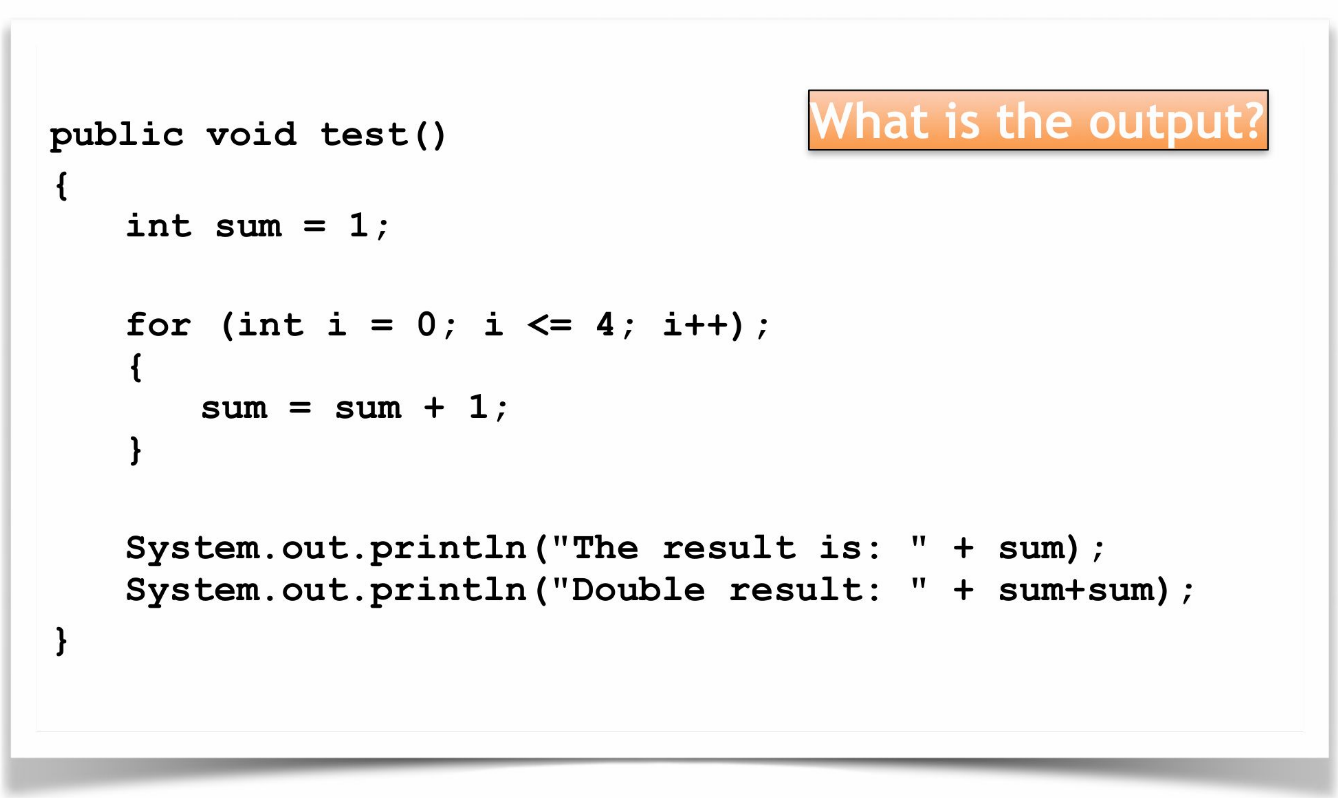
Introduction to Systems Programming

Well-behaved objects. Software Testing

2025 - Simón Gutiérrez Brida *(Based on material by Dr. Nazareno Aguirre)*

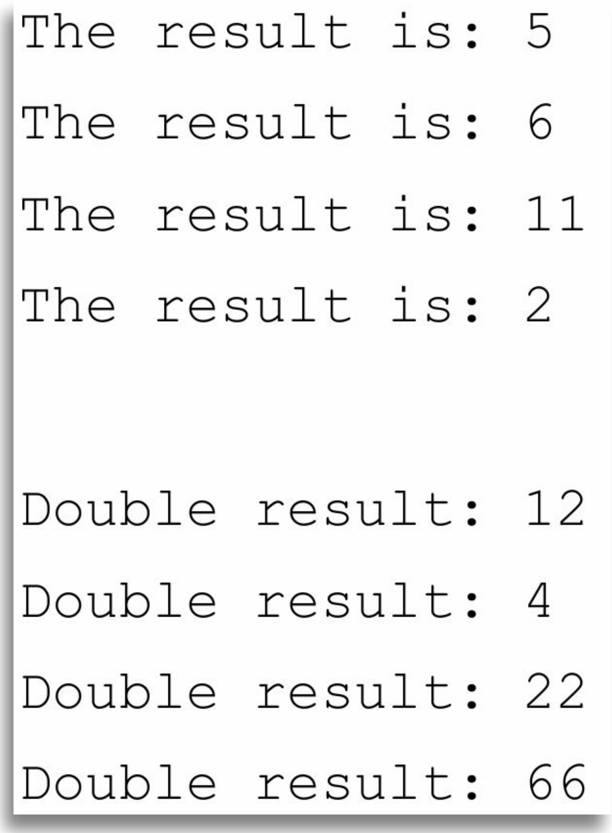
A simple code snippet



```
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("Double result:  " + sum+sum);
