

Testing and Debugging

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);
}

The result is: 2
```

What is the output?

This semi-colon marks the end of an **empty statement**, which is the loop body.

This code block follows the loop. With the semi-colon above, it is **not the loop body**.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

(Reminder) Zuul Assignment

- Two deliverables:
 - **Code** (to your submission folder *by 23:59*)
 - **Report** (to your submission folder *or hard copy to the CAS office by 4pm*)
 - **Deadline** is Tuesday, December 8, 2009

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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);
}

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

What is the output?

This arithmetic is never done – the number is converted to a string and appended to the opening string (*twice*).

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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?
```

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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 **intermittent manifestation** with no obvious trigger:
 - also known as ******* bugs**
- Commercial software is rarely error free:
 - often, it is much worse than this ...

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Prevention vs Detection (Developer vs Maintainer)

- We can lessen the likelihood of errors:
 - encapsulate data in *fields* of an *object*
 - design each *class* to be *responsible* for *one* kind of thing
- We can improve the chances of detection:
 - write documentation *as we go*
 - think about how to test *as we go*
 - test *as we go* (e.g. interact with classes/objects/methods using the facilities of *BlueJ*, build a suite of testing classes – see *Unit Testing* and *Regression Testing* in Chapter 6)
- We can develop detection skills. ☺☺☺☺

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Testing Fundamentals

- Understand what the unit should do (*its contract*):
 - look for violations (*negative tests*)
 - look for fulfillment (*positive tests*)
- Test *boundaries*:
 - for example, search an empty collection
 - for example, add to a full collection

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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 (Chapter 6). *This contains bugs!!!*

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Unit Testing

- Each *unit* of an application may be tested:
 - *each method*
 - *each class*
 - *each package*
- Can (*should*) be done during development:
 - finding and fixing *early* lowers development costs (e.g. *programmer time*).
 - finding problems *after deployment* is very expensive. This is what usually happens.
 - if the system design *lacks cohesion* and has *high coupling* between units, the bugs may *never be found* and, if found, be *unfixable*.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Test Automation

- Good testing is a creative process.
- Thorough testing is time consuming and repetitive.
- *Regression* testing involves re-running tests *whenever* anything is changed.
- Use of a *test rig* can relieve some of the burden:
 - write classes to perform the testing
 - creativity focused in creating these
 - *BlueJ* offers help for this:
 - explore the *diary-testing* project (Chapter 6). Human checking of the results still needed here.
 - explore the *diary-test-junit* projects (Chapter 6). Machine checking of results – human intervention only if one or more checks fail.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Debugging Techniques

- OK, we found errors ... how do we fix them?
 - manual walkthroughs:
 - read your code!**
 - why should it work? [Not: why doesn't it work?!]*
 - it helps to explain this to someone else ...*
 - if light has not dawned:
 - add print statements to your code to show the state of variables (at key points)...**
 - do they show what you expect?*
 - if light has not still not dawned:
 - run your code under an interactive debugger ...**
 - BlueJ has support for this (see chapter 3.13)*

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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();
```



Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Hofstadter's Law

It always takes longer than you expect, even when you take Hofstadter's Law into account.

90% Rule of Project Schedules

The first ninety percent of the task takes ninety percent of the time ... and the last ten percent takes the other ninety percent.

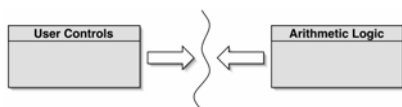
Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Modularisation 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.



Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Tabulating Object State

- An object's behaviour usually depends on its state.
- Incorrect behaviour is often the result of incorrect state.
- Tabulate the values of all fields.
- Record state changes after each method call.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Interactive Debuggers

- Debuggers are language-specific and environment-specific.
 - *BlueJ has an integrated debugger.*
- Supports *breakpoints*.
- Provides *step* and *step-into* controlled execution.
- Shows the *call sequence (stack)*.
- Shows *object state*.
- *Does not provide a permanent record.*

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Verbal Walkthroughs

- Explain to someone else what the code is doing:
 - *explain why should it work!*
 - *[don't ask: why doesn't it work?!]*
 - *the process of explaining helps you spot errors for yourself;*
 - *often, it's easier for someone else!*
- Group-based processes exist for conducting formal walkthroughs or *inspections*.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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.
- Practise a range of debugging skills.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

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.

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Review

- *Unit testing (with BlueJ)*
 - Chapter 6.3
- *Test automation (with BlueJ)*
 - Chapter 6.4
- *Manual walkthroughs (read the code!!!)*
 - Chapter 6.8
- *Print statements (see what's happening)*
 - Chapter 6.9
- *Debuggers (with BlueJ)*
 - Chapter 3.12-13, 6.11

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling