University of Antwerp
Faculty of Applied Engineering

5-Software Design

Lab Session 2

28/10/2025

Jens Duym

Course Outline

Outline labs

- **Part A: UML diagrams**
  Sessions 1 – 2
- **Part B: Design Patterns**
  Session 3 – 5
- **Part C: Projects in groups of 2**
  Session 6 – 9

- **Evaluation:**
  - Entire portfolio: zip containing code, UML diagrams, AI usage
    - Submit before 7th lab at defined date
    - Oral defence
  - Defence of projects

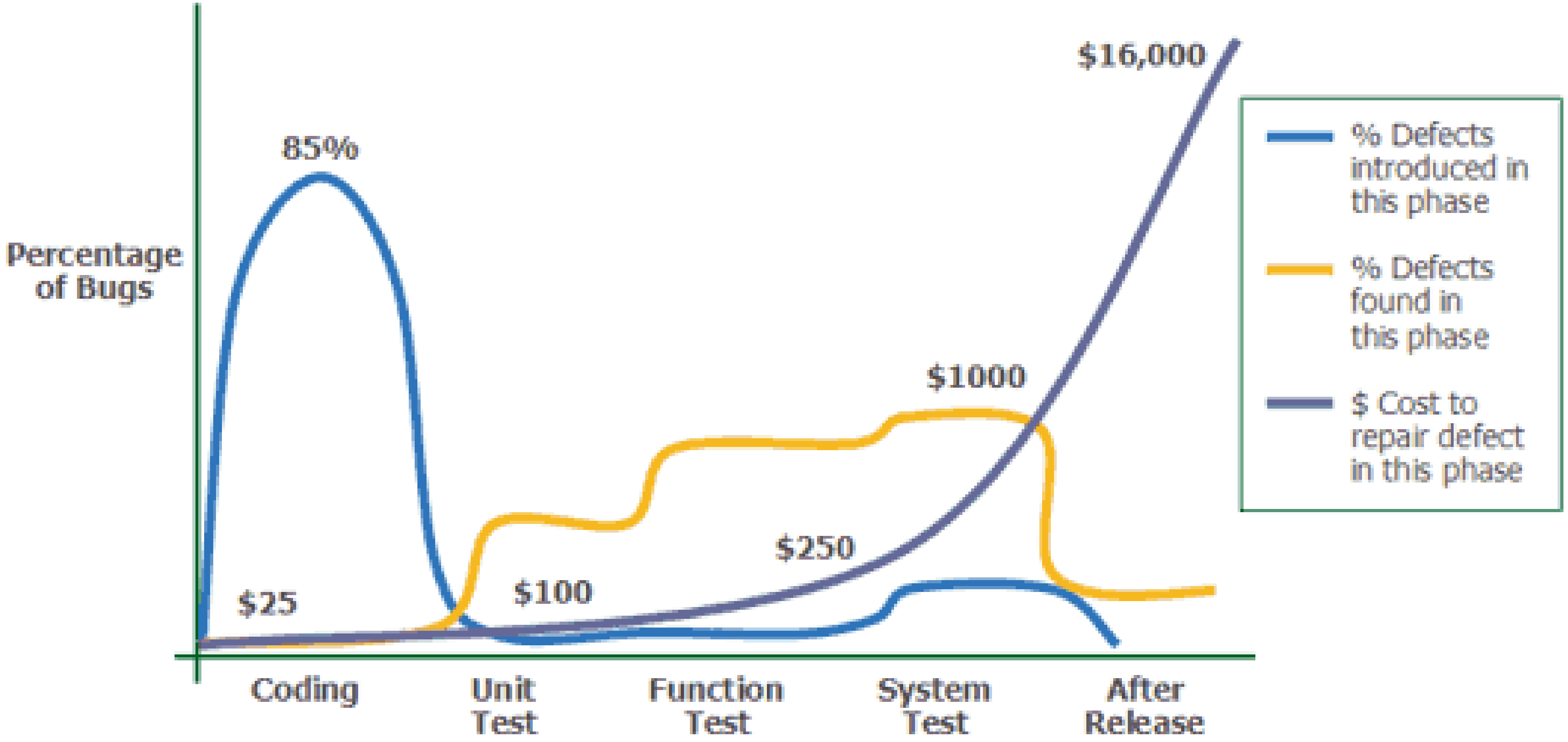
Testing

Introduction to Unit Testing & Integration Testing

Why testing?

- Testing is to show that a program does what it is **intended** to do and to **discover program defects** before it is put into use
- When you test software, you execute a program using **artificial data**
- You check the results of the test run for errors, anomalies or information about the program's non-functional attributes
- Testing and **revealing the presence** of errors **NOT their absence**
- Testing is part of a more general **verification** and **validation process**, which also includes static validation techniques

Why testing?



Source: *Applied Software Measurement*, Capers Jones, 1996

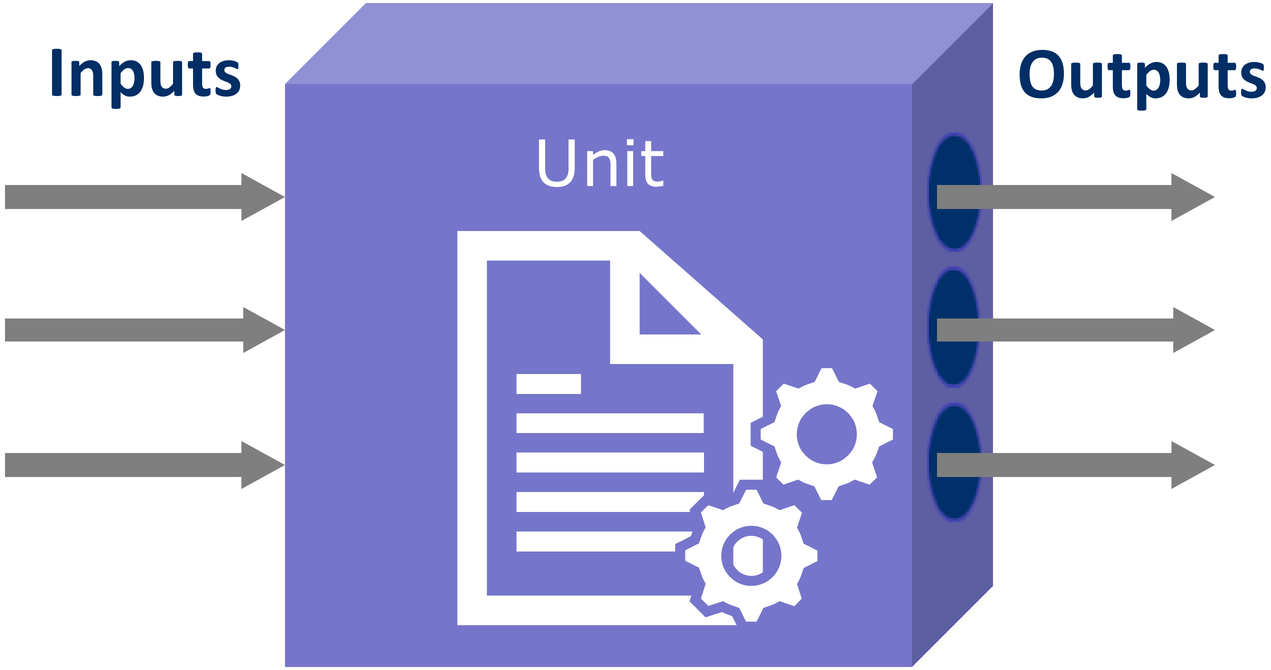
Development Testing

- Bug and defect testing during development
- Carried out by the team developing the system
- Types
    - Unit testing
        - Individual units or object classes are tested
        - Functionality of objects or methods
    - Component testing
        - Several individual units are integrated to create composite components
        - Focus on component interfaces
    - System / integration testing
        - Some/all components are integrated and the system is tested as a whole
        - Focus on testing component interactions