}
```

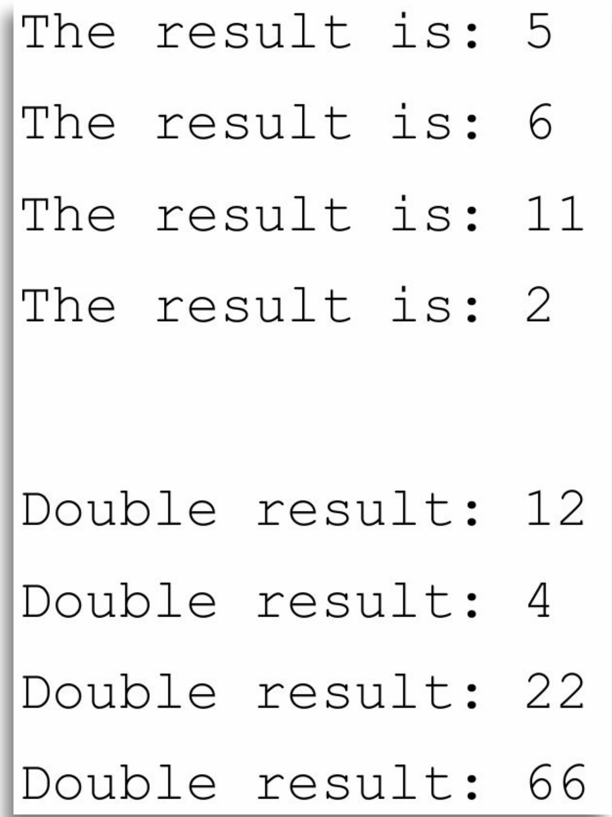
What is the output?

Possible outputs

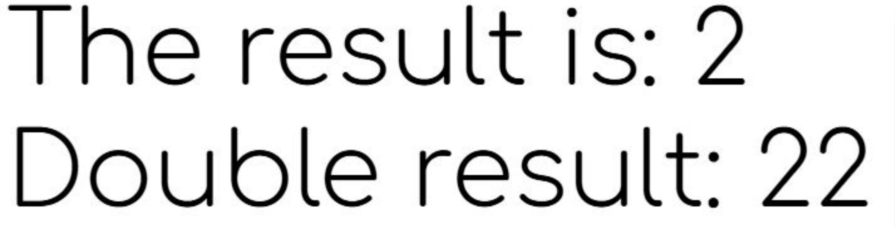


```
The result is: 5
The result is: 6
The result is: 11
The result is: 2


Double result: 12
Double result: 4
Double result: 22
Double result: 66
```

Possible outputs

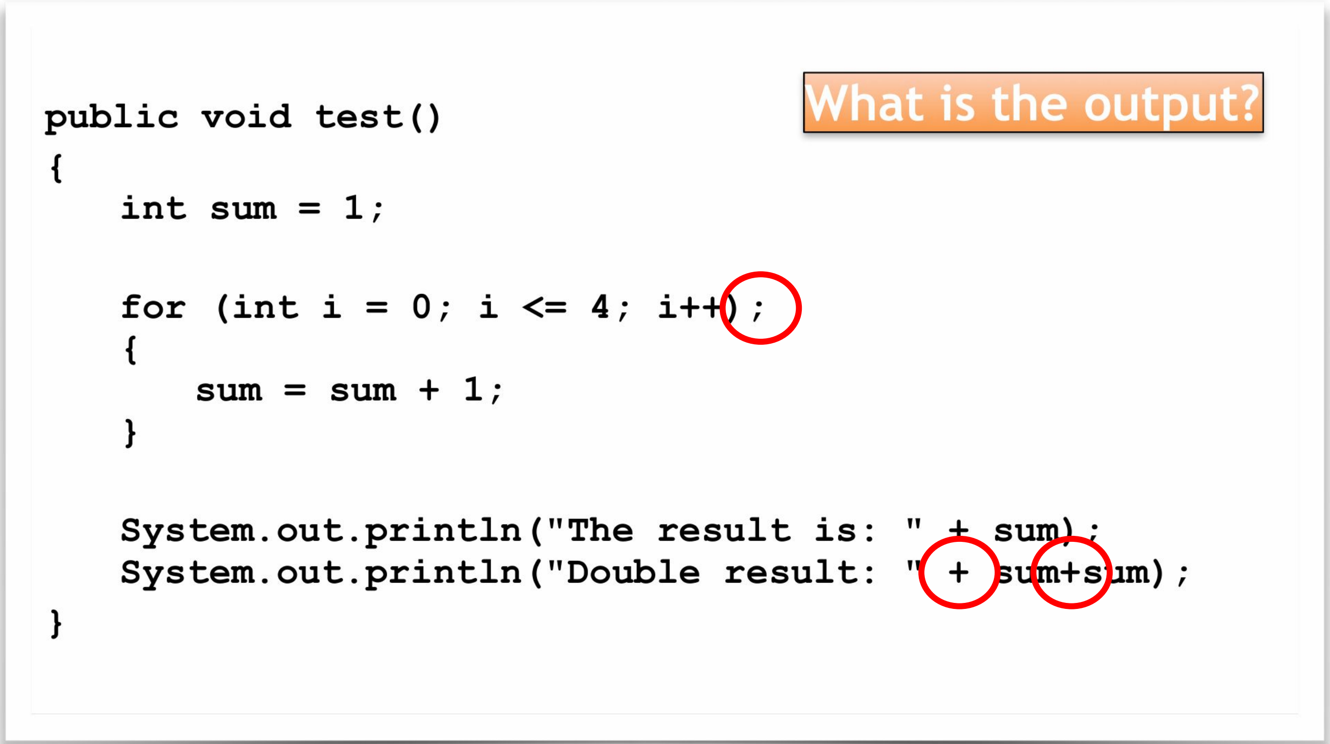


```
The result is: 5
The result is: 6
The result is: 11
The result is: 2


Double result: 12
Double result: 4
Double result: 22
Double result: 66
```



```
The result is: 2
Double result: 22
```

A simple code snippet



What is the output?

```
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("Double result: " + sum+sum);
}
```

We have to deal with errors

- Early (simpler) errors are usually syntax errors.
    - The compiler will spot these.
- Later (more complex) errors are usually logical errors.
    - The compiler usually cannot help with these.
    - Also known as logical bugs.
    - Some logical errors have no immediately obvious manifestation.
    - Commercial software is rarely error free.

Prevention vs Detection

Developer vs Maintainer

- We can reduce the likelihood of errors.
    - Use software engineering techniques, like encapsulation.
    - Pay attention to cohesion and coupling.

- We can improve the chances of detection.
    - Use software engineering practices, like modularization and good documentation.

- We can develop detection skills (gain experience).

Testing and Debugging

- These are crucial skills.

- Testing searches for the presence of errors.

- Debugging searches for the source of errors.
  - The manifestation of an error may well occur in a ‘distant’ location from its source.

Detecting a bug with tests (the RIPR model)

- Reachability : Tests cause faulty statements to be reached
- Infection : Tests cause faulty statement to result in an incorrect state.
- Propagation : The incorrect state propagates to incorrect output.
- Revealability : The oracles must observe part of the incorrect output.

