Fundamentals of Programming Using Java

Dr. Kevan Buckley

Introduction to the Module

Dr. Kevan Buckley

Fundamentals of Programming

- Programming
  - Preparation of a sequence of instructions (a program) that details exactly how a computer must proceed to solve a problem.
  - Computers need instructions on how to solve problems
  - A complete set of instructions is a program
  - Programming in writing a program

- Algorithms
- Data
- Software lifecycle
- Discrete mathematics
- Object orientation

Programming is an Art

- “The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music.”
  — Donald Knuth, 1997

- We cannot teach you how to program
  - You have to learn through experience
  - Problem solving cannot be learnt from a book
  - You have to acquire the skill
  - Practice, practice, practice

Principles of Software Development

- Algorithm design and construction
  - (problem solving)
  - sequence, selection, iteration
- Underpinning mathematics
  - logic, number systems, matrices
- Programming pragmatics
  - style, testing, debugging, refactoring
- Software lifecycle
- Fundamentals of object orientation
  - objects, classes, attributes, methods, encapsulation
- The module uses Java

Dr. K. A. Buckley, September 2005
### Subject Specific Learning Outcomes

- **(a)** Subject knowledge and understanding of
  - (i) The fundamentals of programming, algorithms, object orientation and underpinning mathematics.

- **(b)** Subject specific skills – able to
  - (ii) Demonstrate the practice and applications of computer science.
  - (iii) Develop, test, debug and document software using an introductory programming IDE.

- **(c)** Intellectual skills – able to
  - (iv) Apply an abstract body of knowledge to solve tasks of a real world nature.

### Key Skills

- **Problem Solving**
  - Design algorithms to solve small problems

### Module Sessions

- There are 4 separate runs of the module
  - Wednesday 9:00 – 12:00 MI034
  - Wednesday 9:00 – 12:00 MI035
  - Thursday 14:00 – 17:00 MI035
  - Thursday 14:00 – 17:00 MI034

**You must attend the session specified by your Personal Tutor or Award Leader. There may have been a change – see your email.**

- **Student contact hours** 3 hours
  - 2 x 1 hour lectures
  - 1 hour workshop
- **Student self directed (at least)** 7 hours
  - Reading
  - Programming exercises
  - Making notes
  - Online activities

### Surgeries

- **Extra help sessions**
- **You should attend iff:**
  - You think you are falling behind
  - You a specific query that you want to resolve before the next formal session
  - You have been advised to do so

**These sessions are purely optional - only attend if you need to.**

- **Wednesday 14:00-16:00 room to be announce by Wolf.**

### Self Study

- **Workshops**
  - Exercises and resources can be downloaded from Wolf.
  - Work not completed in class should be attempted as self-study work.
- **Guided reading**
  - You will be instructed on what to read each week. Only a bare minimum will be specified and you should read around the subject.
- **Formative assessments**
  - Help gauge your progress
  - Self test quizzes and online tasks will be issued weekly.
  - Formative mini tests in most lectures.

### Assessment

- **Element 1**
  - In-class programming test
  - 50% of module grade
  - Teaching week 8

- **Element 2**
  - 1.5 hour time constrained assessment
  - 50% of module grade
  - Teaching week 14

You must achieve at least E4 in both tests.
You must achieve D5 overall to pass the module.
### Weekly Program

<table>
<thead>
<tr>
<th>Week</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction to computers, programming, Java and software development. Introduction to an integrated development environment. Experience running simple programs and conducting observations of making minor modifications to them.</td>
</tr>
<tr>
<td>2</td>
<td>Algorithms and programs for making calculations and producing output. Problem solving and algorithm development. Practical implementation of calculation and output algorithms. Analysing programs with using Dry Runs.</td>
</tr>
<tr>
<td>3</td>
<td>Input and strings. Practical implementation of input methods. Logic, truth tables and Boolean expressions.</td>
</tr>
<tr>
<td>5</td>
<td>Reading week</td>
</tr>
<tr>
<td>7</td>
<td>Nested iteration. Analysis of and practical implementation of programs that use nested iteration. Testing and test data. Preparation for assessment.</td>
</tr>
</tbody>
</table>

### Learning Materials

- **Text book**
- **Required software**
  - Bluej - http://www.bluej.org
  - Java Development Kit - JDK 1.5 - http://java.sun.com/j2se/1.5.0/
- **Lecture notes**
  - Only provided for lectures not covered in the course text.
  - Other notes are on Wolf
- **Wolf**
  - http://wolf-nt.wlv.ac.uk
  - Electronic copies of lecture slides, workshop exercises, tutorials etc.
- **Safari**
  - Electronic books
  - http://proquest.safaribooksonline.com/?uicode=wolvham
- **Wiki**
  - Details available next week

### The Charter for Students

- **Attend regularly and punctually.**
  - All sessions start on the hour
  - If late, sit as close to the entry point and cause as little disturbance as possible
- **Transfer time & breaks given at end of sessions**
- **Show courtesy and respect to staff and other students**
  - If you talk during lectures, you will affect other students' concentration – if you are bored don't put others off
- **Respect and abide by the University Regulations, e.g. Equal Opportunities Policy, ID Cards, quiet areas**
  - ID cards to be shown at tests.
- **Bring all necessary equipment to classes/workshops**
  - Something to write with; Something to write on.
  - Resolve login problems and other issues prior to workshops.

