School of Computer Science

UNIVERSIDAD DE OVIEDO

Unit 8: well-behaved objects

Introduction to Programming

Academic year 2023-2024

Concepts

- Testing
- Testing automation
- Debugging
- Exceptions

Think…

```
public void test() {
    int sum = 1;

    for (int i = 0; i <= 4; i++); {
        sum = sum + 1;
    }


    System.out.println(“The result is: " + sum);
    System.out.println(“Its double is: " + sum+sum);
}
```

What is displayed?

The expected result:

```
The result is: 6
Its double is: 12
```

The real result:

```
The result is: 2
Its double is: 22
```

Think…

```
public void test() {
   int sum = 1;

   for (int i = 0; i <= 4; i++); {
      sum = sum + 1;
   }


   System.out.println(“The result is: " + sum);
   System.out.println(“Its double is: " + (sum+sum));
}
```

What is displayed?

The expected result:

```
The result is: 6
Its double is: 12
```

The real result:

```
The result is: 2
Its double is: 22
```

Dealing with errors

- When you begin learning a programming language, initial errors are usually **syntactic errors**
  - The compiler will mark them in the source code.
- Later, most of the errors are **logic errors**.
  - The compiler cannot help us to find them.
  - These errors are also known as **bugs**.
- Some logic errors do not reveal themselves immediately.
  - Commercial software sometimes have bugs.

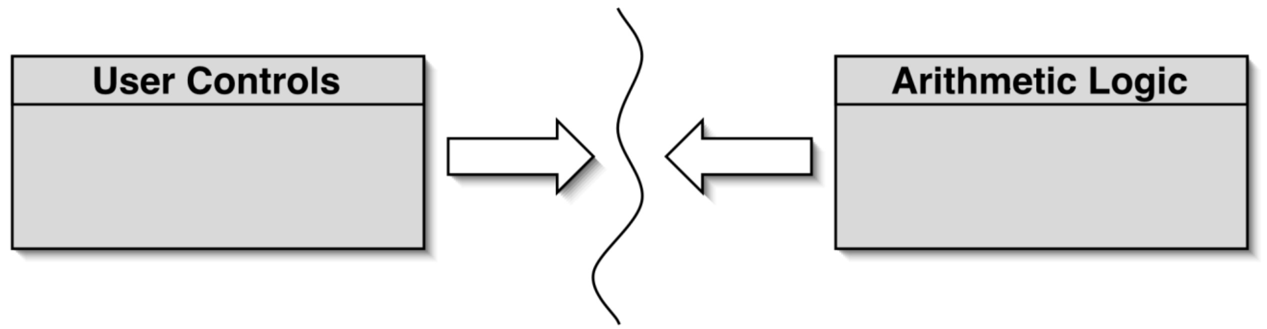
Prevention vs detection (Development vs maintenance)

- We can reduce the **probability of error occurrence**
  - Using software engineering techniques like encapsulation and information hiding.
- We can **improve detection**
  - Using software engineering good practices like modularization and documentation.
- We can also develop **detection skills**

Modularization and interfaces

- Applications are made up of cooperating modules.
  - Different teams may be responsible of the development of each of them.
- The interface among modules must be <u>clear and well defined</u>.
  - It increases the probability of success when integrating them.
  - Enforces the simultaneous and independent development of each module.

Modularization example

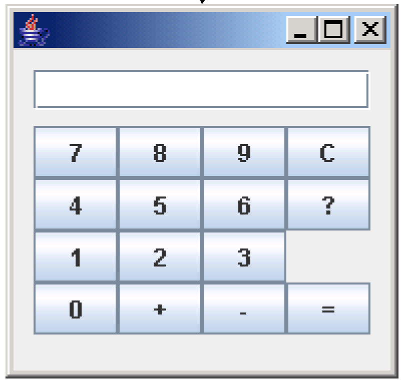


- A calculator.
- Each module does not need to know the implementation details of the other.
  - User controls could be a GUI or a hardware device.
  - The calculator logic could be implemented in hardware of software.

Interfaces between modules

```
// It returns the value to be shown.
public int getValueOnDisplay();

// Invoked when a digit is pressed
public void digitPressed(int number);

// Invoked when button + is pressed
public void plus();

// Invoked when button - is pressed
public void minus();

// Invoked to end a calculation
public void equal();

// Invoked to initialize the calculator
public void clean();
```

User interface



Application logic interface

Testing and debugging

- **Testing** and **debugging** are two key skills.
- When **testing**, you search for errors in the source code.
- When **debugging**, you search for the cause of errors.
  - Sometimes, an error reveals itself far away from the point in the source code that caused it.

Testing and debugging techniques

- **Testing** techniques
  - “Handmade” unit testing
  - Automatic testing
- **Debugging** techniques
  - Manual monitoring
  - Verbal monitoring
  - Printing sentences
  - Debuggers

Unit testing

- Each application unit can be tested
  - Methods, classes, modules (packages, in Java)
- Unit testing is usually done during application development
  - Early error finding and solving reduces development costs (e.g., programmers time).
  - It is possible to build test cases that will be used time and time again (as the system grows).

Essential tests

- The unit **contract**: what the unit is expected to do
  - Possible **contract infringements** must be searched for.
  - **Positive tests** (the system does what it is supposed to do with valid input data) and **negative tests** (the system does not do what it should not with invalid input data) will be used.
- Test in the limits
  - Regarding collections: empty, full, containing some elements.
    - Search for an element in an empty collection.
    - Add an element to an empty collection.