Unit Testing

- Unit testing: test the behavior of a unit of software as independently of its context as possible.

- Each unit of an application may be tested.
    - Method, class, module (package in Java).

- Can (should) be done during development.
    - Finding and fixing bugs as early as possible reduces development costs (e.g., development/programming time).

Testing fundamentals

- Understand what the unit should do – its contract.
  - You will be looking for violations.

  - Use positive tests and negative tests.

- Test objectives
  - Try to thoroughly cover the unit, e.g.: cover as many statements as possible, as many branches as possible, etc

- Test boundaries in the behavior, e.g.: search an empty collection, add to a full collection, etc.

Testing fundamentals

- Understand what the unit should do – its contract.
    - You will be looking for violations.

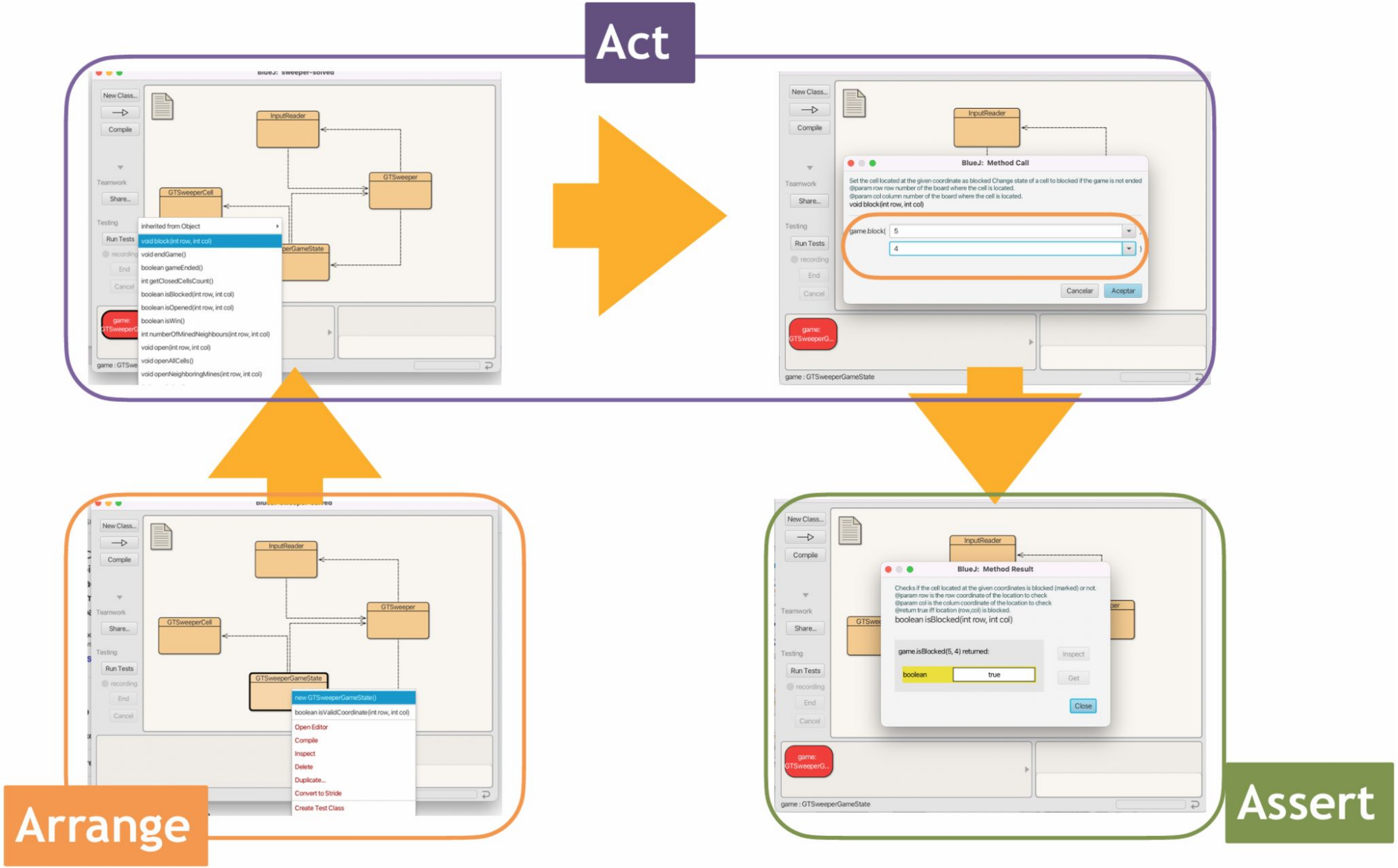
    - Use positive tests and negative tests.

