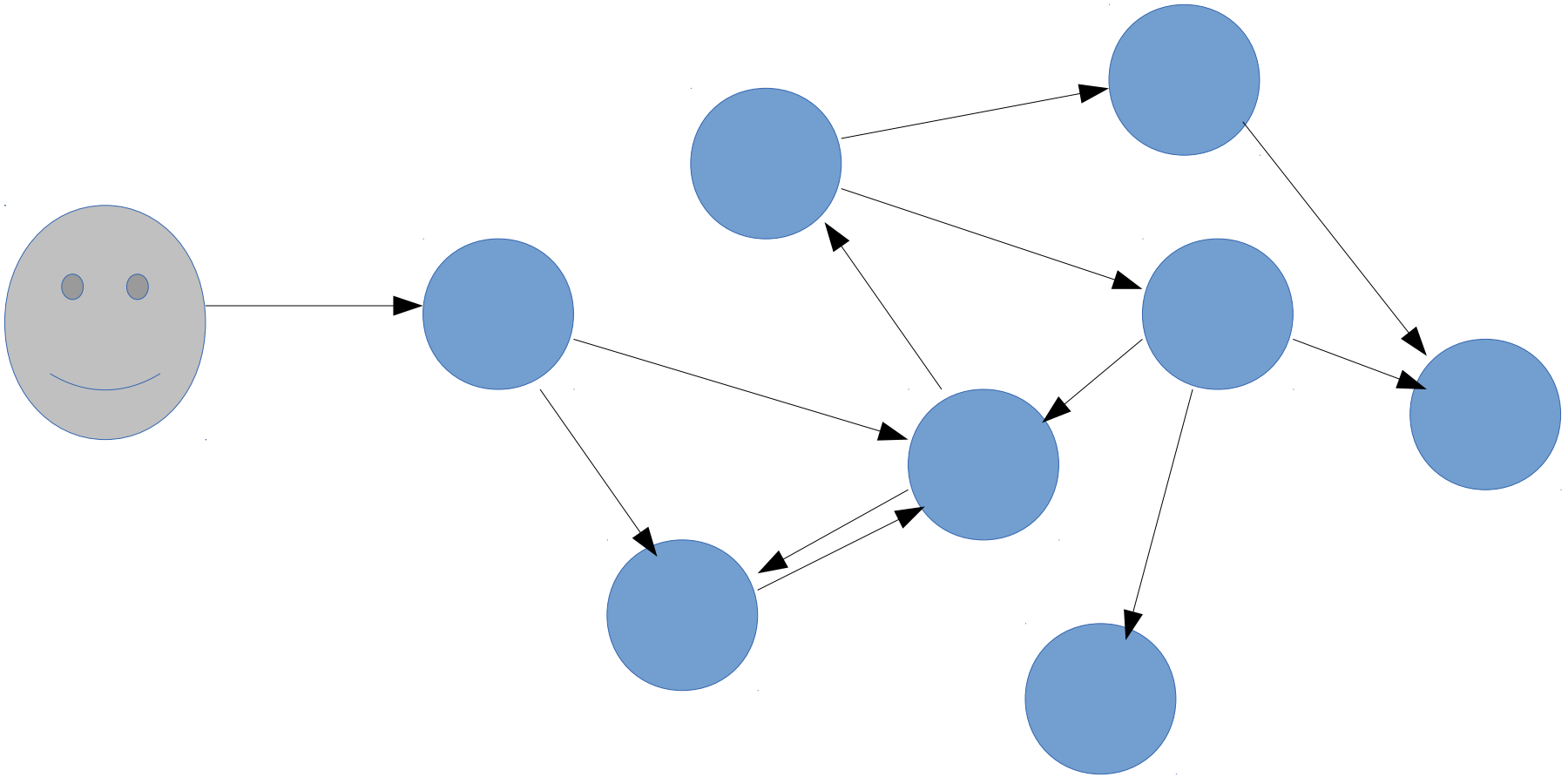
Testing units with dependencies

01219343 / Spring 2013

