

Well-behaved objects

Chapter-07;
Objects First with Java using BlueJ (5th edition)

Code snippet of the day

```
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?

Results

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

Double result: 10
Double result: 12
Double result: 22
Double result: 66

Which ones are printed?

Results

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

Code snippet of the day

```
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 errors are usually *syntax errors*.
 - The compiler will spot these.
- Later errors are usually *logic errors*.
 - The compiler cannot help with these.
 - Also known as bugs.
- Some logical errors have no immediately obvious manifestation.
 - Commercial software is rarely error free.

Prevention vs Detection (Developer vs Maintainer)

- We can lessen the likelihood of errors.
 - Using software engineering techniques, like encapsulation \ information hiding.
- We can improve the chances of detection.
 - Using software engineering practices, like modularization and documentation.
- We can develop detection skills.

Testing and Debugging

- Crucial skills in software engineering.
- Testing is the activity of finding-out whether a piece of code produces the intended behavior.
- Debugging is the attempt to pin-point and fix the source of an error.
 - The manifestation of an error may well occur some ‘distance’ from its source.

Testing

- Testing means running the program using a set of test-cases, and examining the results.
- A test-case is a pair of (inputs, outputs) that are designed to verify program functionality.
- Designing effective test-cases is the art of good software-testing.

Testing Fundamentals

- How to design effective test-cases?
- Understand the “contract”:
what the program should, and shouldn’t do.
- Use positive tests and negative tests.
- Test *boundaries*.
 - Zero, One, Full.
 - Search an empty collection.
 - Add to a full collection.

Positive vs Negative Testing

- Positive testing is the testing of cases that are expected to succeed.
- Negative testing is the testing of cases that are expected to fail.
- A common pitfall for inexperienced testers is to conduct only positive tests! Negative tests are crucial for a good test procedure.

Assertions

- An assertion is an expression that states a condition that we expect to be true.
- If the condition is false, we say that the assertion fails.
 - This indicates an error in our program.
- Some programming languages (like Java) have special statements for assertion.

Assertions

```
int sum = 1;  
for (int i = 0; i <= 4; i++)  
    sum = sum + 1;  
assert (sum == 6);
```

- If the assert condition is true, then nothing happens.
- If the assert condition is false, an exception is raised:
 - `java.lang.AssertionError`
 - **NOTE:** must be run using: `java -ea` to “enable assertions”.

Unit Testing

- By far the most common testing strategy.
- Each unit of an application may be tested.
 - method, class, module (package in Java).
- Should be done during development.
 - Finding and fixing early, lowers development costs (e.g. programmer time).
 - A test suite is built up.

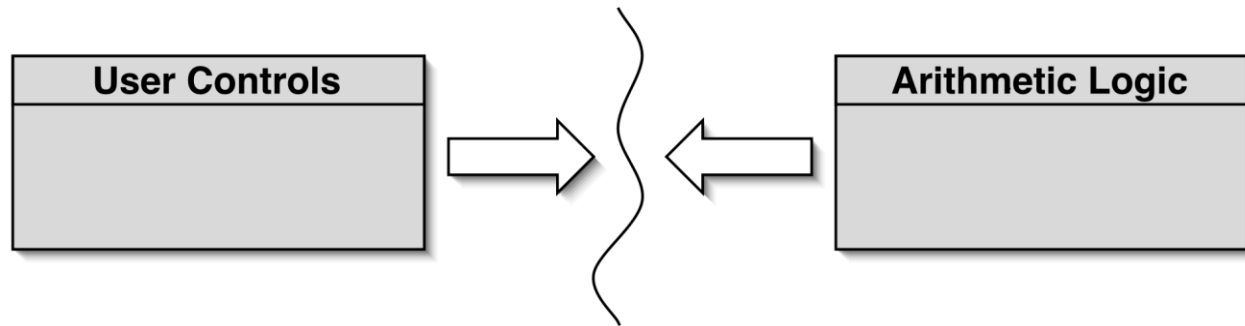
Test Automation

- Good testing is a creative process, but ... thorough testing is time consuming and repetitive.
- *Regression testing* involves re-running a set of tests.
- Use of a *test rig* or *test harness* can relieve some of the burden.
 - Classes are written to perform the testing.
 - Creativity focused in creating these.

Modularization & Interfaces

- Applications often consist of different modules.
 - E.g. so that different teams can work on them.
- The *interface* between modules must be clearly specified.
 - Supports independent concurrent development.
 - Increases the likelihood of successful integration.

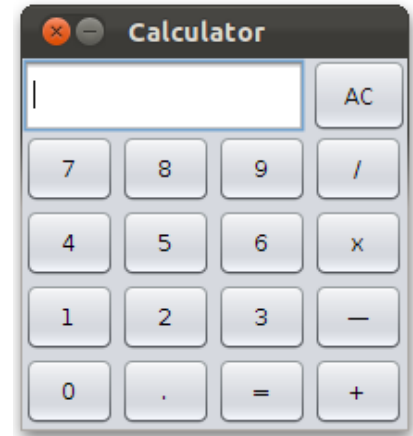
Modularization in a calculator



- 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 signatures as Interfaces

```
// Return the value to be displayed.  
public int getDisplayValue();  
  
// Call when a digit button is pressed.  
public void numberPressed(int number);  
  
// Call when a plus operator is pressed.  
public void plus();  
  
// Call when a minus operator is pressed.  
public void minus();  
  
// Call to complete a calculation.  
public void calculate();  
  
// Call to reset the calculator.  
public void clear();
```



Choosing a test strategy

- Be aware of the available strategies.
- Choose strategies appropriate to the point of development.
- Automate whenever possible.
 - Reduces tedium.
 - Reduces human error.
 - Makes (re)testing more likely.

Debugging Techniques

- Manual walkthroughs
- Print statements
- Debuggers

Debugging Skills

- 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.

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 usually determined by its state.
- Incorrect behavior is often the result of incorrect state.
- Tabulate the values of all 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 requires forethought.

Debuggers

- Debuggers are both language- and environment-specific.
 - BlueJ has an integrated debugger.
- Support breakpoints.
- Step and Step-into controlled execution.
- Call sequence (stack).
- Object state.

Review

- Errors are a fact of life in programs.
- Good software engineering techniques can reduce their occurrence.
- Testing and debugging skills are essential.
- Make testing a habit.
- Automate testing where possible.
- Practice a range of debugging skills.