Positive tests: Test correct behaviour on valid scenarios.
Negative tests: Test correct behaviour on invalid scenarios.

Components of a Unit Test

- Arrange: preparation of the scenario.
    - State and inputs/arguments necessary for testing the unit.
- Act: this is the execution of the unit being tested.
    - It typically just involves calling the software under test in the prepared scenario.
- Assert: captures the expectations on the execution of the test, i.e., the expected behavior (if the software were correct).
    - It requires understanding precisely what the software is supposed to do in the given context for the given data.
    - Checks expected behaviour against actual behaviour, e.g.: asserts the output is the output that was expected.

Example (ad hoc unit testing)



BlueJ: Method Call

Set the cell located at the given coordinate as blocked Change state of a cell to blocked if the game is not ended
@param row row number of the board where the cell is located.
@param col column number of the board where the cell is located.
void block(int row, int col)

game.block(5 , 4)

Cancelar Aceptar

BlueJ: Method Result

Checks if the cell located at the given coordinates is blocked (marked) or not.
@param row is the row coordinate of the location to check
@param col is the colum coordinate of the location to check
@return true iff location (row,col) is blocked.
boolean isBlocked(int row, int col)

game.isBlocked(5, 4) returned:

boolean true

Inspect Get Close

2025 - Simón Gutiérrez Brida

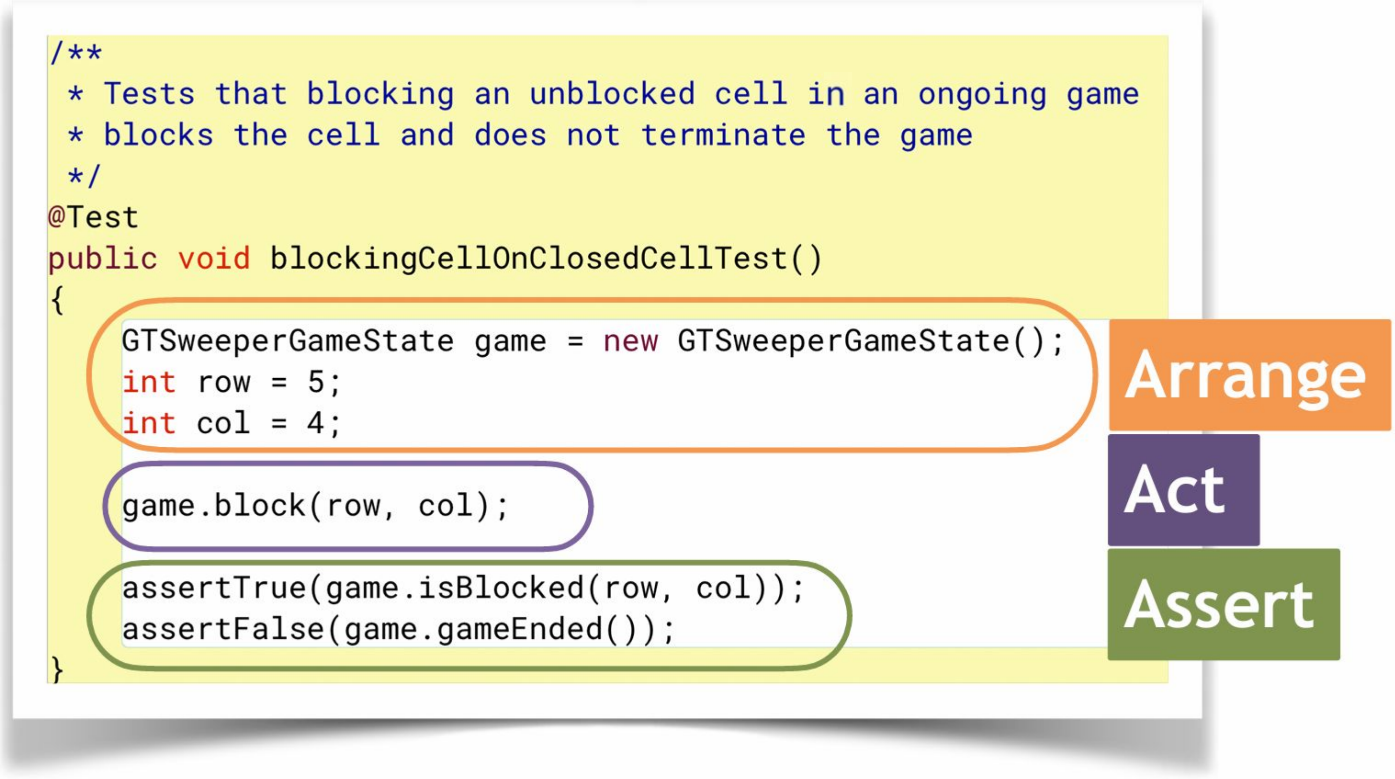
Drawbacks of ad hoc testing

- Inconvenient for repeated testing.
  - It does not store tests for future runs.