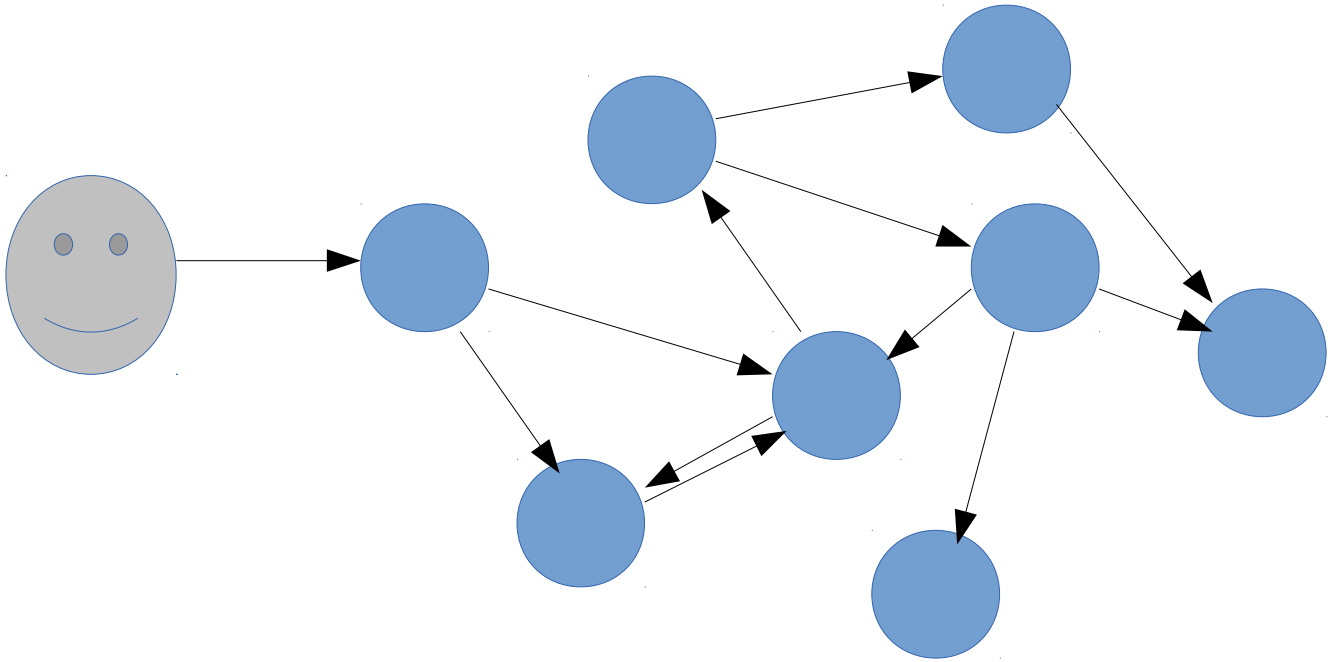
An Object-Oriented System

- The system consists of many objects (or units) working together.