Development Testing

- Bug and defect testing during development
- Carried out by the team developing the system
- Types
  - Unit testing
    - Individual units or object classes are tested
    - Functionality of objects or methods
  - Component testing
    - Several individual units are integrated to create composite components
    - Focus on component interfaces
  - System / integration testing
    - Some/all components are integrated and the system is tested as a whole
    - Focus on testing component interactions

Unit testing

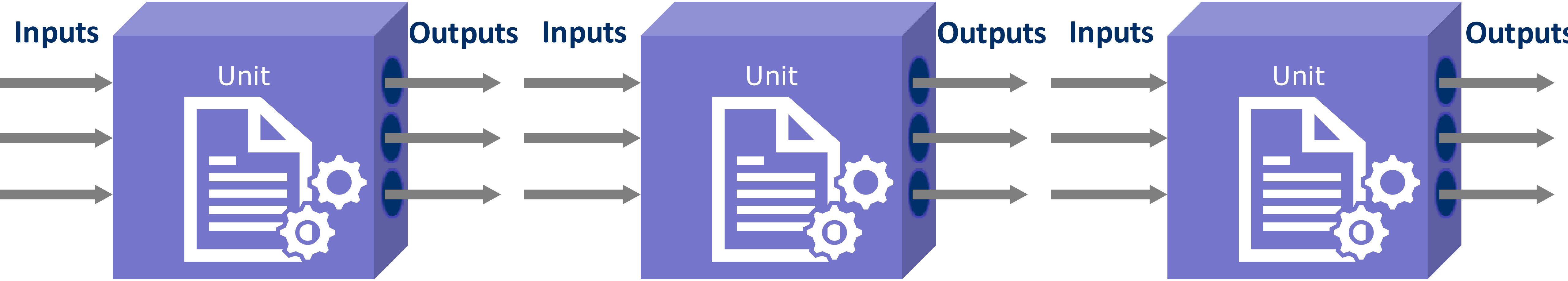


Inputs

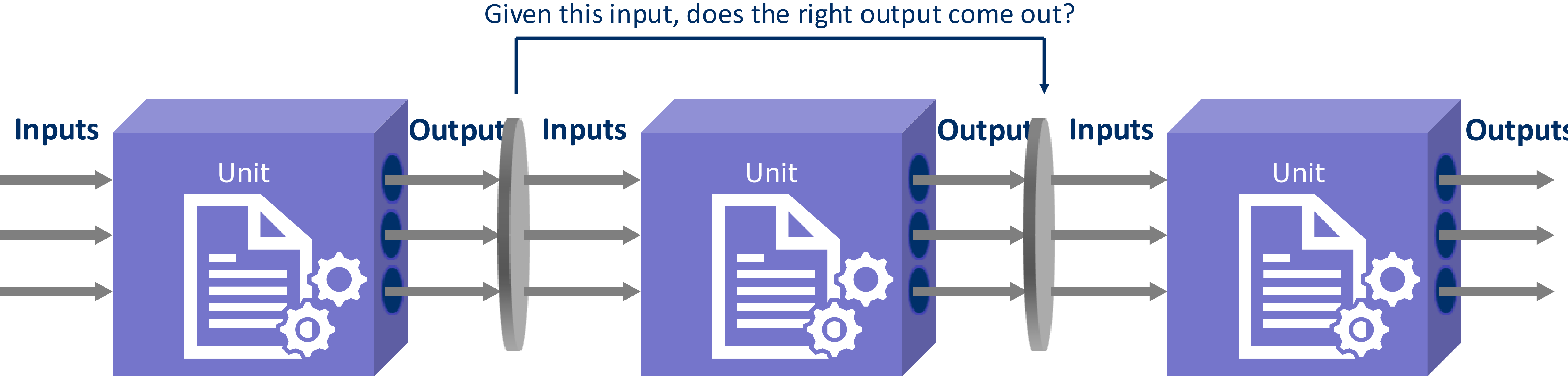
Unit

Outputs

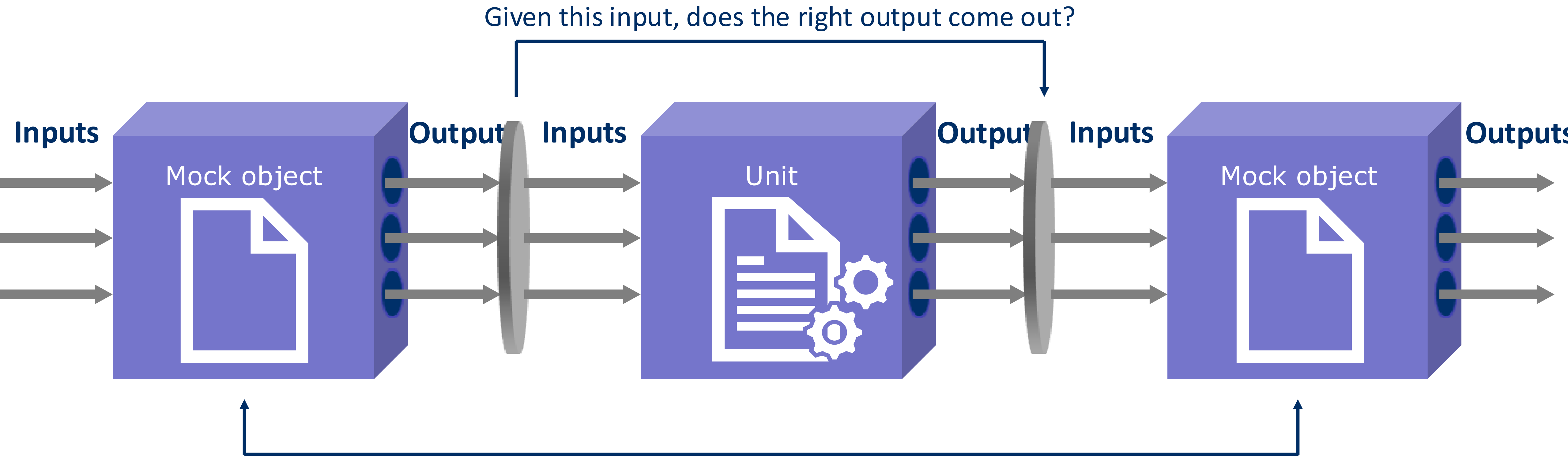
Unit testing



Unit testing



Unit testing

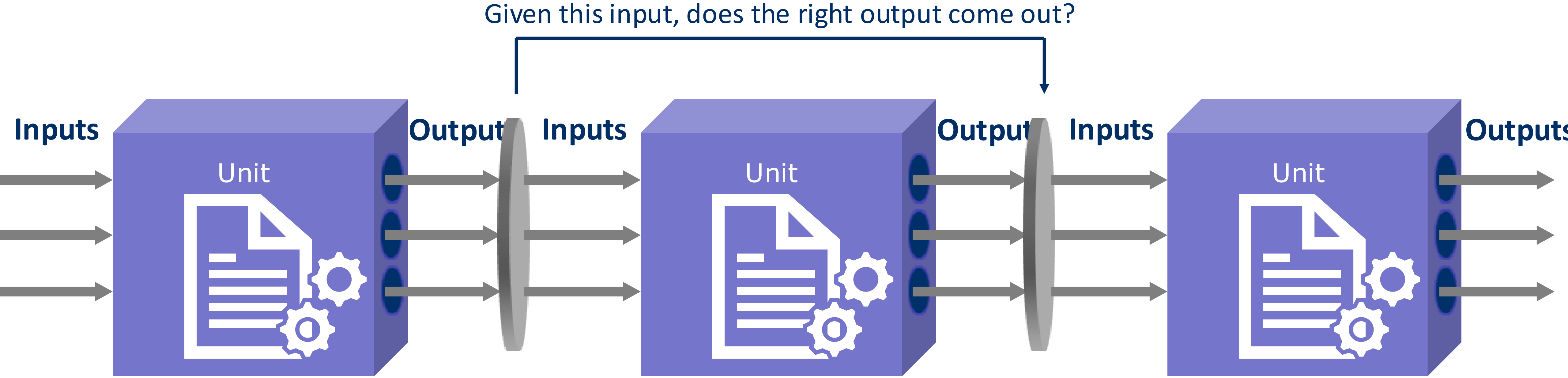


Isolation of behavior

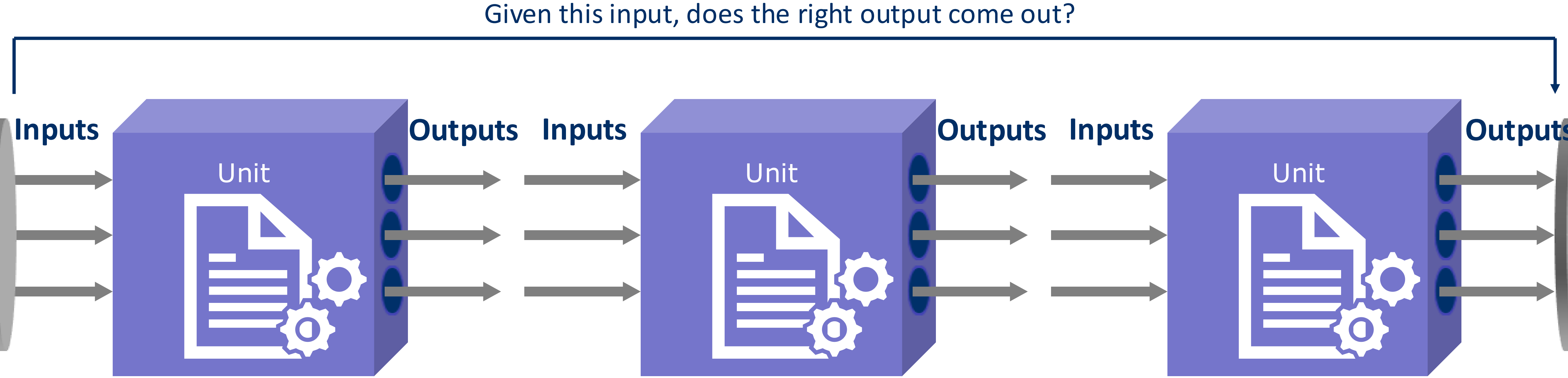
→ Only testing the middle box, so the others become mock objects

→ Same inputs & outputs, but no internal behavior

Unit testing

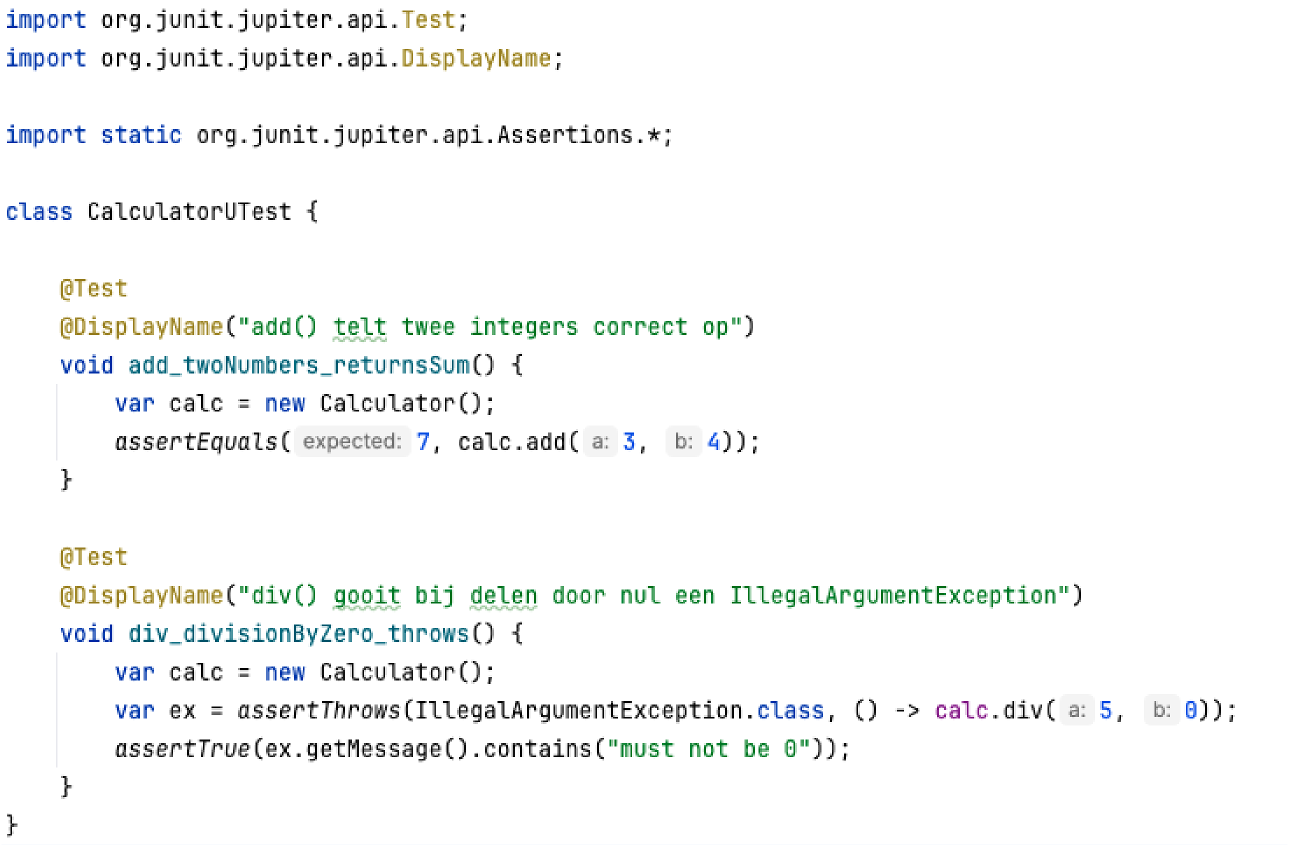


Integration testing

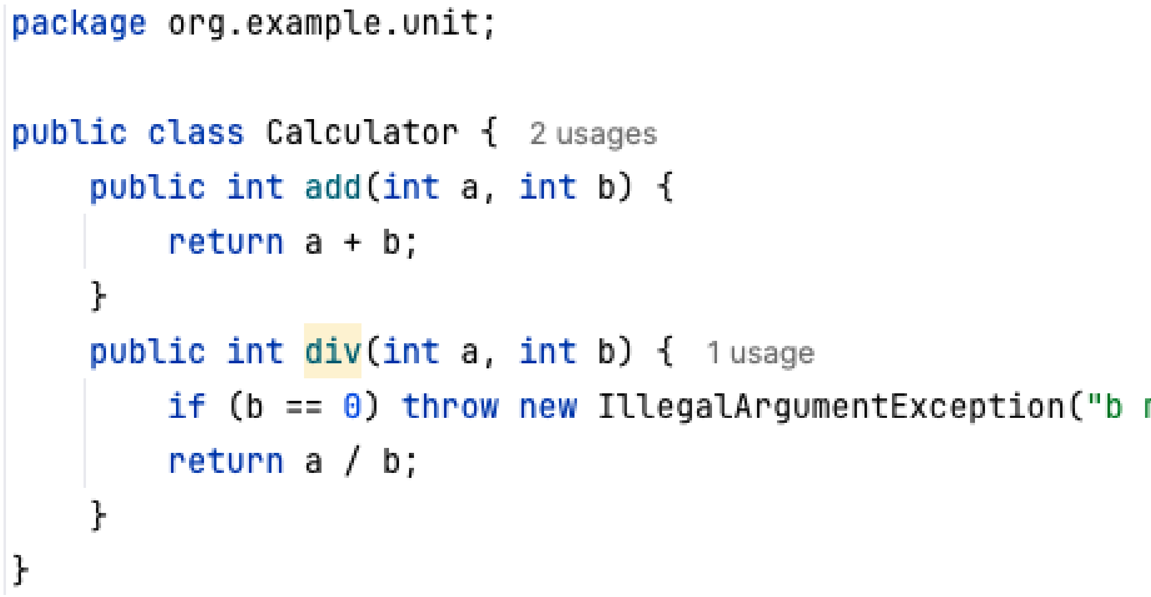


Unit Testing Example

- See TestingStudents.zip on BB



```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;

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

class CalculatorUTest {

    @Test
    @DisplayName("add() telt twee integers correct op")
    void add_twoNumbers_returnsSum() {
        var calc = new Calculator();
        assertEquals( expected: 7, calc.add( a: 3,  b: 4));
    }

    @Test
    @DisplayName("div() gooit bij delen door nul een IllegalArgumentException")
    void div_divisionByZero_throws() {
        var calc = new Calculator();