- Cumbersome and error-prone as the setting of the environment becomes more complex.
  - Too many actions or method calls to set the environment in a testing condition.

- It requires human intervention to attest if test “passes”.
  - Developer has to examine software outputs to assess if software behaves as expected

Example using testing framework (JUnit)



```
/**
 * Tests that blocking an unblocked cell in an ongoing game
 * blocks the cell and does not terminate the game
 */
@Test
public void blockingCellOnClosedCellTest()
{
    GTSweeperGameState game = new GTSweeperGameState();
    int row = 5;
    int col = 4;

    game.block(row, col);

    assertTrue(game.isBlocked(row, col));
    assertFalse(game.gameEnded());
}
```

Arrange

Act

Assert

Tests automation

- Good testing is a creative process, but thorough testing is time consuming and repetitive.

- Regression testing involves re-running tests.

- Use of a test rig or test harness can relieve some of the burden.

JUnit

- JUnit is a Java test framework.
- Test cases are methods that contain tests.
- Test classes contain test methods.
- Assertions are used to assert expected method results.
- Fixtures are used to support multiple tests (allows to define the same scenarios for several tests).

Modularization and interfaces

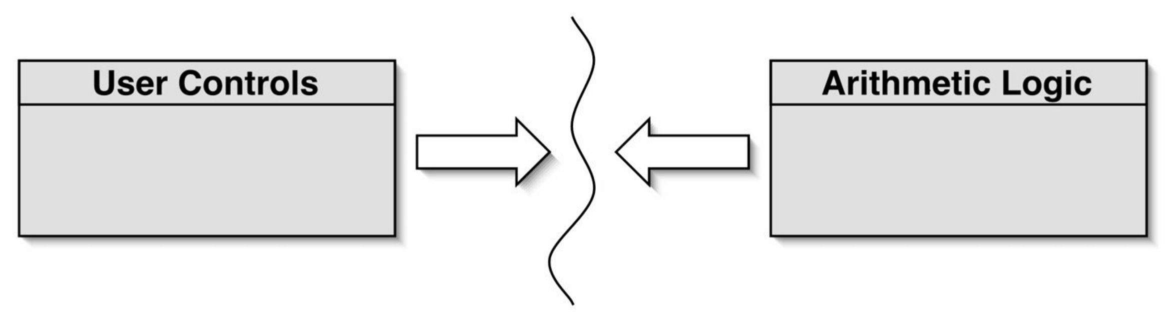
- Applications often consist of different modules.
  - E.g.: to separate classes into logical modules (packages).

- The interface between modules must be clearly specified.
  - Provides a level of abstraction and modularization that increases software quality.

  - Supports independent concurrent development.