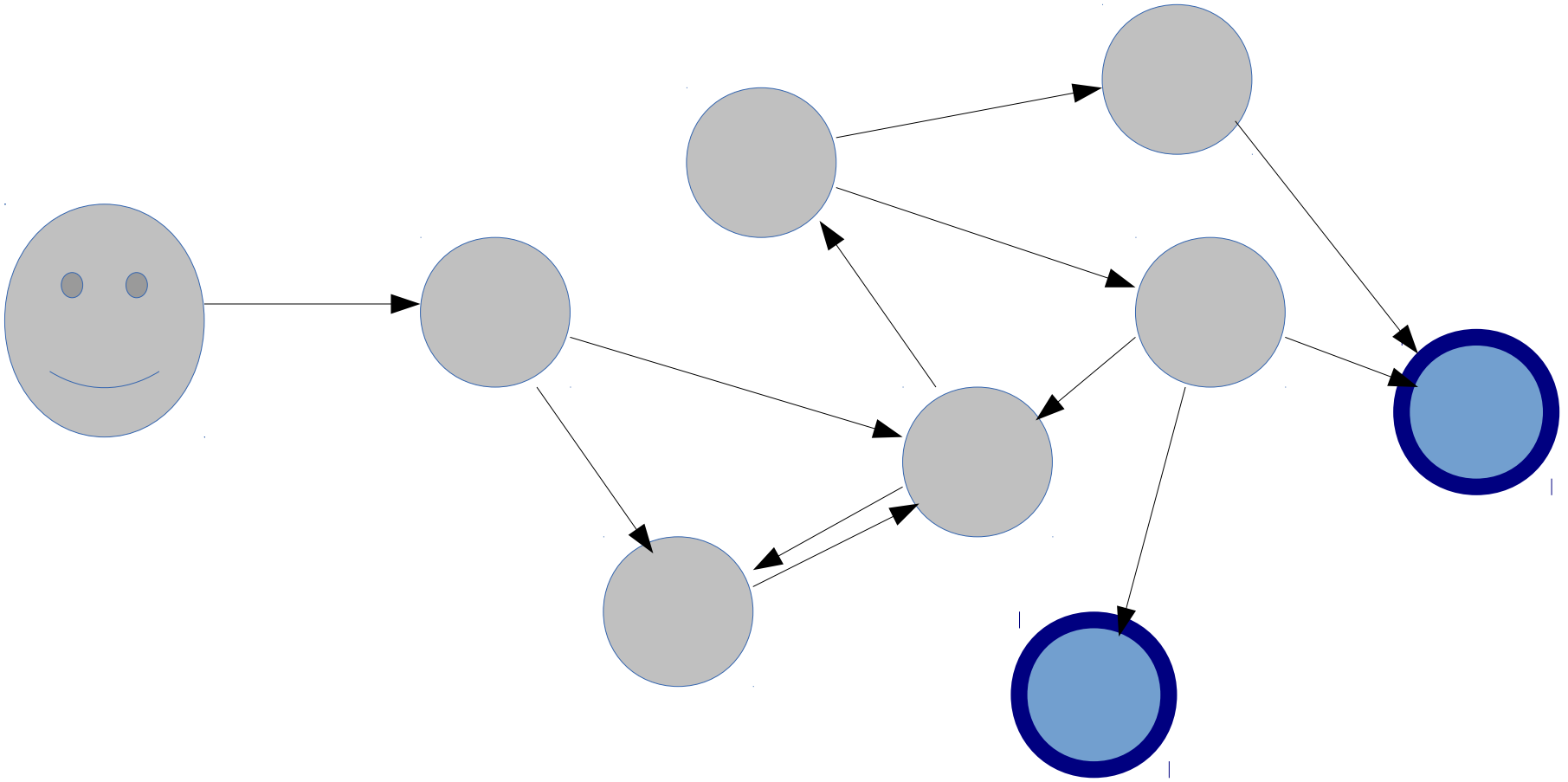
Types of testing

- Unit testing
- Integration testing
- End-to-end testing



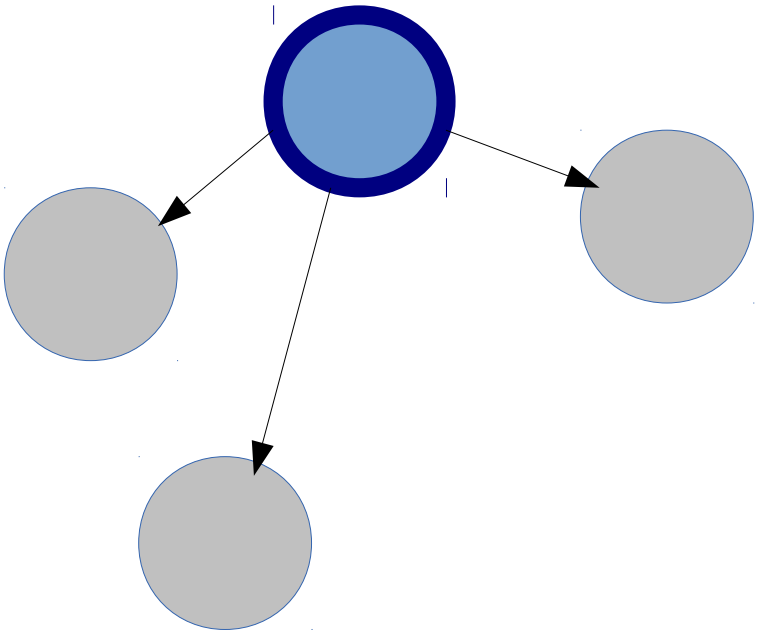
Easy-to-test units

- Isolated units are usually easy to test.



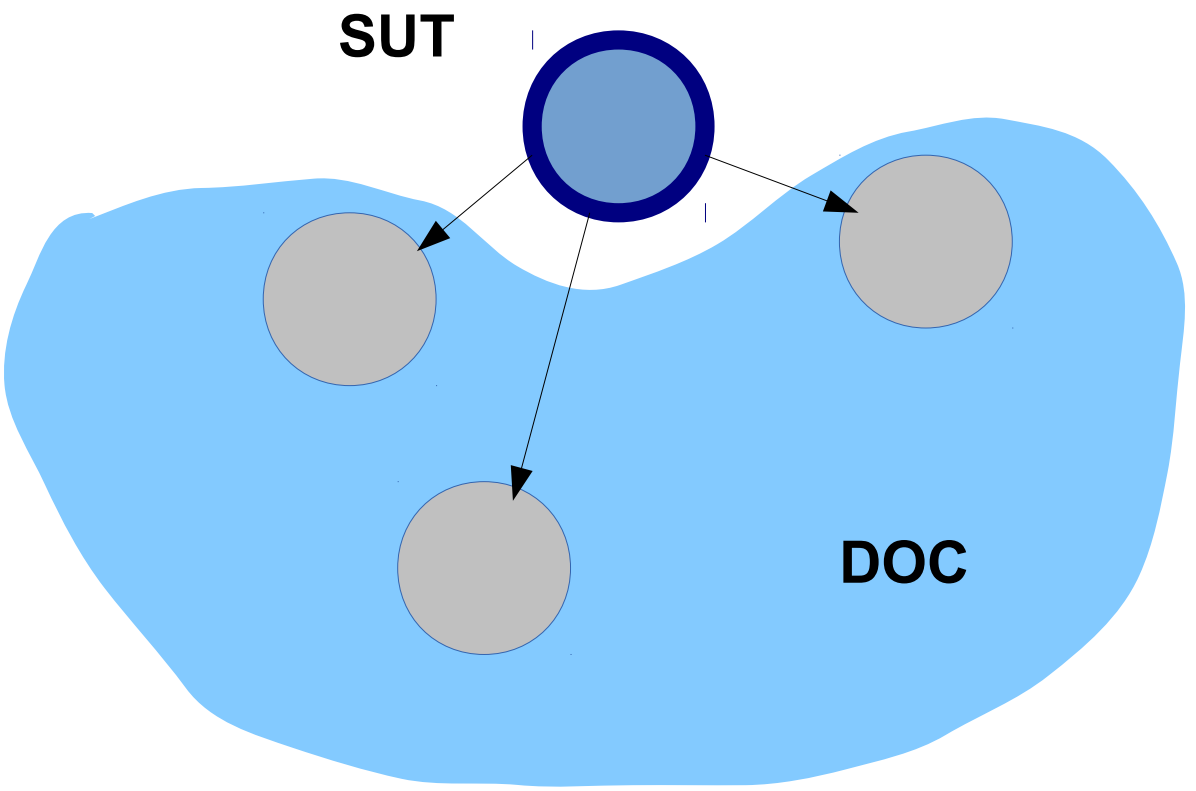
Units with dependencies

- You can only test units with dependencies only when the dependencies are provided.



Terms related to testing

- System Under Test (SUT)
- Depended On Component (DOC)



Practice: Test cases

- Recall a library exercise from last time.
- We would like to charge 5 baths for each late day, except weekends.
- Write a few test cases:

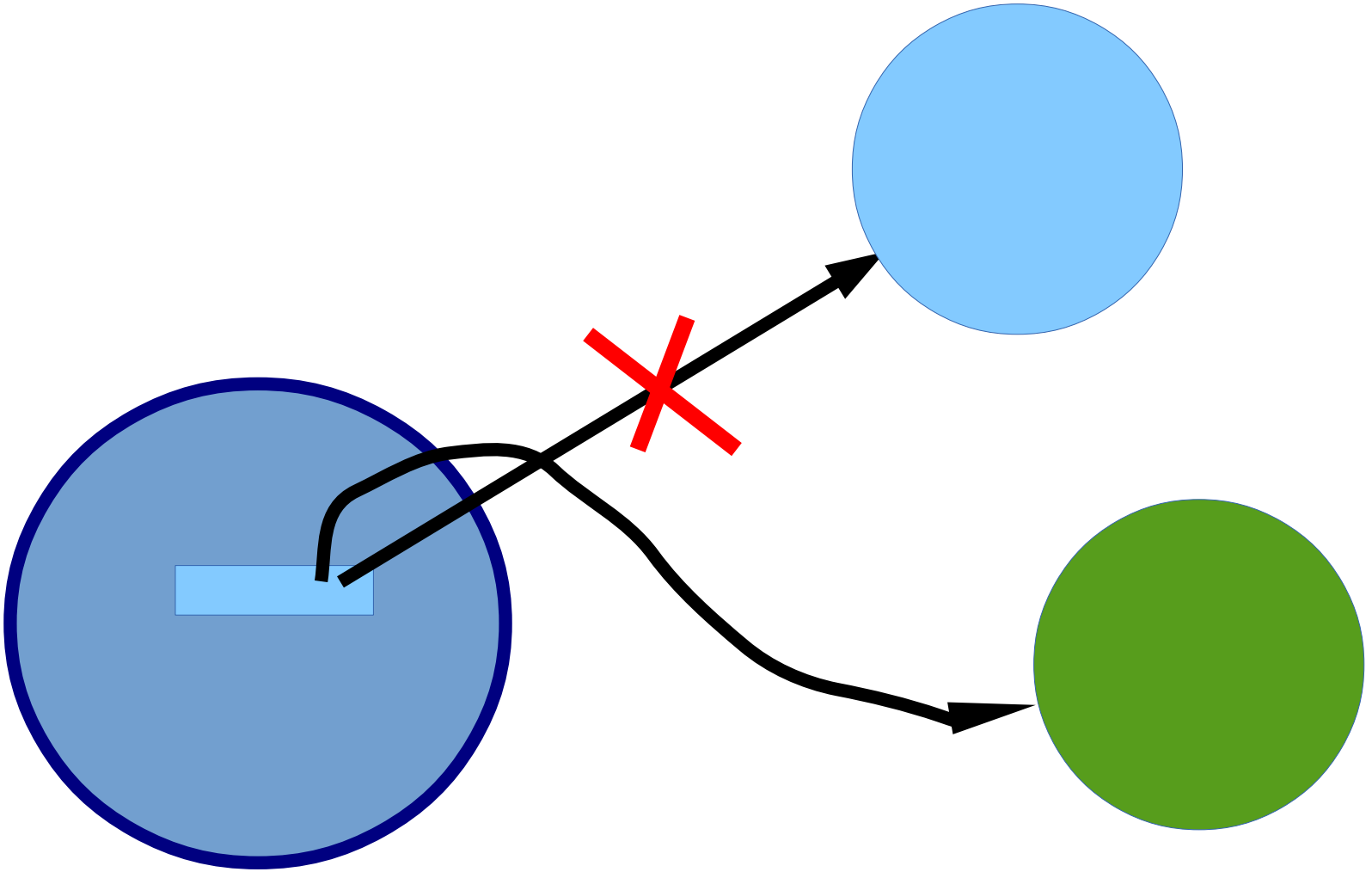
Case #	Due date	Today	Fee
1	Feb 17, 2013	Feb 25, 2013	tt94350943r

Hard-coded dependencies

- Consider the following example.

```
public class LateTimeCalculator {

    public static int lateFee(Date dueDate) {
        Date today = new Date();
        long seconds = today.getTime() - dueDate.getTime();
        int daysLate = (int) (seconds / 60 / 24);
        return daysLate * 5;
    }
}
```

Dependency Injection



Test doubles

- Test doubles are replacements for real DOC.
- There many types of test doubles.
  - Stub
  - Mock
  - Test spy

Stub

- We want to calculate the amount of tax a person has to pay.
- The current rate is 1000 bath per car.

```
public class Person {
      // ....
     public int getNumCars() {
          // do some database query, and return the result
          return 10;
     }
}
```

