Classes and Objects

How to create a program

Classes

Collection of

- ***Data members*** – to store class or object state
- ***Methods*** – to change and read object state or perform state independent operations

Organization rules

- **Encapsulation** – logically related data and operations
- **Abstraction** – hide implementation, interface of a certain abstraction level
- **Single responsibility** – class is responsible for one task (abstraction dependent)

Class hierarchy

- Reuse of code
- **Inheritance**: Extension of successor class
- **Polymorphism**: change inherited behaviours
- Reflects abstraction hierarchy (general – special connection)

Abstract classes

- Abstract members do not represent business objects, because they are „too generic”. **Can not be instantiated**.
- Body of abstract methods can not be defined.

- To define classes with abstract method
- To specify classes as abstract

Classes for IO

- Separate class(es) according to Single Responsibility
- Create separate classes for IO management
  - Input – reading from console or files
  - Output – writing to console or files
- An I/O class can combine both

Objects

Represent items of the real world (business or system)

Objects are **instance**s of classes

All objects of the same type have:

- Custom values of common properties – instance data, instance state
- Shared methods, working on instance data
- Controlled state transitions – consistent internal state descriptors

Object relations

- Data members can store references to other objects
- These references
  - Are **binary** relations (between 2 objects)
  - **Declare** class **associations**
  - **Implement** object **links**

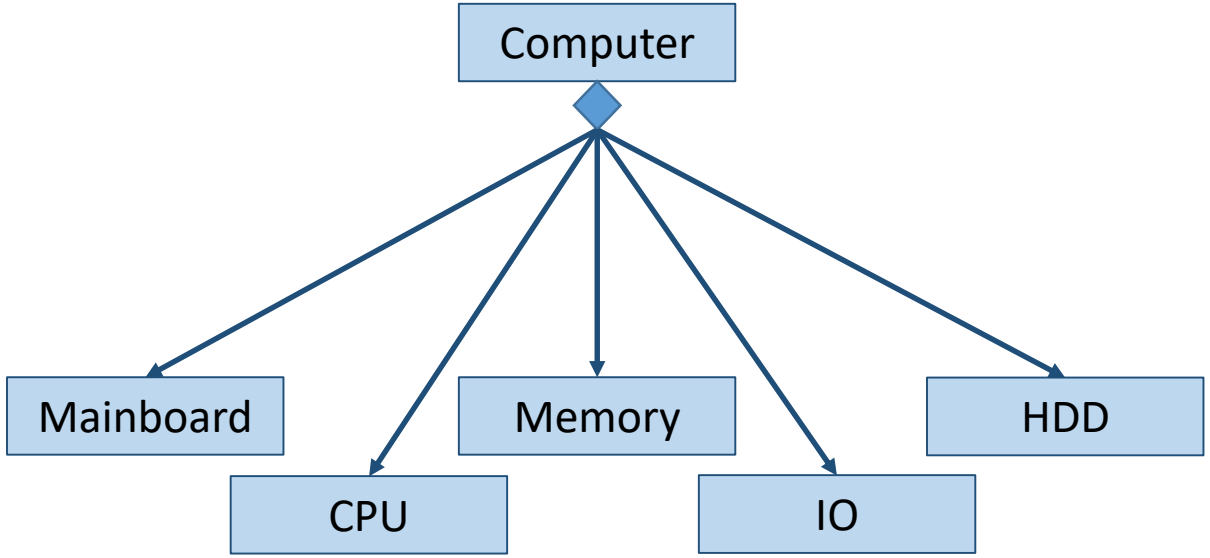
Component association

- Using association, logically an object can contain other objects
- Other words: object can be composition of other (component) objects
- Unlike inheritance, ask with question: HAS-A?

- Composition/component is a special case of association
- There is no reference to the component out of the container

Aggregation

- Multiple component association
- A logical part-whole relationship
- Where parts are components of the whole



Transitivity

Composition and Aggregation are

- Transitive – when A is a part of B and B is a part of C then A is a part of C
- Antisymmetric – when A is a part of B, then B is not part of A

Composition is a special case of Aggregation, where the number of components is restricted to 1

Association multiplicity

- Physical link is binary
- Logical association (a logical relation) has multiplicity on both end

- Multiplicity: how many links could be initiated from or end in an object

Association example

Association multiplicity

- Exactly one: 1
- Zero or one: 0..1
- Zero or more: 0..*
- One or more: 1..*
- Interval: 3..8
- Discrete values: 2, 4, 6, 8

Association implementations – links

- Exactly one: 1
- Zero or one: 0..1
- Zero or more: 0..*
- One or more: 1..*
- Interval: 3..8
- Discrete values: 2, 4, 6, 8

Link **data member**

Link **data member** with NULL value

ArrayList, can be empty

ArrayList, can NOT be empty

ArrayList with controlled number of items

ArrayList with controlled number of items

Link restrictions

- Exactly one: 1
- Zero or one: 0..1
- Zero or more: 0..*
- One or more: 1..*
- Interval: 3..8
- Discrete values: 2, 4, 6, 8

Link **data member**	Ctor, get, set
Link **data member** with NULL value	No restriction
ArrayList, can be empty	No restriction
ArrayList, can NOT be empty	Ctor, add, remove
ArrayList with controlled number of items	Ctor, add, remove
ArrayList with controlled number of items	Ctor, add, remove