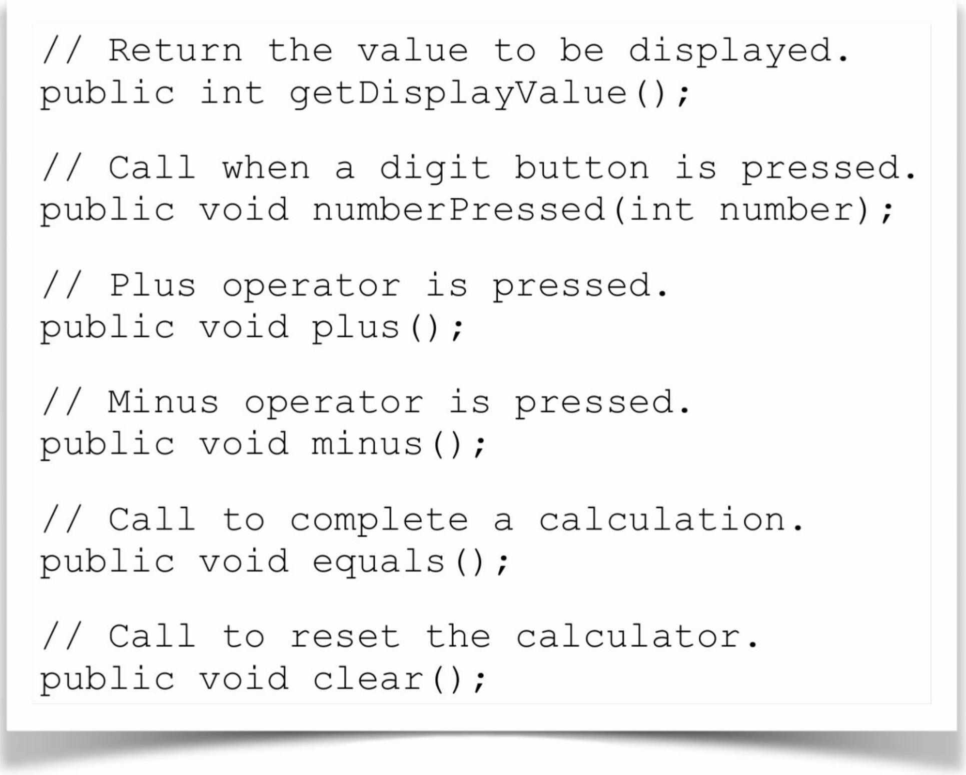
  - Increases the likelihood of successful integration.

Modularization in a calculator

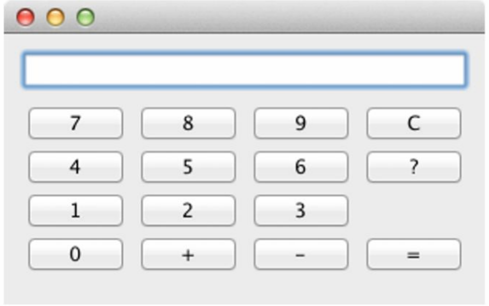


- Each module does not need to know implementation details of the other.
    - User controls could be a GUI or a hardware device.
    - Logic could be hardware or software.

Method headers as an interface



```
// Return the value to be displayed.
public int getDisplayValue();

// Call when a digit button is pressed.
public void numberPressed(int number);

// Plus operator is pressed.
public void plus();

// Minus operator is pressed.
public void minus();

// Call to complete a calculation.
public void equals();

// Call to reset the calculator.
public void clear();
```



Debugging - revisited

- It is important to develop code reading skills.
  - Debugging will often be performed on others’ code.

- Techniques and tools exist to support the debugging process.

- Explore through the calculator-engine project.

Manual Walkthroughs

- Relatively underused.
  - A low-tech approach.
  - More powerful than appreciated.

- Get away from the computer!

- ‘Run’ a program by hand.

- High-level (Step) or low-level (Step into) views.

Tabulating object state

- An object’s behavior is largely determined by its state.
- Incorrect behavior is often the result of incorrect state.
- Tabulate the values of key fields.
- Document state changes after each method call.

Verbal walkthroughs

- Explain to someone else what the code is doing.
  - They might spot the error.
  - The process of explaining might help you to spot it for yourself.

- Group-based processes exist for conducting formal walkthroughs or inspections.

Print statements

- The most popular technique.
- No special tools required.
- All programming languages support them.
- Only effective if the right methods are documented.
- Output may be voluminous!
- Turning off and on is labor intensive and error prone.

Demo

Let’s make a code review
of the
“Online-shop-junit”
project.