Stub

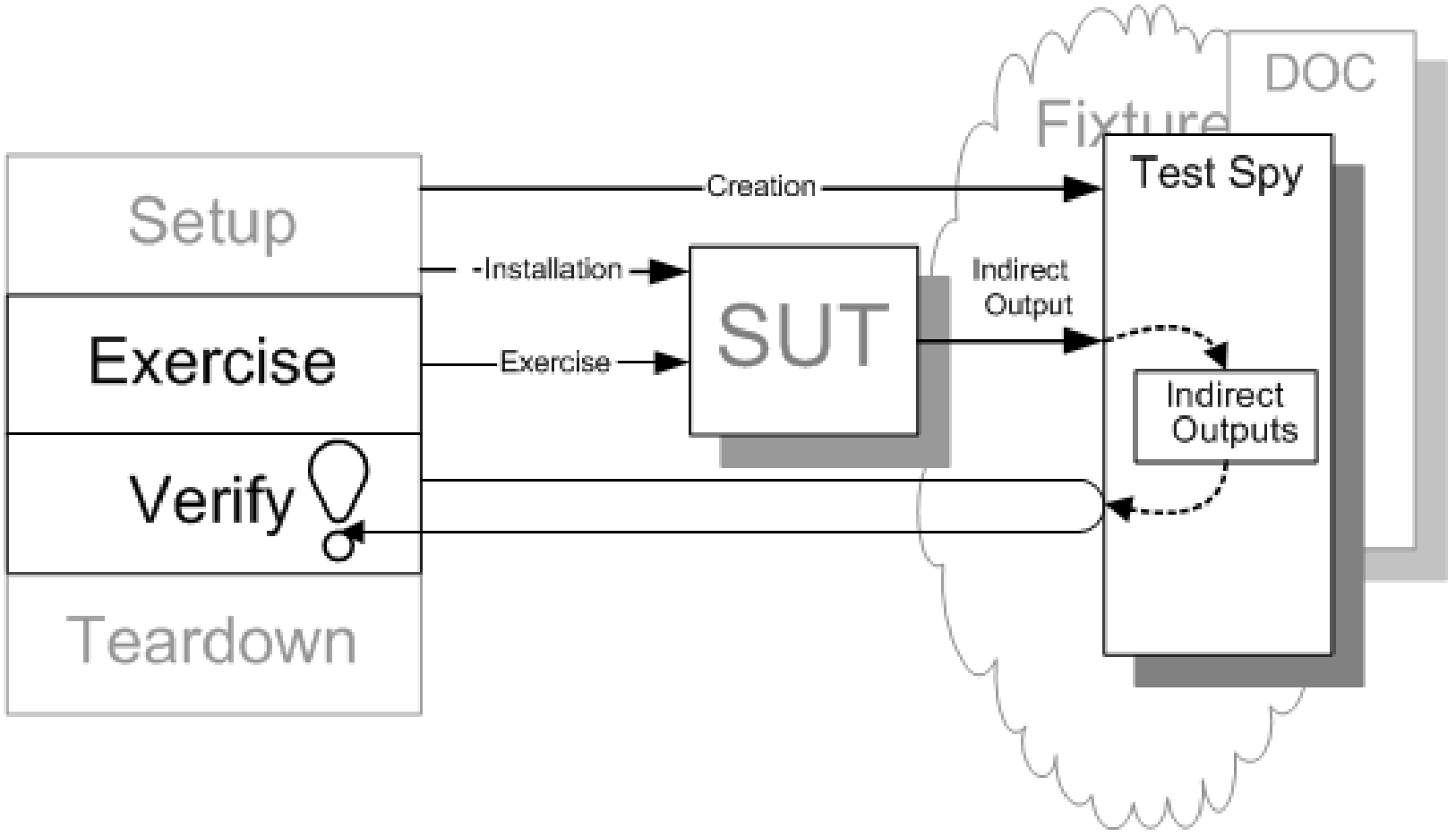
```
public class PersonStub extends Person {

    private int numCars;

    public void setNumCars(int c) { numCars = c;}

    public int getNumCars() { return numCars; }
}
```

```
    public void test1() {
        PersonStub p = new PersonStub();
        p.setNumCars(3);
        assertEquals(2000,cal.calculate(p));
    }
```

Test spies

- Test spies are objects that observes the interactions so that you can verify.
- It can also response to method calls.

Test spies



From: http://xunitpatterns.com/Test%20Spy.html

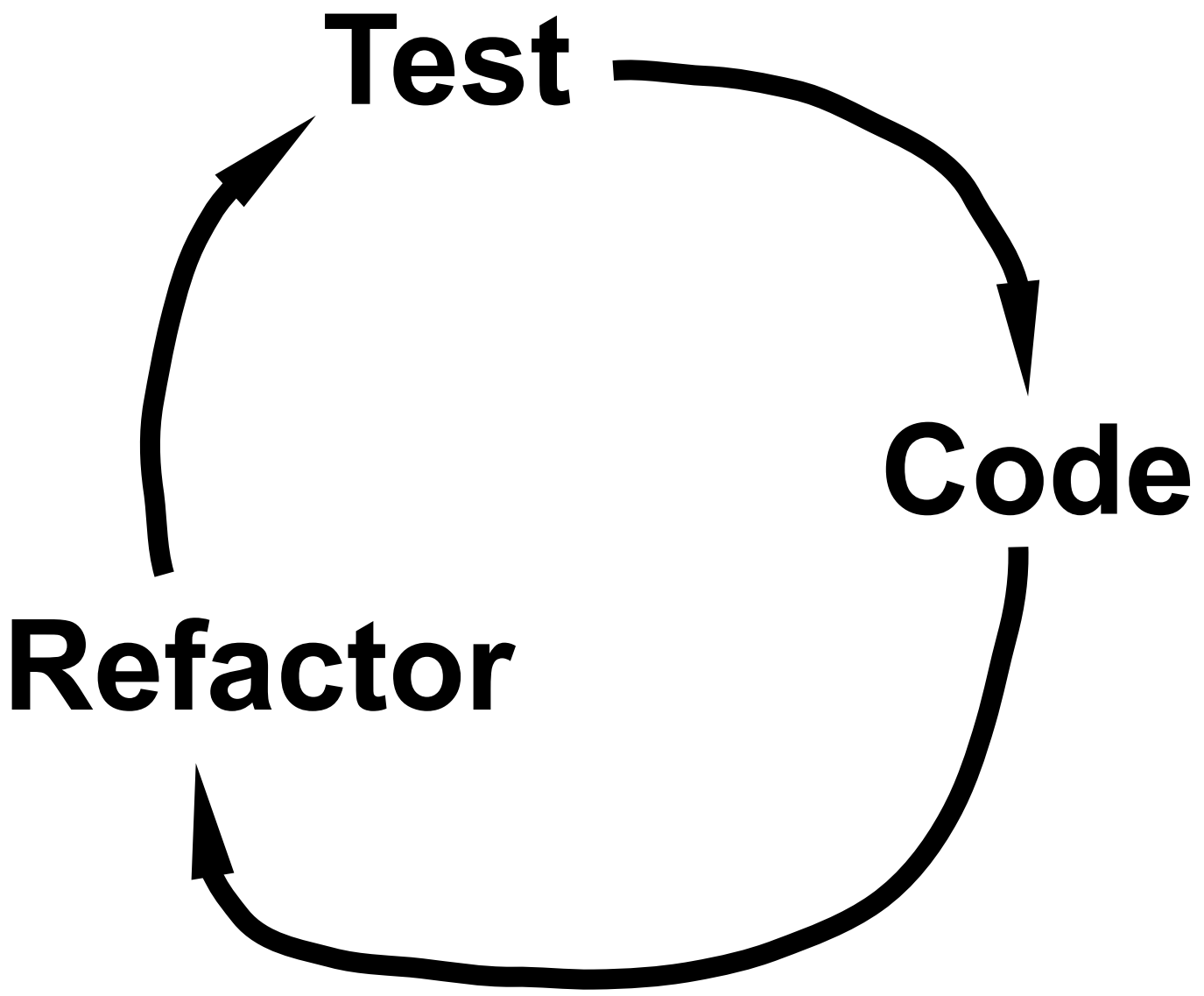
Demo 1

- You are writing an authentication module.
- To login, the user provides a user name which can be KU address or gmail address.
  - The user provide just user name, e.g., jtf or jittat, with no host information (i.e., no @ku.ac.th), then the system has to verify for both servers.
  - If the user provide the host explicitly (e.g., jtf@ku.ac.th), then the other authentication method will not be used.

Demo 1

- SUT: Authenticator
- DOC
  - KUAuthenticator
  - GMailAuthenticator

Review: TDD cycles



Exercise 1: KU Internet Quota

- Think about KU Internet Quota system.
- Go play at login.ku.ac.th, and see statistics at pi.ku.ac.th

Pair programming

- Split into pairs.
- One of you will take the keyboard and mouse and typing the codes
- The other one should help navigating.
- We shall switch roles for every 15 minutes.

Pair programming / driving

- **The driver** puts more concern on the details and make sure the car runs.
- **The navigator** looks ahead and think further to avoid problems, bugs, etc.

Pair programming

From: http://en.wikipedia.org/wiki/File:Pair_programming_1.jpg

Components

- What are the components in the KU Internet Quota system?

SUT: Quota calculation

- The feature that we would like to implement and test today is quota calculation.

The system should compute the remaining quota of each user correctly.

Tasks

The system should compute the remaining quota of each user correctly.

- Break the story down into smaller tasks.

TDD

- For each task, think about how your SUT interacts with other units.
- Write test; create test spies for the interaction.
- Write code to fix the test.
- Don’t forget to refactor.