Interfaces

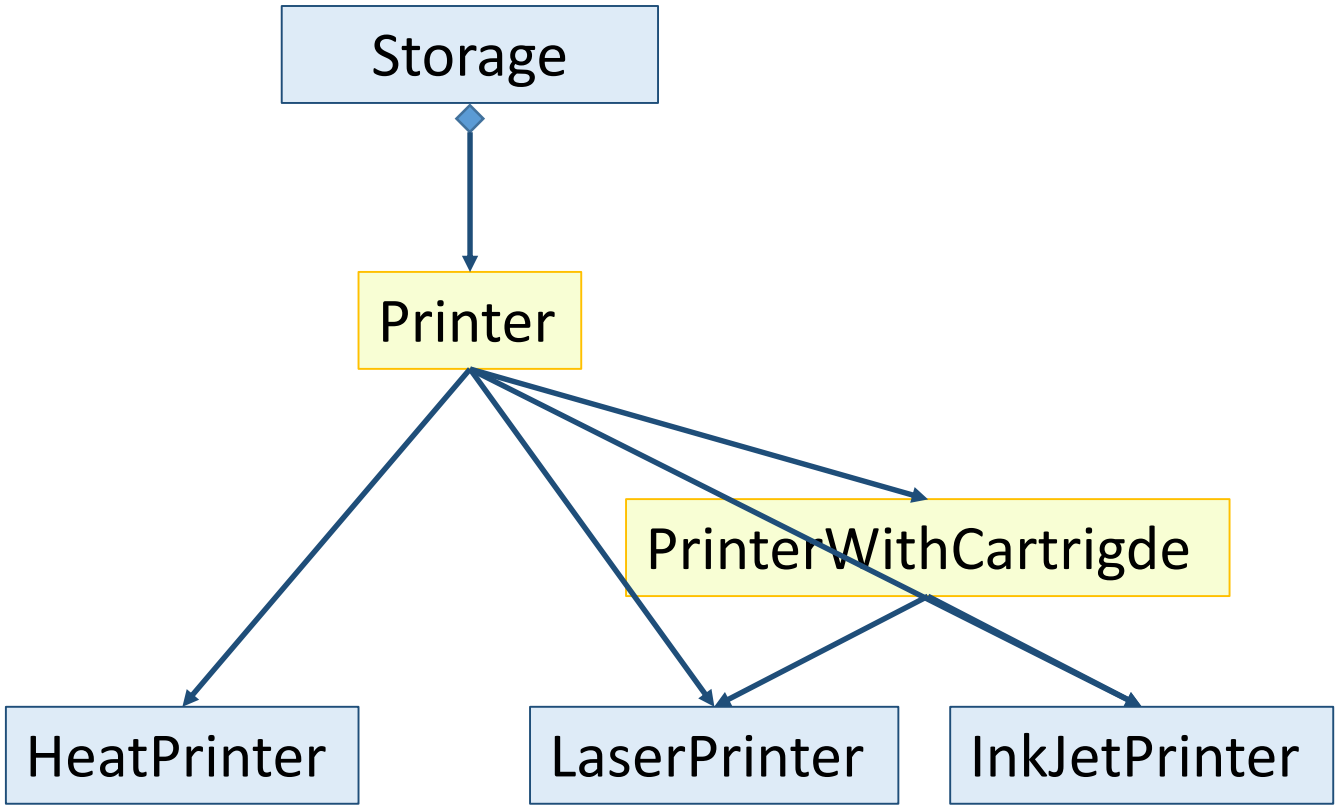
- Associations of classes couple (bind) them to each other

- Create abstraction independent connection in class hierarchy
- Make classes independent (decouple)

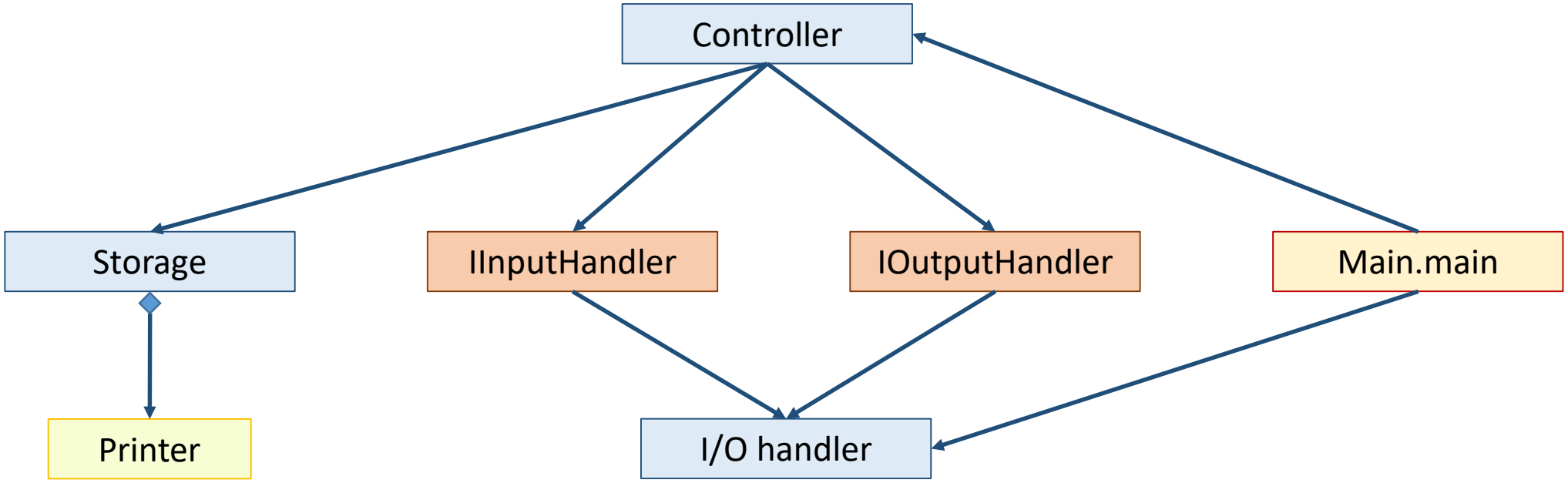
Final step – make it working

- Create Controllers to associate models and IO interfaces
- In execution entry point (`Main.main`) create link to controllers
- On execution
  - Initialize controllers (and models)
  - Handle call to controllers

Create model



Storage, Controller, Main



Compilation support

Maven

Apache Maven (https://maven.apache.org/)