        var ex = assertThrows(IllegalArgumentException.class, () -> calc.div( a: 5,  b: 0));
        assertTrue(ex.getMessage().contains("must not be 0"));
    }
}
```



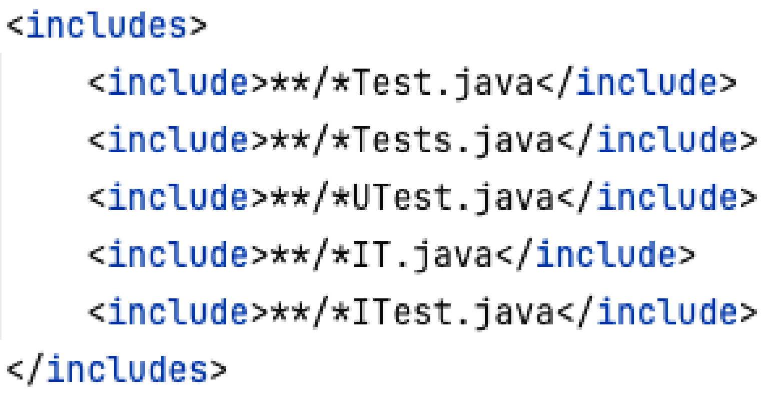
```
package org.example.unit;

public class Calculator {  2 usages
    public int add(int a, int b) {
        return a + b;
    }
    public int div(int a, int b) {  1 usage
        if (b == 0) throw new IllegalArgumentException("b must not be 0");
        return a / b;
    }
}
```

Integration/Component Testing Example

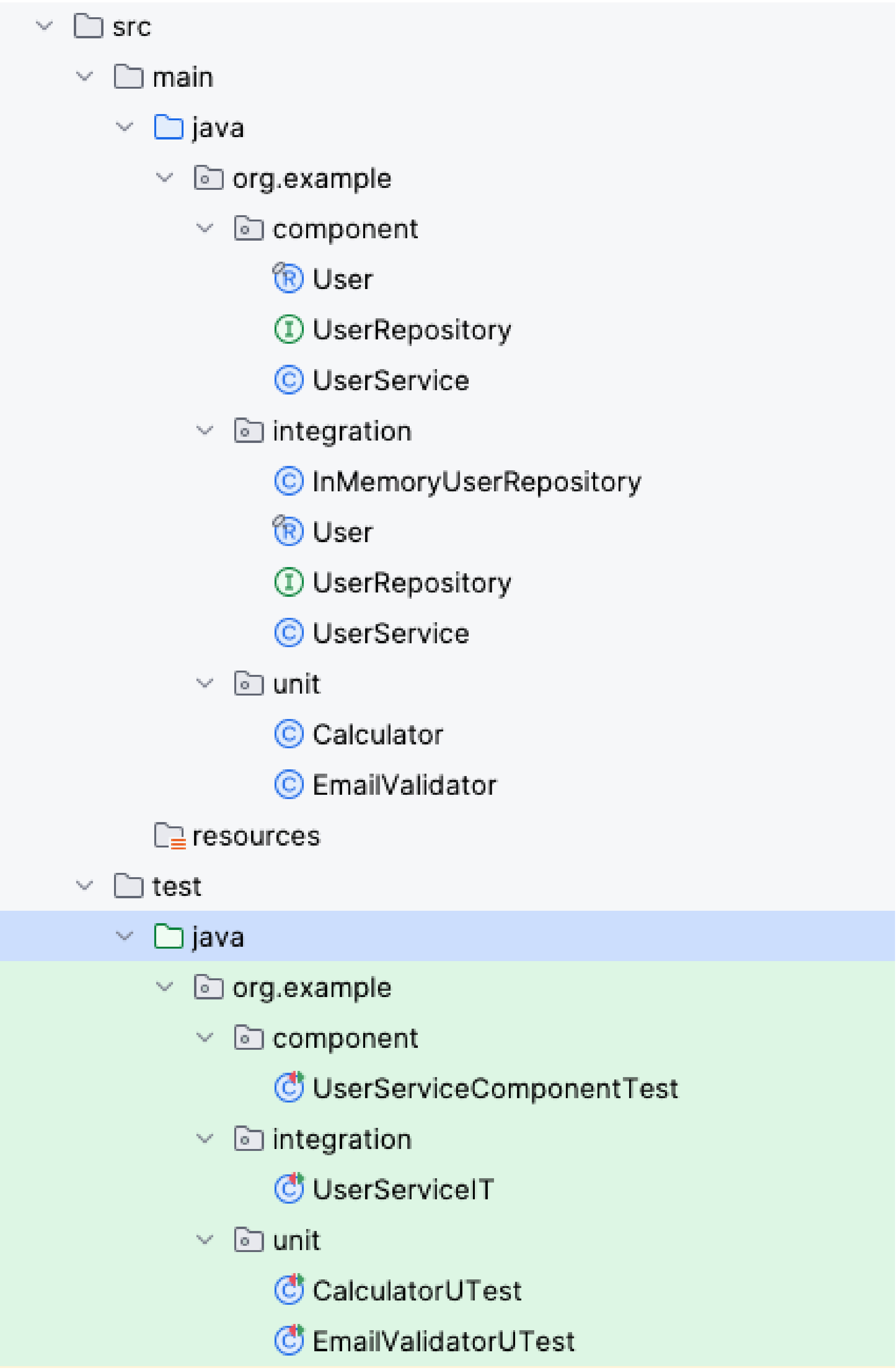
- See TestingStudents.zip on BB
- Documentation can be found online
  - https://docs.junit.org/5.12.2/user-guide/index.html
  - https://site.mockito.org/
  - https://maven.apache.org/components/surefire-archives/surefire-LATEST/maven-surefi

IntelliJ Example

- Create a ‘test’ folder
- All tests should end with:
  - Test
  - UTest
  - ITest



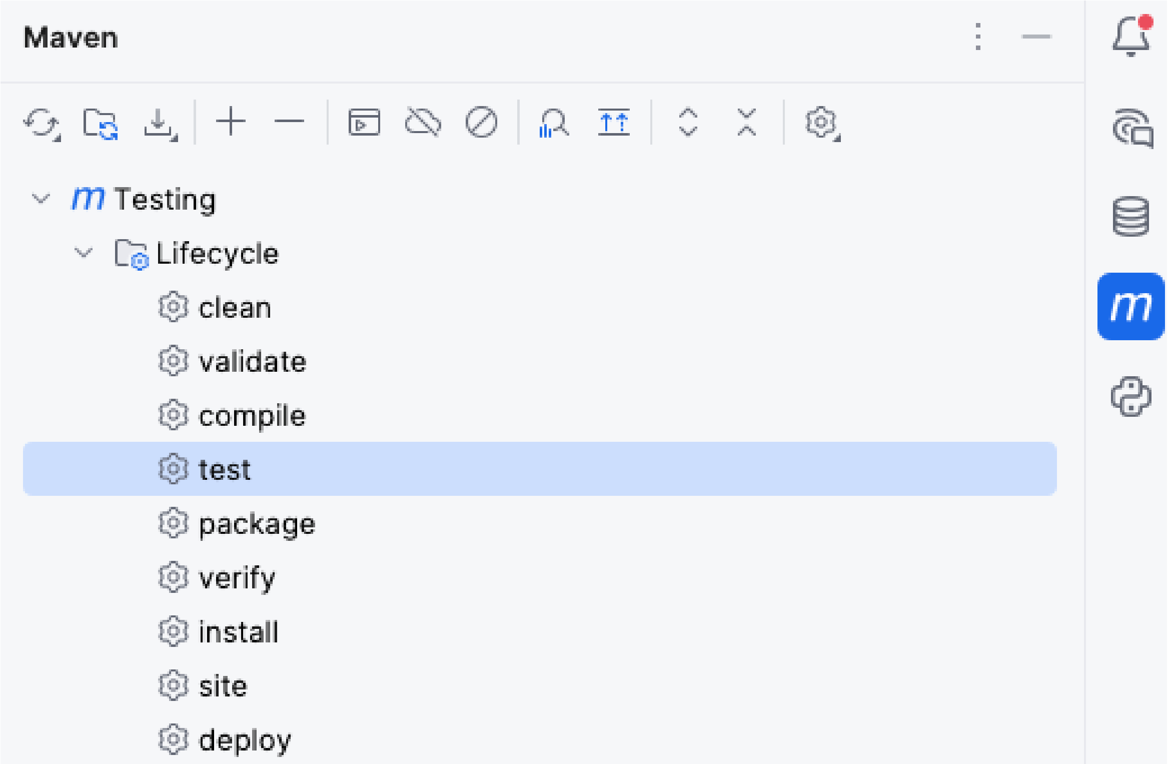
```
<includes>
    <include>**/*Test.java</include>
    <include>**/*Tests.java</include>
    <include>**/*UTest.java</include>
    <include>**/*IT.java</include>
    <include>**/*ITest.java</include>
</includes>
```

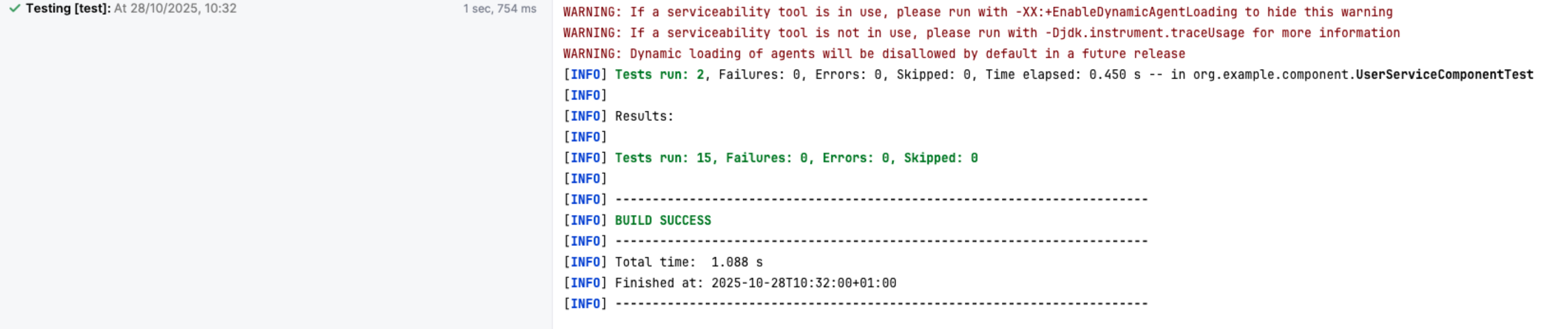


```
src
  main
    java
      org.example
        component
          User
          UserRepository
          UserService
        integration
          InMemoryUserRepository
          User
          UserRepository
          UserService
        unit
          Calculator
          EmailValidator
    resources
  test
    java
      org.example
        component
          UserServiceComponentTest
        integration
          UserServiceIT
        unit
          CalculatorUTest
          EmailValidatorUTest
```

IntelliJ Example

- Running tests during building
  - Go to the right side of IntelliJ
  - Press the M
  - Double click Testing>Lifecycle>test

