Well-behaved objects

Main concepts to be covered

• Testing
• Debugging
• Test automation
• Writing for maintainability
Code snippet of the day

```java
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: 11
The result is: 2
Double result: 22
Double result: 22
Double result: 66
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.

---

```java
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);
}
```
Prevention vs Detection (Developer vs Maintainer)

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

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 some ‘distance’ from its source.
Testing and debugging techniques

- Unit testing (within BlueJ)
- Test automation
- Manual walkthroughs
- Print statements
- Debuggers

Unit testing

- Each unit of an application may be tested.
  - Method, class, module (package in Java).
- Can (should) be done during development.
  - Finding and fixing early lowers development costs (e.g. programmer time).
  - A test suite is built up.
Testing fundamentals

• Understand what the unit should do - its contract.
  - You will be looking for violations.
  - Use positive tests and negative tests.
• Test boundaries.
  - Zero, One, Full.
    • Search an empty collection.
    • Add to a full collection.

Well-behaved objects

Test automation
Main concepts to be covered

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

Unit testing within BlueJ

- Objects of individual classes can be created.
- Individual methods can be invoked.
- Inspectors provide an up-to-date view of an object’s state.
- Explore through the *diary-prototype* project.
Test 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.
  - Classes are written to perform the testing.
  - Creativity focused in creating these.

Test automation

• Explore through the diary-testing project.
  - Human analysis of the results still required.
• Explore fuller automation through the diary-test-junit projects.
  - Intervention only required if a failure is reported.
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

Well-behaved objects

Debugging
Prevention vs Detection
(Developer vs Maintainer)

• We can lessen the likelihood of errors.
  • Use software engineering techniques, like encapsulation.
• We can improve the chances of detection.
  • Use software engineering practices, like modularization and documentation.
• We can develop detection skills.

Debugging techniques

• Manual walkthroughs
• Print statements
• Debuggers
Modularization and 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 an interface

// 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 equals();

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

Debugging

• 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 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.
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)testng more likely.

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.

Acknowledgement

The original authors of these slides are the authors of the textbook. The instructor made necessary modifications, with permissions from the authors.