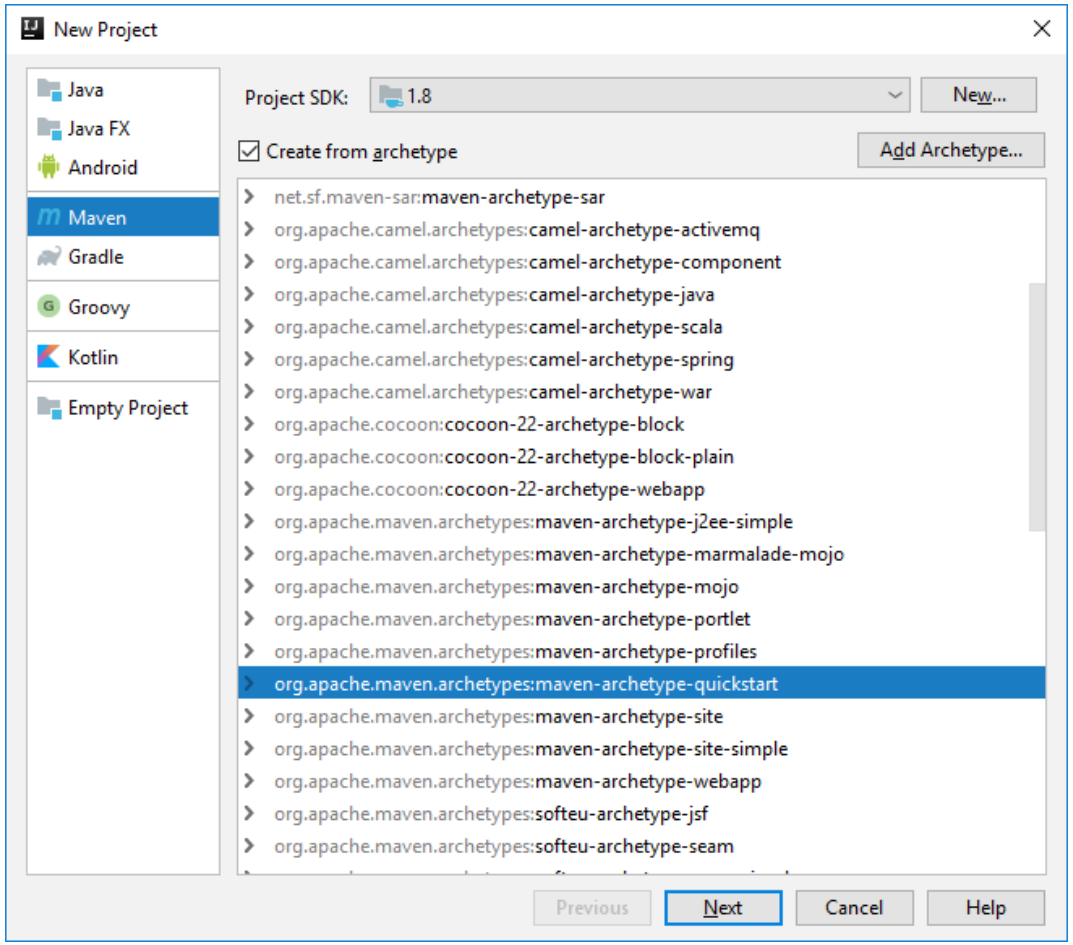
Apache Maven is a software project management tool.

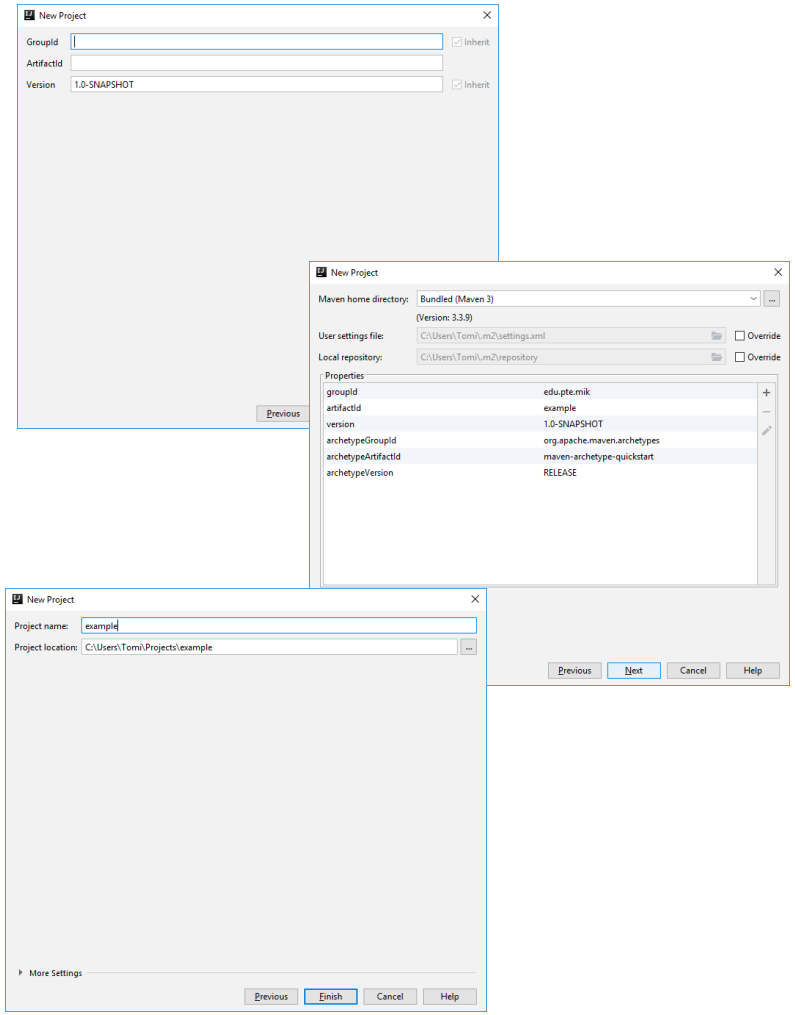
Maven can manage a project's build, reporting and documentation from a central project object model (POM).