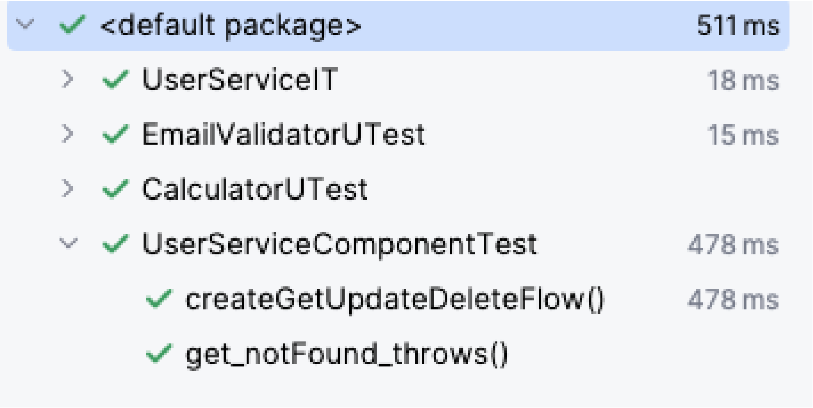




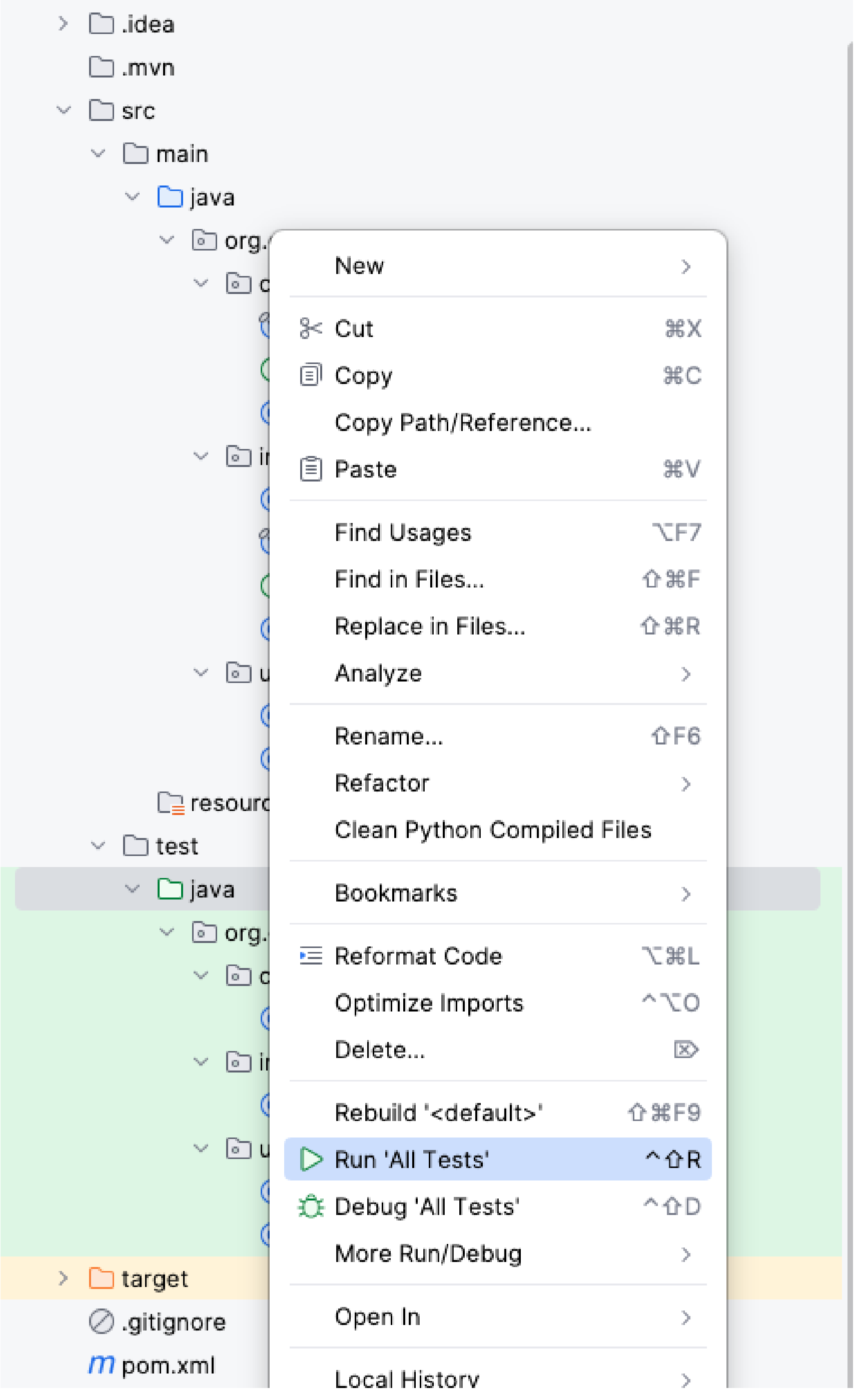
```
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.450 s -- in org.example.component.UserServiceComponentTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 15, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.088 s
[INFO] Finished at: 2025-10-28T10:32:00+01:00
[INFO] ------------------------------------------------------------------------
```

IntelliJ Example

- Running tests within package
  - Go to the right side of IntelliJ
  - Press the M
  - Double click Testing>Lifecycle>test



```
✓ <default package>                 511 ms
  ✓ UserServiceIT                    18 ms
  ✓ EmailValidatorUTest              15 ms
  ✓ CalculatorUTest
  ✓ UserServiceComponentTest        478 ms
      ✓ createGetUpdateDeleteFlow() 478 ms
      ✓ get_notFound_throws()
```



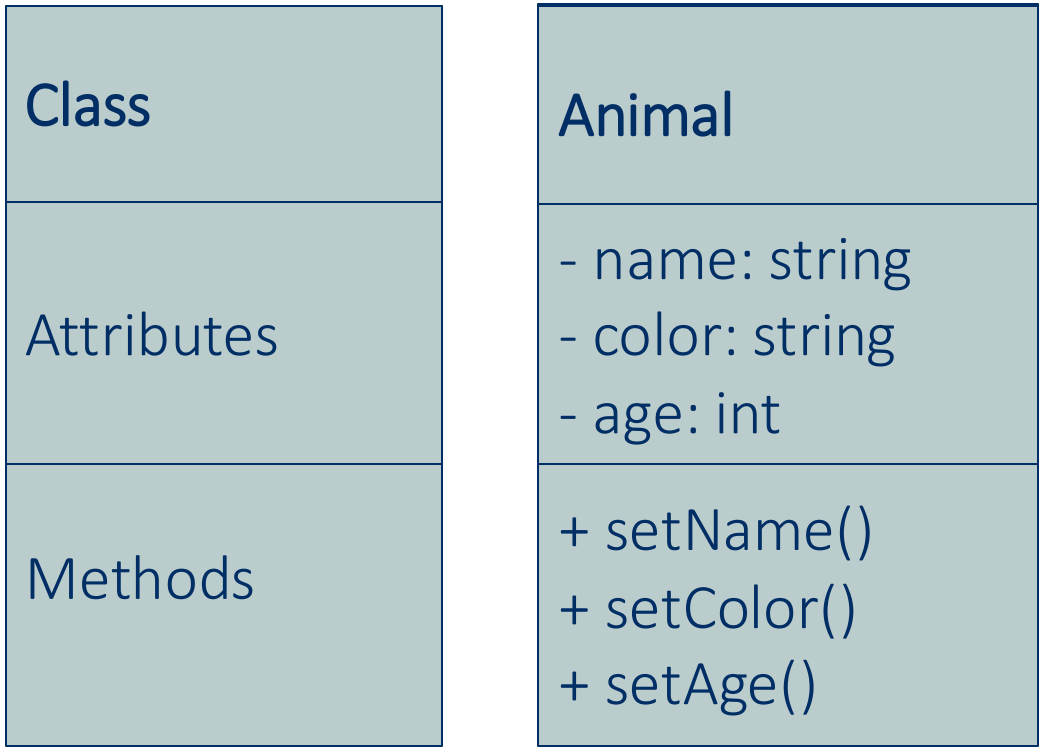
Part A
UML Diagrams

Class Diagrams

Class Diagrams

- **What is the purpose of a class diagram?**
  ➞ Defines structure of the architectural design

