Create a method that executes **before each test:**

```
@Before
public void initializeTestObjects() {
    this.axe = new Axe(AXE_ATTACK, AXE_DURABILITY);
    this.dummy = new Dummy(DUMMY_HEALTH, DUMMY_XP);
}
```

Make use of constants and private fields, as well as add assertion messages:

```
@Test
public void weaponAttacksLosesDurability() {
    // Act
    this.axe.attack(this.dummy);

    // Assert
    Assert.assertEquals("Wrong Durability, ",
            EXPECTED_DURABILITY,
            this.axe.getDurabilityPoints());
}
```

Follow the same logic for other test methods and **`TestDummy`** class.

Part II: Dependencies

5. Fake Axe and Dummy

Test if the hero gains XP when a target dies.

To do this, you need to:

- Make **Hero** class **testable** (use **Dependency Injection**).
- Introduce **Interfaces** for Axe and Dummy:
    - Interface Weapon
    - Interface Target

Create a fake Weapon and fake Dummy for the test.

Hints

Create **Weapon** interface:

```
public interface Weapon {

    void attack(Target target);

    int getAttackPoints();

    int getDurabilityPoints();
}
```

Create **Target** interface:

```
public interface Target {

    void takeAttack(int attackPoints);

    int getHealth();

    int giveExperience();

    boolean isDead();
}
```

Implement interfaces :

```
public class Axe implements Weapon {
```

Modify implementation methods to **make use of interfaces:**

```
public void attack(Target target) {
    if (this.durabilityPoints <= 0) {
        throw new IllegalStateException("Axe is broken.");
    }

    target.takeAttack(this.attackPoints);
    this.durabilityPoints -= 1;
}
```

Modify both **Axe** and **Dummy** classes.

Use **Dependency Injection** for Hero class:

```
public Hero(String name, Weapon weapon) {
    this.name = name;
    this.experience = 0;
    this.weapon = weapon;
}
```

Create **`HeroTests`** class and test gaining XP functionality by faking Weapon and Target classes:

```
@Test
public void attackGainsExperienceIfTargetIsDead() {
    Target fakeTarget = new Target() {
        public void takeAttack(int attackPoints) { }
        public int getHealth() { return 0; }
        public int giveExperience() { return TARGET_XP; }
        public boolean isDead() { return true; }
    };

    Weapon fakeWeapon = new Weapon() {
        public void attack(Target target) {}
        public int getAttackPoints() { return 10; }
        public int getDurabilityPoints() { return 0; }
    };

    Hero hero = new Hero(HERO_NAME, fakeWeapon);
    hero.attack(fakeTarget);
    Assert.assertEquals("Wrong experience", TARGET_XP, hero.getExperience());
}
```

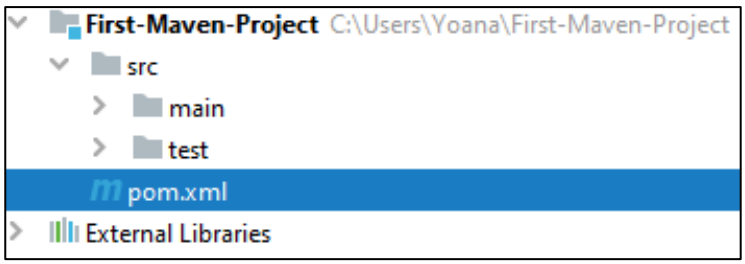
6. Mocking

Include **`Mockito`** in the project dependencies, then:

1. Mock fakes from the previous problem.
2. Implement **Hero Inventory**, holding unequipped weapons:
    a. method - **`Iterable<Weapon> getInventory()`**
3. Implement Target giving random weapons upon death:
    a. field - **`private List<Weapon> possibleLoot`**
4. Test Hero killing a target getting loot in his inventory.

Hints

Locate **`pom.xml`**.



Add **Mockito** dependency.

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

    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-android</artifactId>
        <version>2.7.19</version>
        <type>pom</type>
    </dependency>
</dependencies>
```

Go to **HeroTests** and refactor the code, making use of **Mockito:**

```
@Test
public void attackGainsExperienceIfTargetIsDead() {
    Weapon weaponMock = Mockito.mock(Weapon.class);
    Target targetMock = Mockito.mock(Target.class);
    Mockito.when(targetMock.isDead()).thenReturn(true);
    Mockito.when(targetMock.giveExperience()).thenReturn(TARGET_XP);

    Hero hero = new Hero(HERO_NAME, weaponMock);

    hero.attack(targetMock);

    Assert.assertEquals("Wrong experience", TARGET_XP, hero.getExperience());
}
```

*Implement hero inventory and **Target** dropping loot functionalities.

*Test **Hero** getting loot upon killing a **Target.**