### Finally

**The Ground Rules**

- **Switch off mobile phones when in class**
  - Put them out of sight
  - No texting
  - If you leave the room, don't attempt to come back
- **Behave considerately in lectures, listen attentively and participate in class activities**
  - Give the tutor full attention during lectures.
- **Use the advertised times (or make an appointment by email) to seek advice from your tutors/lecturers.**
  - If you don't attend normal sessions, don't expect help outside them.
Exercise - Using a Command Prompt

- Working in a group of 4 from around your table
  - Huddle around one computer
- Subscribe to CP1068 on Wolf
  - Find Week 1
  - Find Workshop 1
  - Work though Part 1
- If you get stuck put up your hand and indicate to a tutor that you need help.
- Make notes on the commands you have learnt.
  - You will need to use them when studying on your own later.
- You have got 30 minutes before we review what you have done.

Lecture Outcomes

- To understand the context of Computer Science Programming
  - High level and low level languages
  - The software development process
    - Authoring, compilation and interpretation
- Overview of Java
  - History, motivation and capabilities.

What is a Computer?

- Computer
  - Electromechanical device.
  - Performs computations and makes logical decisions millions / billions times faster than human beings.
- Hardware
  - Physical devices of computer system.
  - The parts of the computer you can touch
- Software
  - Programs that run on computers
  - Sets of instructions that are followed by the computer to processes data.
  - Instructions are written in a computer language.

Computer Components

- Six logical units of computer system - Hardware
  - Input unit
    - Mouse, keyboard
  - Output unit
    - Printer, monitor, audio speakers
  - Memory unit
    - RAM
  - Arithmetic and logic unit (ALU)
    - Performs calculations
  - Central processing unit (CPU)
    - Supervises operation of other devices
  - Secondary storage unit
    - Hard drives, floppy drives

Layers of a Computer System

- User
  - Applications programs
- Operating System
  - Hardware
- Examples:
  - Operating system:
    - Application program:
Most programs consist of the following steps:

1. Input some data
2. Process the data
3. Output the result

This could be considered a generic design/solution to all problems!

Tom requires a program that will calculate and display a bill.
- The weight of vegetables and the cost per kilo are input and are multiplied together to form the output.

The Program

<table>
<thead>
<tr>
<th>Input</th>
<th>Process</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>costPerKilo</td>
<td>priceToPay = costPerKilo X weightOfVeg</td>
<td>priceToPay = 2.17</td>
</tr>
<tr>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weightOfVeg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Do the Following Match the I-P-O Model?

- What are the inputs?
- What are the outputs?
- What processing needs to be done?

- Scientific calculator
- Supermarket checkout
- Arcade-style computer game (e.g. GTA)

Programming Exercise

- Work through Part 2 of Workshop 1.
- If you get stuck put up your hand and indicate to a tutor that you need help
- You have got 10 minutes before we review what you have done.

Programming

- Preparation of a sequence of instructions (a program) that details exactly how a computer must proceed to solve a problem.
  - i.e. provide a plan of how to transform inputs into outputs
- Programs are defined and passed to a computer using a language.
- There are different levels of language

<table>
<thead>
<tr>
<th>Low Level</th>
<th>High Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer's understanding</td>
<td>Easy</td>
</tr>
<tr>
<td>Human's understanding</td>
<td>Difficult</td>
</tr>
</tbody>
</table>
Levels of Language

- **Machine language**
  - “Natural language” of computer component
  - Machine dependent
- **Assembly language**
  - English-like abbreviations represent computer operations
  - Translator programs convert to machine language
- **High-level languages**
  - Allows for writing more “English-like” instructions
    - Contains commonly used mathematical operations
  - Compiler converts to machine language or an interpreter is used.

Machine Language

- This lowest level of programming language
- Represents data and program instructions as binary digits (0s and 1s)
  - Corresponds to the on and off electrical states in the computer.
  - The only language the computer truly understands.
  - Each type of computer has its own machine language.
  - Not convenient for people to read and use.
- All other languages must be translated to the machine language before execution.
- The computer industry quickly moved to develop assembly languages.