Maven objectives (https://maven.apache.org/what-is-maven.html)

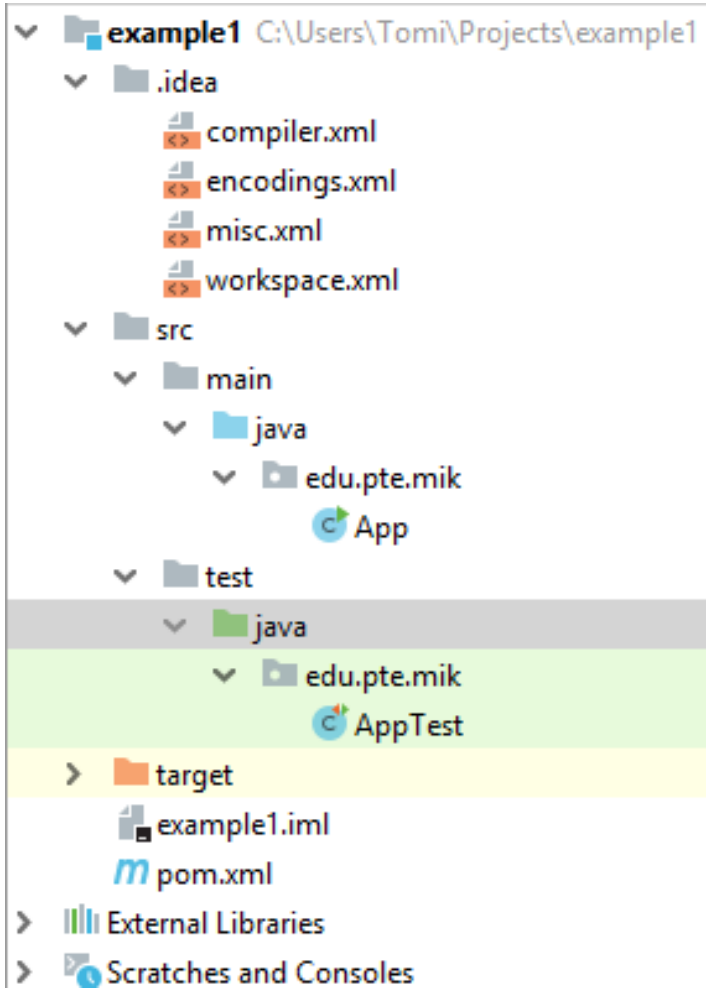
- Making the build process easy – hide the details
- Providing a uniform build system – generic knowledge
- Providing quality project information – dependencies, changelog, unit tests
- Providing guidelines for best practices development – separate test sources and environment, applying naming conventions
- Allowing transparent migration to new features – handles plugin and dependency updates

Create Maven project





Project structure



Known components:

- **main** – project sources
- **iml** – idea module descriptor

New components

- **test** – unit test sources
- **pom** – Project Object Model

Project Object Model and dependencies

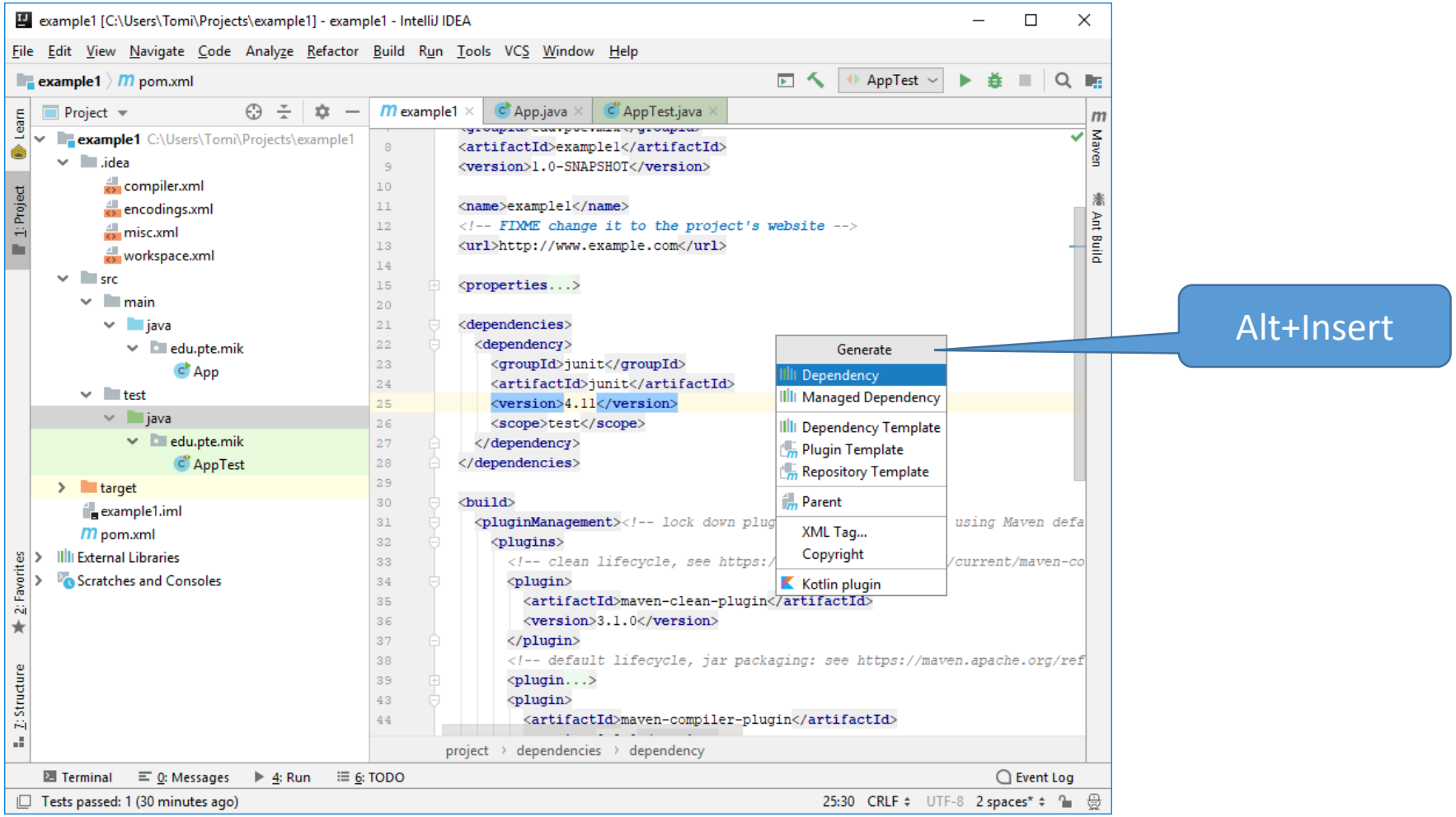
```xml
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>edu.pte.mik</groupId>
  <artifactId>example1</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>example1</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties...>

  <dependencies...>

  <build...>
</project>
```

```xml
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

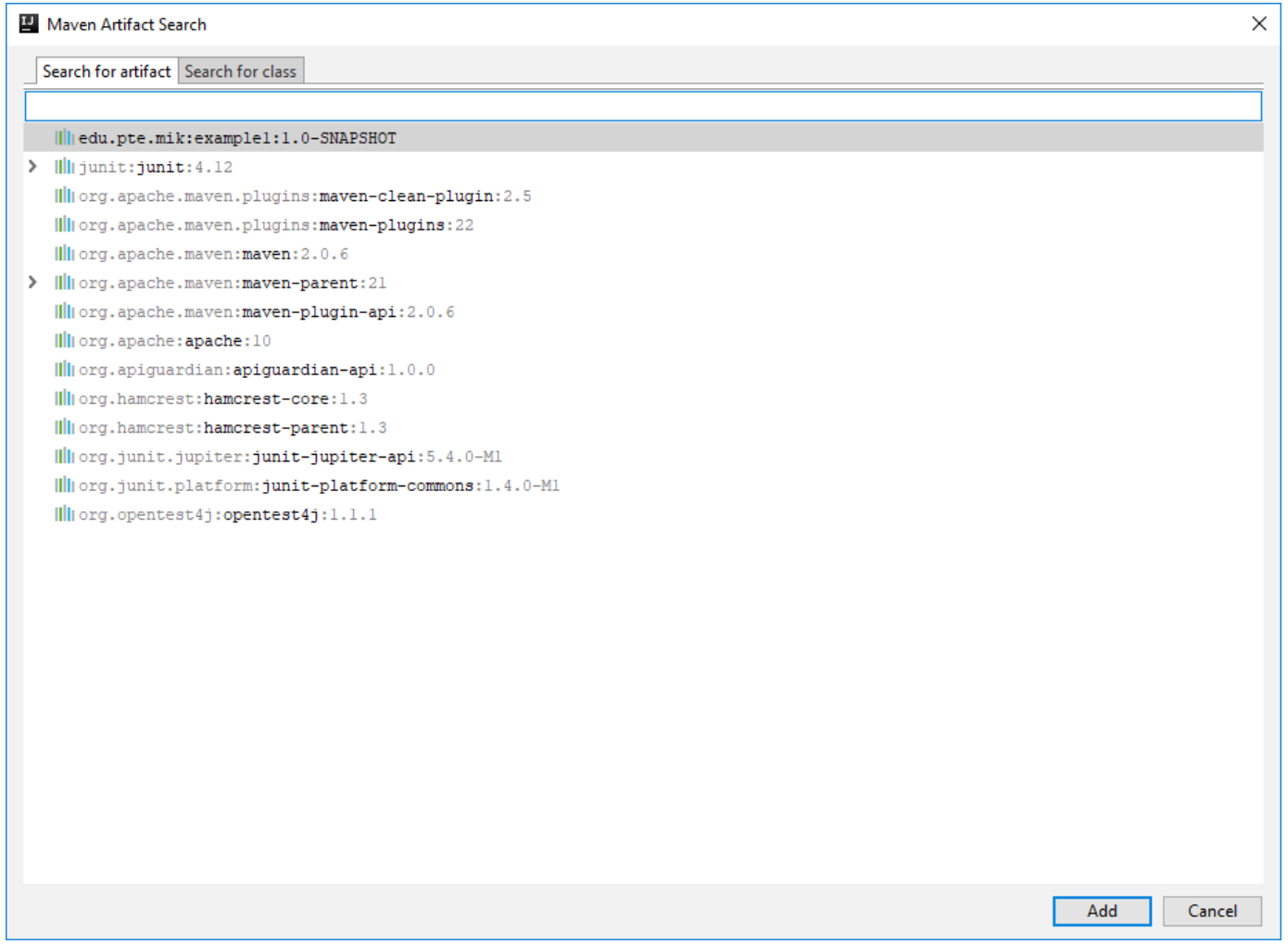
Maven Build