“Handmade” unit testing in BlueJ

- You can create instances of each class in the project.
- Methods may be invoked individually.
- You can see how a method execution affects the state of objects by using the **inspect** operation.
- But
  - Making good tests is a creative process, they require a lot of time and are very repetitive.

Automatic testing

- Regression testing
- JUnit
  - Test cases
  - Assertions
  - Fixtures

Regression testing

- **Regression testing** consists in repeating tests execution to ensure that changes made in the code have not introduced new bugs.
- The use of a testing tool may decrease their inherent workload
  - Classes that perform the tests are implemented (they contain the test methods).
  - Creativity focuses in creating those classes.
  - Those classes are named unit tests because they test individual classes.

Testing with JUnit

- Each test class is associated to an ordinary class of the project
- Automatic **regression tests** will be more effective if
  - It is possible to build auto controlled tests.
  - Human intervention is only necessary if an error appears.
- JUnit is a testing framework for the Java language.

Testing with JUnit

- Each test class created with JUnit contains:
  - Source code to execute the tests over a given class.
  - Source code to control t

ult execution of the tests by using assertions.
- An **assertion** is an expression that states a condition expected to be true. If it is finally false, the assertion failed and so, there is a bug in the program.
- A **fixture** is a fixed state of a set of objects used as a baseline for running tests.

Debugging techniques

- **Debugging** techniques
  - Manual monitoring
  - Verbal monitoring
  - Printing sentences
  - Debuggers

Debugging

- It is very important to develop skills in source code reading.
  - Sometimes you have to debug code written by others.
- There exist techniques and tools that help the programmer in the debugging process.

Manual monitoring

- A source code fragment is considered, studying it line by line, and any change in the state of the objects involved or any other behavior of the application must be taken into account.
  - You use paper and a pencil to simulate what happens in the computer (in the JVM) when the code is executed.
- Not very used technique nowadays.
  - Very low level.
- But very useful sometimes.

Verbal monitoring

- You have to explain another person what a class or method does.
  - The listening person may identify the bug.
  - Simply “reading out loud” what a code fragment is expected to do is enough to figure out what the bug is.

Printing sentences

- It consists in **adding** to methods temporary **printing sentences** or adding to the class **printing methods** (only invoked if debugging is on).
- They provide a lot of information:
  - Which methods were invoked.
  - The values of the parameters.
  - The order in which methods were invoked.
  - The values of local variables and attributes in interesting places in the code.

Printing sentences

- Disadvantages:
  - It is impractical.
  - Adding too many printing sentences may lead us to lose sight of important information.
  - Whenever you finish using them, removing them is tedious.
  - If you need them again, you have to write them again!

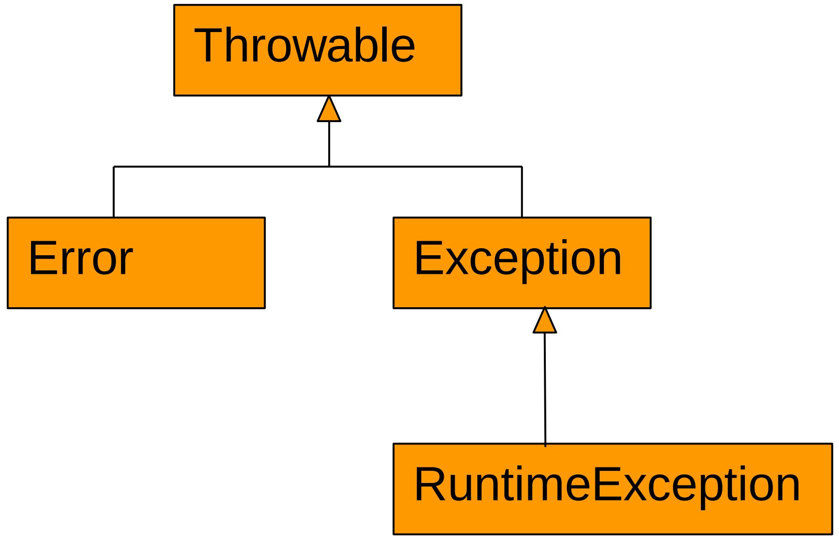
Using a debugger

- A **debugger** is a software tool that provides the necessary help to monitor the execution of a code fragment.
  - Break points are set to monitor the execution step by step.
  - It provides information about the method invocation stack.
- Advantage
  - They do not modify the state of the objects.
- Disadvantage:
  - It does not exist a recording of all the state changes so it is not possible to “rewind” the execution.

Exceptions

- When an error is detected, the user must be informed.
- The **Java exception mechanism** allows the **server object** (the one whose method is executing) to send a signal to the **client object** (the one from which the method was invoked) when something goes wrong.

Exceptions hierarchy



Creating and throwing exceptions

- Exceptions are objects.
- This term, we will use only exceptions of the Java class library.
  - **RuntimeException** class.
- Exception object creation

```
new RuntimeException(“Message describing the error”);
```

- Throwing an exception

```
throw new RuntimeException(“Message describing the error”);
```

Catching an exception

- **try-catch** blocks are used to **catch exceptions**

```
try {
        // sentences to protect from
        // potential errors
}
 catch (RuntimeException e) {
      // Process here the exception of the type RuntimeException
      // and recover from it
 }
```

Summary

- Bugs are part of programs lifetime.
- You can reduce their appearance using good software engineering techniques.
- Skills in test development and debugging are essential.
- Building tests should become a (good) habit.
- Tests must be automatized as much as possible.