- **What does it look like?**



Class

Attributes

Methods

Animal

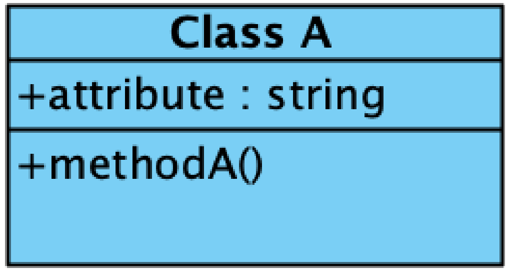
- name: string
- color: string
- age: int

+ setName()
+ setColor()
+ setAge()

Class Diagrams

Attributes & Methods

- Visibility

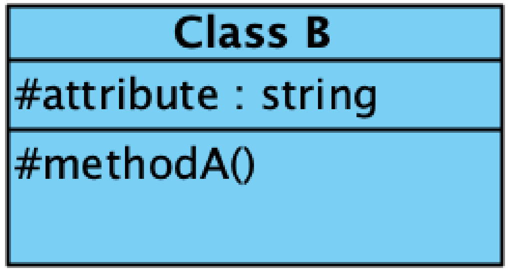


Class A

+attribute : string

+methodA()

Public

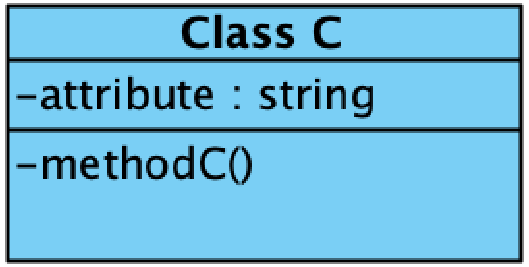


Class B

#attribute : string

#methodA()

Protected

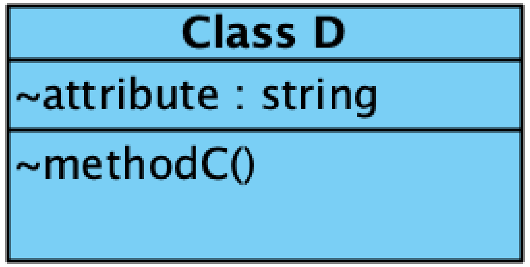


Class C

−attribute : string

−methodC()

Private



Class D

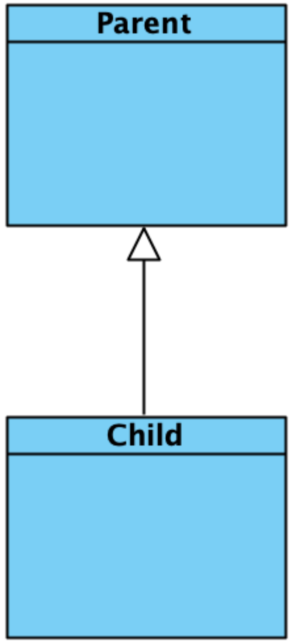
~attribute : string

~methodC()

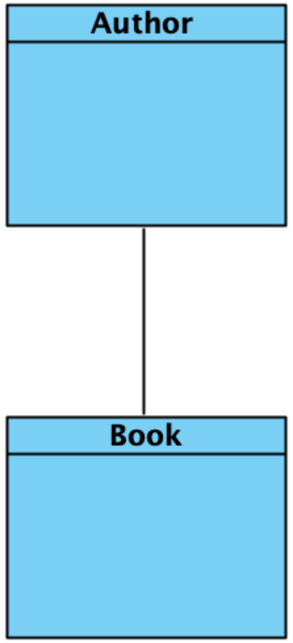
Package

Class Diagrams

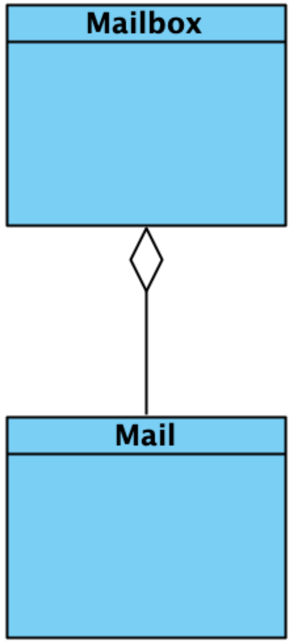
Relations



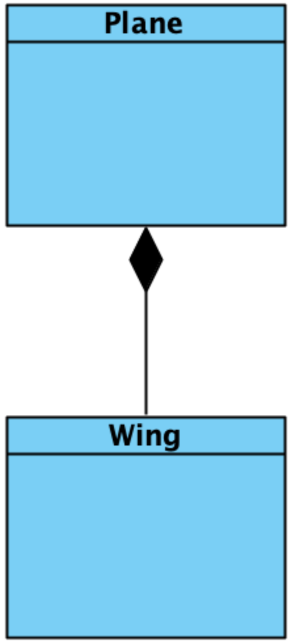
Inheritance



Association



Aggregation



Composition

Assignments

Class Diagrams

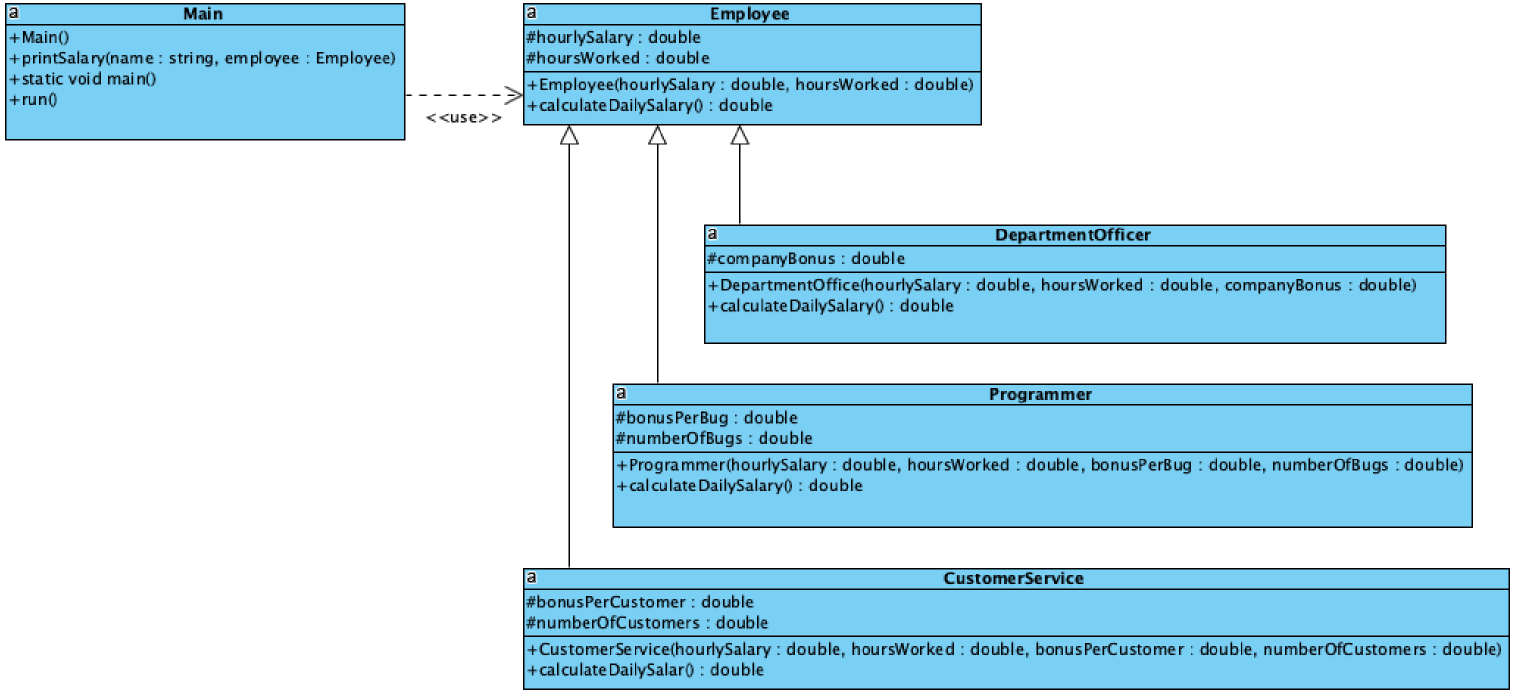
Assignment

- **4 sub assignments:**
  - Inheritance
  - Abstraction
  - Interfacing
  - Relations

- **Useful links:**
  - https://www.tutorialspoint.com/java/index.htm
  - https://beginnersbook.com/2013/04/oops-concepts/
  - https://www.javatpoint.com/java-oops-concepts

Assignment

Inheritance



Main
+Main()
+printSalary(name : string, employee : Employee)
+static void main()
+run()

<<use>>

Employee
#hourlySalary : double
#hoursWorked : double
+Employee(hourlySalary : double, hoursWorked : double)
+calculateDailySalary() : double

DepartmentOfficer
#companyBonus : double
+DepartmentOffice(hourlySalary : double, hoursWorked : double, companyBonus : double)
+calculateDailySalary() : double

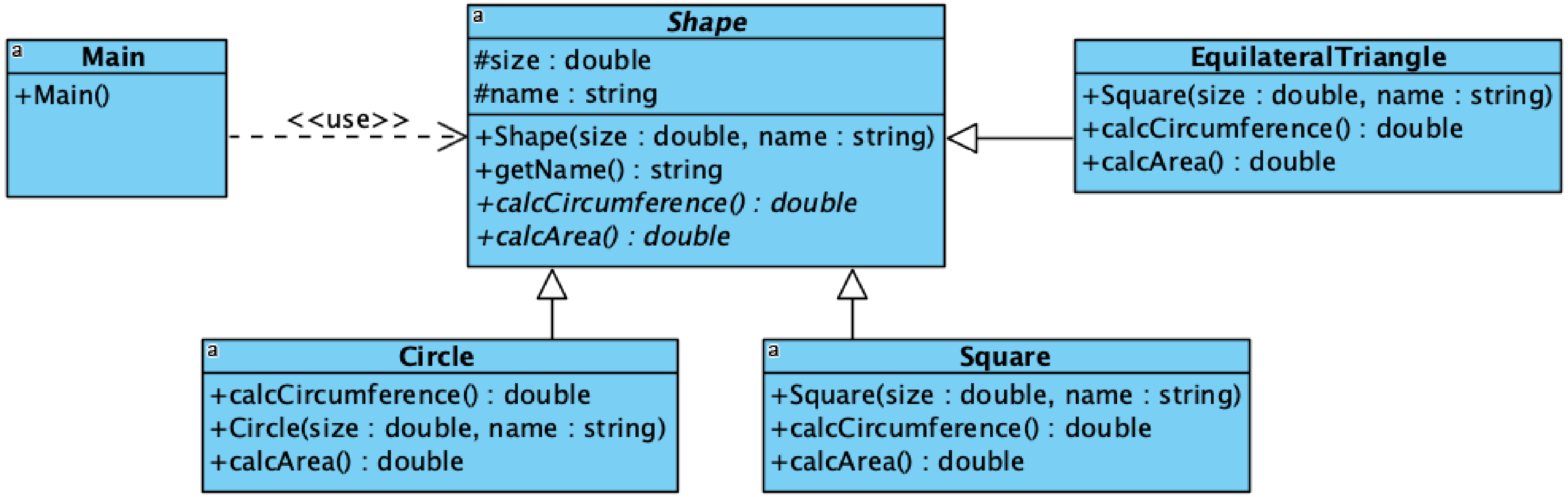
Programmer
#bonusPerBug : double
#numberOfBugs : double
+Programmer(hourlySalary : double, hoursWorked : double, bonusPerBug : double, numberOfBugs : double)
+calculateDailySalary() : double

CustomerService
#bonusPerCustomer : double
#numberOfCustomers : double
+CustomerService(hourlySalary : double, hoursWorked : double, bonusPerCustomer : double, numberOfCustomers : double)
+calculateDailySalar() : double

- For multiple employees: calculate their daily wage
  - Programmer: (hourly salary * hours worked) + (bonus per bug fixed * # bugs fixed)
  - CustomerService: (hourly salary * hours worked) + (bonus per customer helped * # customers)
  - DepartmentOfficer: (hourly salary * hours worked) + company bonus

Assignment

Abstractions



Main
+Main()

<<use>>

Shape
#size : double
#name : string
+Shape(size : double, name : string)
+getName() : string
+calcCircumference() : double
+calcArea() : double

EquilateralTriangle
+Square(size : double, name : string)
+calcCircumference() : double
+calcArea() : double

Circle
+calcCircumference() : double
+Circle(size : double, name : string)
+calcArea() : double

Square
+Square(size : double, name : string)
+calcCircumference() : double
+calcArea() : double

- Methods **calcCircumference()** and **calcArea()** are abstract in ***Shape***
- In main:
  - Make several different objects of **Circle, Square, and EquilateralTriangle**, all based on ***Shape***
  - Calculate circumference and area for both kinds of shapes
- Tests

Assignment

Interfaces

- **Use case: a simple universal remote control**
- User can:
  - Lower or raise volume of multiple devices (TV, radio, CD-player, …) that implement VolumeDevice interface
    → Execute remote.lowerVolume() to lower volume of all devices in the remote
  - Add a new VolumeDevice to the remote
- Devices comply to this interface:

```
public interface VolumeDevice
{
   void volumeUp();
   void volumeDown();
}
```

- Design the class diagram for this use case
  Pay attention to the way the interface is built
- Implement the necessary code to get this to work
- **Tests are not obligatory, but feel free to implement**

Assignment

Relations

- **Use case: a Film Festival**

- Develop a class diagram based on the given code
- **Pay attention to**
  - the associations: composite/aggregate/association/inheritance, multiplication
  - the difference between abstract classes and interfaces
  - Variables/Methods/Constructors
    - + / - / # / ~

Assignment

Class Diagrams

- **Portfolio:**
  - 5SD_Portfolio_FirstnameLastname.zip
    - Code repository (all labs):
      - Filled in version of the student git repository (working code and tests)
    - Exported Visual Paradigm project
      ➜ Zip export of Visual Paradigm project, with all your diagrams in it (sequences & use cases)
    - AI Usage Document ➜ for all labs