Assembly Languages

- Assembly languages are considered very low level
  - They are not as convenient for people to use as more recent languages.
  - At the time they were developed they were considered a great leap forward.
- Assembly languages use mnemonic codes, abbreviations that are easy to remember than m/c
  - ADD for addition, LD for loading data into memory, JP for jumping to another part of the program etc.
  - Assembly languages permit the use of names for memory locations instead of actual address numbers.
  - Instructions have a one-to-one correspondence with machine code.
  - As with machine language, each type of computer has its own assembly language.

High-Level Languages

- Programs are written in an English-like manner, thus making them more convenient to use.
  - First widespread use of high-level languages in the early 1960s.
  - Transformed the way programming was done.
  - A programmer could accomplish more with less effort.
  - Programs can perform much more complex tasks.
- Java, C++, BASIC are all high level languages.
- Programs written in high level language are easier for people to understand
  - Need to be translated to machine code for a computer to understand.
  - Translation done by a compiler or an interpreter.

Traditional Models of Execution

- **High level languages understood by programmer**
  - Java, C++, Visual Basic
- **Low level languages understood by computer**
  - Machine code

1. Authoring/Editing
2. Compiling
   - Converts high level to low level
3. Run
   - Low level program executed by computer

Java is Compiled and Interpreted

Programmer types the program into the computer in the high level Java language.

1. The Java program is translated into a form that is “understood” by the computer.
2. The program executes
3. Hardware and Operating System

Text Editor
- Source Code
- Created with notepad, emacs, vi, bluej etc.

Compiler
- Byte Code
- The javac command

Interpreter
- Interprets and runs the program
- The java, appletviewer, netscape etc.
Programming Exercise

- Work through Part 3 of Workshop 1.
  - Try to see how the steps you take relate to the previous diagram.
  - If you need to look back at the diagram you should be able to find it on Wolf.
- If you get stuck put up your hand and indicate to a tutor that you need help
- You have got 5 minutes before we review what you have done.

Java is Compiled and Interpreted

Java – Applications & Applets

- Application
  - Standalone program resides and is run on local machine
  - Text based (console) I/O by default
  - Lax security model (but operates in JVM)
- Applet
  - Resides on remote server and is downloaded when needed
  - Runs inside another application in embedded JVM
  - Uses processor of local machine, not remote server
  - Graphical
  - Strict security model

Exercise

- Go to the Workshop 1 resources page and click on the link for the applet demo.
- You have got 1 minute before we review what you have done.

Errors

- Syntax errors
  - Bad punctuation, spelling etc
  - Java is case sensitive
  - Program will not compile
- Logical errors
  - Program compiles and runs, but does not give correct results
  - Caused by poor design
- Runtime errors
  - Program stops working (crashes, falls over, blue screen etc)
- Testing is very important
  - We will look at this in depth later in the course
- Writing a program is an iterative process.

Syntax Errors

- Compiler error.
- Violation of the rules of whatever language a computer program is written in.
- A statement that violates one or more rules of a programming language.
- Is a mistake in a software code's grammar.
- An error in a program caused by using a statement or other part of the language incorrectly. Easier to find than logical errors.
Syntax Errors

• An error detected by the compiler during its parsing of a program. Syntax errors usually result from mis-ordering symbols within expressions and statements. Missing curly brackets and semicolons are common examples of syntax errors.
• An error in a program that makes it impossible to parse (and therefore impossible to compile).
• An error of language resulting from code that does not conform to the syntax of the programming language; "syntax errors can be recognized at compilation time"; "a common syntax error is to omit a parenthesis"

Logic Errors

• A programming error in which the syntax of the statements is correct, but the algorithm is incorrect, or has been incorrectly coded.
• Causes a program to do something it isn't supposed to do but doesn't prevent the program from running. Hard to find.
• Is a mistake in the way an algorithm solves a problem.
• Error that occurs when a programmer has used an incorrect calculation or left out a programming procedure.

Runtime Errors

• An error that occurs during the execution of a program. In contrast, compile-time errors occur while a program is being compiled. Runtime errors indicate bugs in the program or problems that the designers had anticipated but could do nothing about. For example, running out of memory will often cause a runtime error.
• An error that causes a program to terminate when it is being run.
• An error in logic or arithmetic that must be detected at run time

Exercise for Self Study

• In your independent study time experiment with creating errors in Program1.java and note the results.
• Instructions are in Part 5 of Workshop 1.

The Programming Process

• There are 6 main elements in the programming process
  – Analysing/defining/modelling/specifying the problem
  – Planning/designing the solution
  – Coding/implementing the program
  – Testing the program
  – Documenting all aspects of the process
  – Maintenance
• This is often referred to as the software lifecycle.

Documenting The Process

• UML – Unified Modelling Language
  – Graphical representation scheme
  – Enables developers to model object-oriented systems
  – Flexible and extendible
• History of the UML
  – Brainchild of Booch, Rumbaugh and Jacobson
    • All had their own methodologies
  – UML unified the existing methods
    • Object Management Group (OMG) supervised
Structured Programming

- The Old programming strategy.
- Languages such as C, Pascal and Basic.
- Structured programs
  - Data structures carefully designed
  - Algorithms/methods to operate on the data designed
  - Data and programs considered to be separate entities.
  - Clearer than unstructured programs
  - Easier to test, debug and modify
- Does not scale well to very large problems.

Object-Oriented Programming

- Objects
  - Reusable software components that model real-world items
  - Look all around you
    - People, animals, plants, cars, etc.
  - Attributes
    - Size, shape, color, weight, etc.
  - Behaviors
    - Babies cry, crawl, sleep, etc.
- Object-oriented design
  - Models real-world objects
  - Models communication among objects
  - Encapsulates data (attributes) and functions (behaviors)
    - Information hiding
    - Communication through well-defined interfaces
- Scales well to complex problems

Personal Study - At Least 7 Hours

- You should have already read Chapter 1 of Currie
  - Covers material related to this session
  - Try the programming exercises
  - Answer the review questions
  - Use books and the Internet to find better explanations of things that Currie does not explain well
  - Bring you problems to the session next week
- Do any exercises from this session that you did not complete or need to go over again.
  - If you did them in a group and you did them on somebody else’s computer repeat the exercises to ensure you fully understood them.
- Do the Quiz Sheet on Wolf
  - There will be a short formative test in next week’s tutorial
- Contribute to the Wiki
  - If you think the definitions found on our Wiki could be better, please contribute to making them better.
- Read Currie Chapters 2 and 3 in preparation
  - Make a list of things you don’t understand and bring them to the next session.

Java History

- C++
  - Evolved from C
    - Evolved from BCPL and B
  - Provides object-oriented programming capabilities
  - Objects – reusable software components that model real-world items
- Java
  - Based mainly on C++, but uses concepts from other languages such as SmallTalk and Eiffel.
  - Originally for intelligent consumer-electronic devices
  - Then used for creating Web pages with dynamic content
- Now also used for:
  - Develop large-scale enterprise applications
  - Enhance WWW server functionality
  - Provide applications for consumer devices (cell phones, etc.)

Java – Simple, OO & Familiar

- Simple
  - Can easily be programmed by most developers
- Object Oriented
  - Take advantage of modern design methodologies
  - Matches the needs of client-server applications
  - Encapsulated, message passing
  - Existing class libraries can be extended to provide new functionality
- Familiar
  - Current developers can easily learn the language
  - Quick migration improves productivity
## Java – Robust and Secure

- Designed for creating reliable software
  - Compile time and runtime checking
- Language features guide developers towards reliable programming habits
  - Forces the use of exception handlers
  - Variables must be initialised before use
- Simple memory management model
  - Makes programs robust
  - Automatic garbage collection (explicit deletion not required)
  - Array bounds checking
- Security built into language and runtime environment
  - Execution-time checking of code
  - Restricted operating environment
  - End users can trust applications are secure

## Java – Architecture Neutral & Portable

- Design driven by nature of computing environments
  - Deployment required into heterogeneous networks
- Applications must be capable of executing on a variety of hardware and operating environments
  - Uses bytecode, a neutral format for transporting code
  - Programs are the same on every platform
  - No data incompatibilities
  - Write once – run anywhere
- Requires virtual machine on target machines
  - Based on the POSIX interface specification
  - Implementation is straightforward as long as platform meets basic requirements

## Java - Interpreted, Threaded & Dynamic

- Multithreading is built into the core of the runtime environment
  - Concurrent programming easy to implement
  - Monitor and condition lock primitives provided
  - Allows high performance GUI’s
- Interpreter alleviates the need for explicit linking
  - Enhanced development cycle
- Dynamic loading
  - Just-in-time linking
  - Applications are adaptable as code can be loaded dynamically over the network

## Design Patterns

- Effective design crucial for large programs
- Design patterns
  - Proven architectures for developing object-oriented software
    - Architectures created from accumulated industry experience
    - Similar to architectural elements
      - Arches and columns
  - Promotes design reuse in future systems
  - Reduce design-process complexity
  - Helps identify common design mistakes and pitfalls
  - Helps design independently of implementation language
  - Establishes common design “vocabulary”
  - Shortens design phase in software-development process