```
<build>
  <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
    <plugins>
      <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle -->
      <plugin>
        <artifactId>maven-clean-plugin</artifactId>
        <version>3.1.0</version>
      </plugin>
      <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin_bindings_for_jar_packaging -->
      <plugin...>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin...>
      <plugin...>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>2.5.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
      <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle -->
      <plugin...>
      <plugin>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>3.0.0</version>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
```

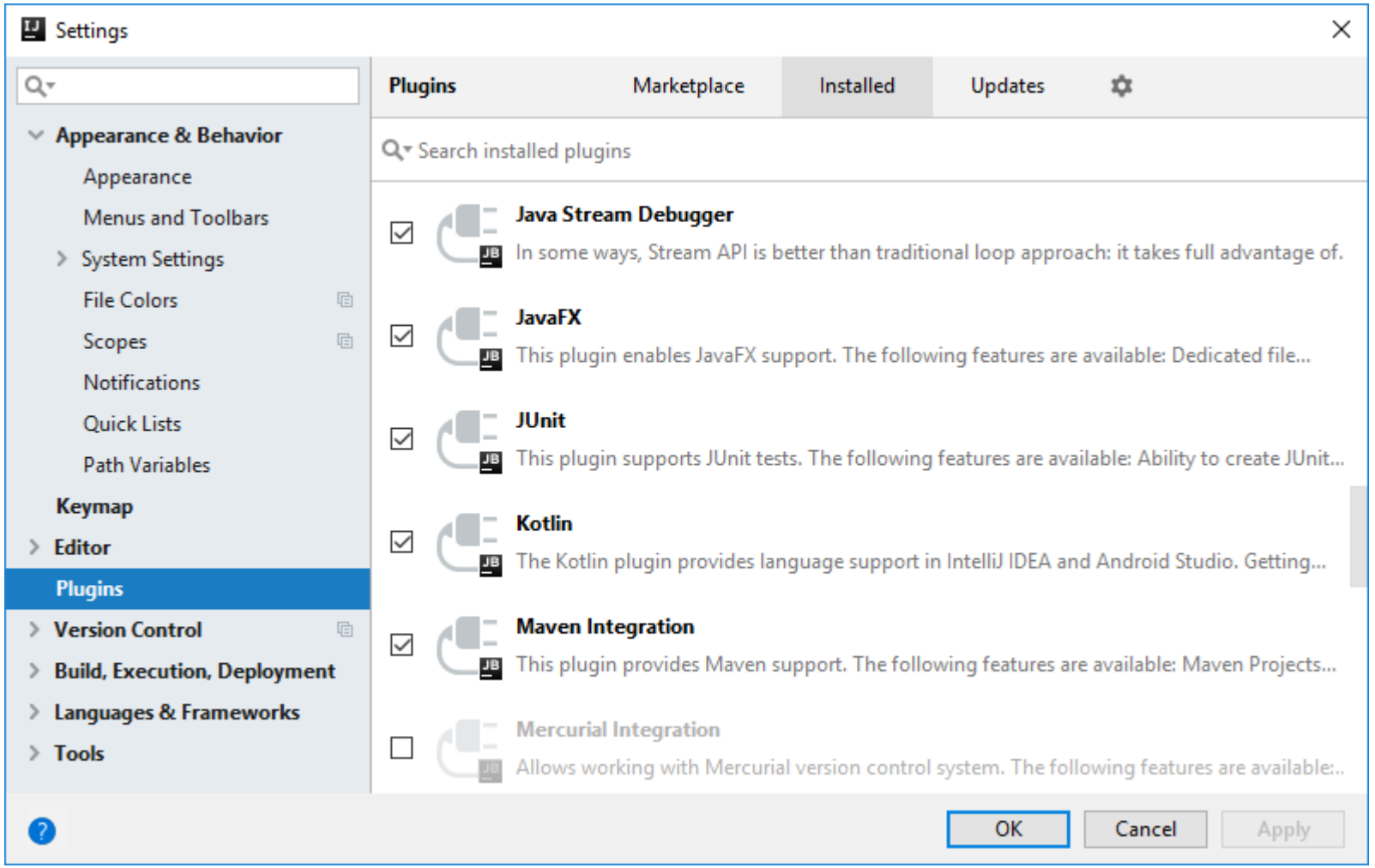
Manage dependencies



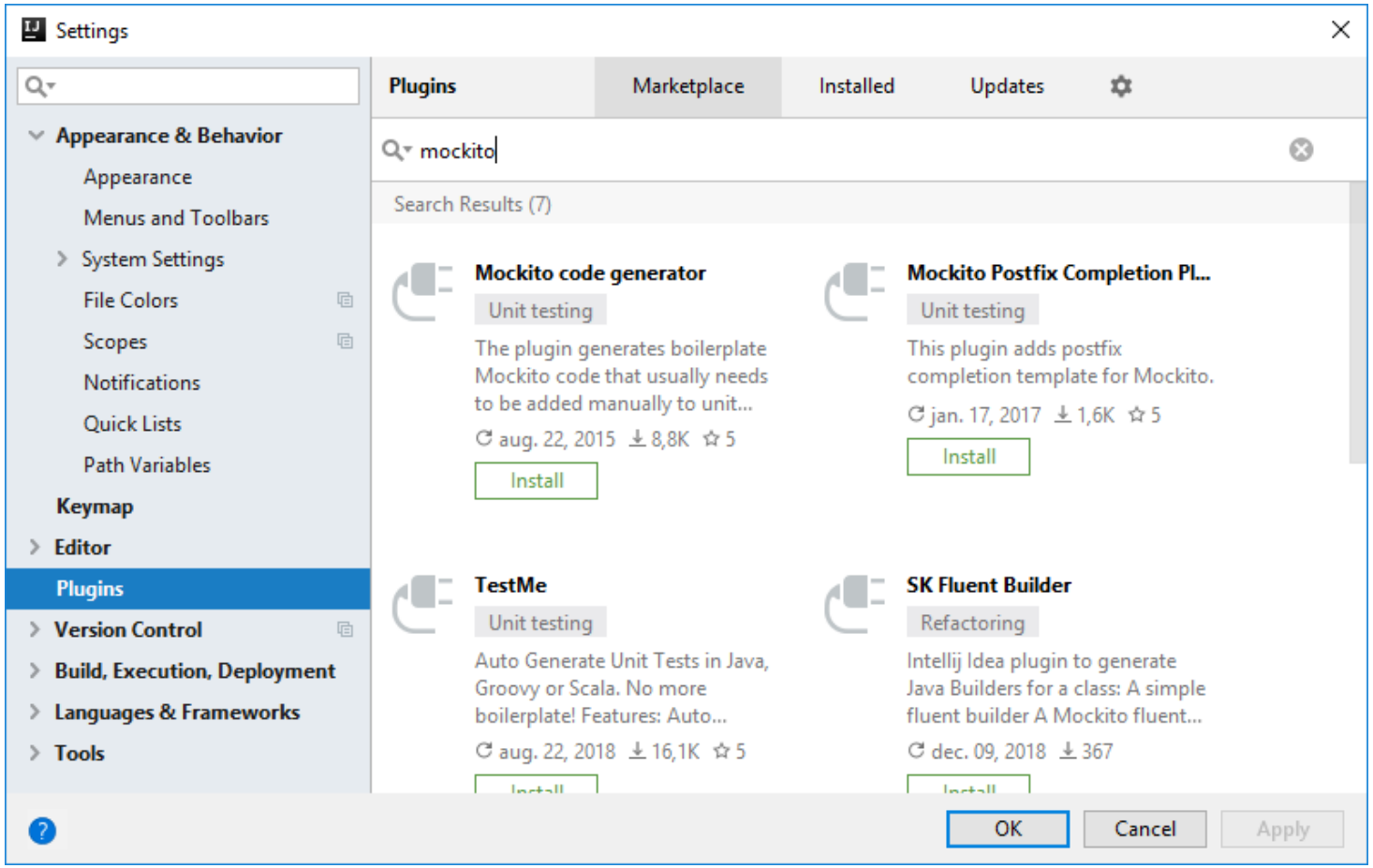
Select dependencies



Installed plugins



Add plugins


Settings
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace
Installed
Updates
mockito
Search Results (7)
Mockito code generator
Unit testing
The plugin generates boilerplate Mockito code that usually needs to be added manually to unit...
aug. 22, 2015 8,8K 5
Install
Mockito Postfix Completion Pl...
Unit testing
This plugin adds postfix completion template for Mockito.
jan. 17, 2017 1,6K 5
Install
TestMe
Unit testing
Auto Generate Unit Tests in Java, Groovy or Scala. No more boilerplate! Features: Auto...
aug. 22, 2018 16,1K 5
SK Fluent Builder
Refactoring
Intellij Idea plugin to generate Java Builders for a class: A simple fluent builder A Mockito fluent...
dec. 09, 2018 367
OK
Cancel
Apply

Automated tests

Junit, Mockito

Types of tests

- **Unit** test – low level tests of source codes
- **Integration** test – verify that modules and services work well together
- **Functional** test – focus on business requirements
- **End-to-end** test – user flow tests
- **Acceptance** test – formal tests of business requirements of the whole application
- **Smoke** test – reduced set of tests of basic functionalities, runs fast
- **Regression** test – set of tests to verify that system did not fall back to a previous state

Program unit

- The smallest testable part of the code
- In OOP
  - A class
  - An interface

Unit tests

Short code fragments (usually written by developers) to test

- Individual units of source code
- Set of modules
- Usage procedures
- Operating procedures

together with control data are tested

Code coverage

Unit test are related to source code units.

One measure of tests is the coverage of source code by tests

This approach is false, because it is based on implementation, does not identify design or implementation errors

Testing important parts

Better approach is to identfy important or dangerous cases.

Some of them can be identified by the developer based on representation, storage or computation knowledge or experience

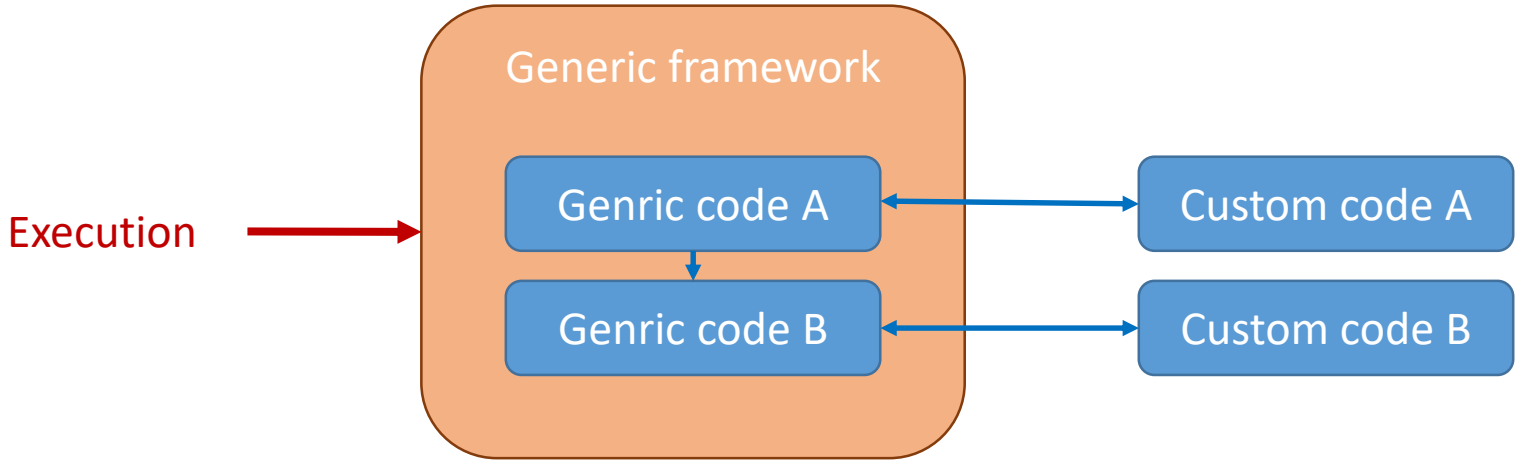
But some cases can be identified by system designers or domain experts, because they understand the full process and input/output data

Data (User story) based approach

Frameworks

Inversion of control

- specialized, custom portions of a computer program receive
- the flow of control
- from a generic framework.



JUint

Up-to-date foundation of developer-side testing on the JVM.
An instance of xUnit architecture for java language.
A simple unit testing framework to write repeatable, automated tests.

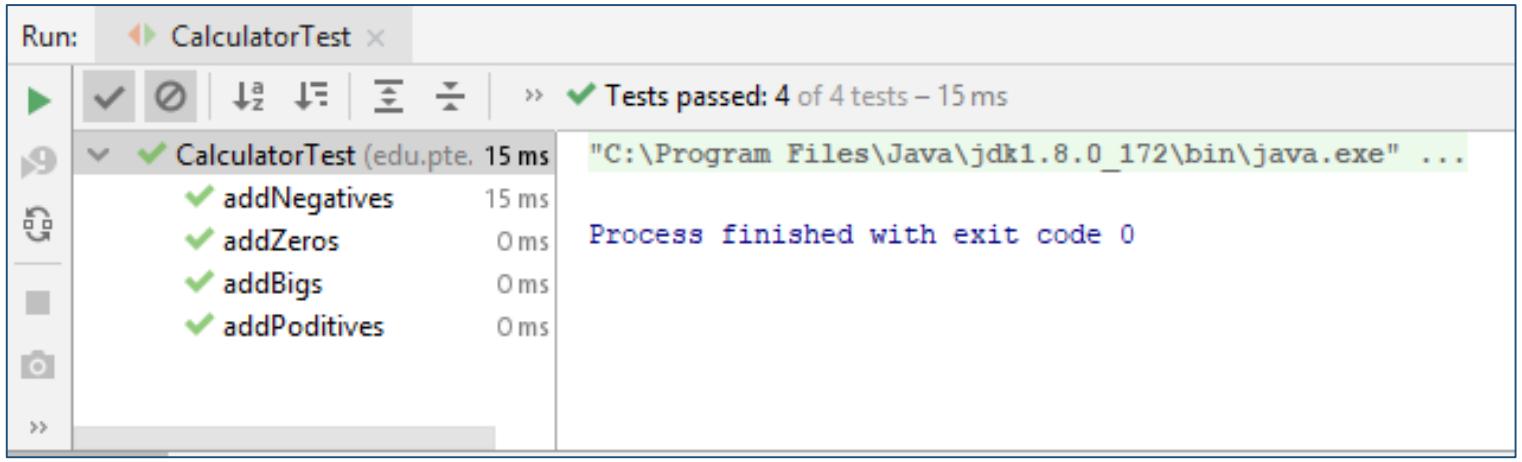
Can be installed as an IDEA plugin
IDEA is able to generate test classes
Tests can be injected into the framework
Can be run independently from the application
Unit tests can be run parallel

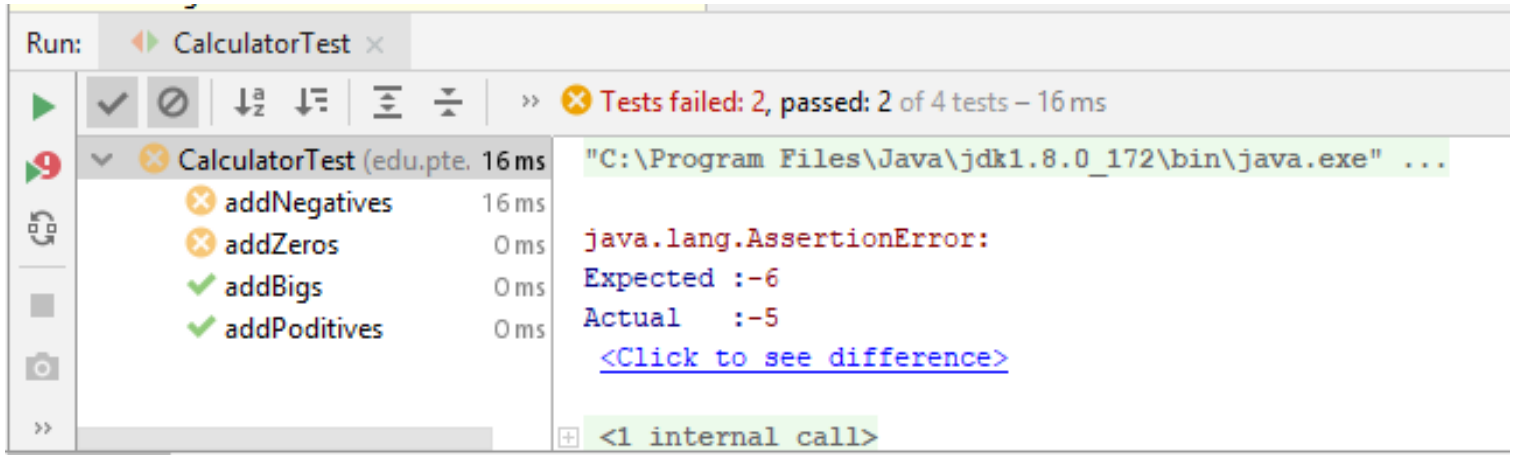
Unit tests

```java
public class Calculator {

    public static int add(int a, int b) {
        return a+b;
    }

}
```

```java
public class CalculatorTest {
    @Test
    public void addPoditives()
    {
        assertEquals( expected: 5, Calculator.add( a: 3,  b: 2));
    }

    @Test
    public void addNegatives()
    {
        assertEquals( expected: -5, Calculator.add( a: -3,  b: -2));
    }

    @Test
    public void addZeros()
    {
        assertEquals( expected: 0, Calculator.add( a: 0,  b: 0));
    }

    @Test
    public void addBigs()
    {
        assertEquals( expected: 2000000000, Calculator.add( a: 1000000000,  b: 1000000000));
    }
}
```

Unit test results





Mock objects

Mock objects are simulated objects that **mimic** the behavior of real objects in **controlled ways**.

This is used to test class associations and dependencies.

No need to test more units together.

Mockito (https://site.mockito.org/)

- Mockito is a mocking framework that lets you write beautiful tests with a clean & simple API
- Mockito mocks are often *ready* without expensive setup upfront. They aim to be transparent and let the developer to focus on testing selected behavior rather than absorb attention.
- Mockito has very slim API, almost no time is needed to start mocking. There is only one kind of mock, there is only one way of creating mocks.

Mockito features

- Mocks concrete classes as well as interfaces
- Little annotation syntax sugar - @Mock
- Verification errors are clean – clean stack trace
- Allows flexible verification in order (verify what you want, not every single interaction)
- Supports exact-number-of-times and at-least-once verification
- Flexible verification using argument matchers (reflection-based equality matching)
- Allows creating custom argument matchers or using existing hamcrest matchers

Hamcrest – matcher framework

- Hamcrest is a framework for writing matcher objects allowing ‘match’ rules to be defined declaratively.
- There are a number of situations where matchers are invaluble, such as UI validation, or data filtering, but it is in the area of writing flexible tests that matchers are most commonly used.