

Object Oriented Program Design

(COMP1001)

Module 1



Copyright Warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulation 1969

WARNING

This material has been copied and communicated to you by or on behalf of Curtin University of Technology pursuant to Part VB of the Copyright Act 1968 (the Act)

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice



This Module

– Topics:

- Basics of a Computer
- Using the Unix (Linux) Operating System
- Introduction to Java
- Java virtual machine & byte code
- Program components
- Format of program
- Variables & constants
- Data types
- Expressions & how values change in memory



What is a computer?

- Machine that accepts data, processes it and produces output
- Computer consists of:
 - CPU
 - Input/output devices
 - Dynamic memory
 - Secondary storage
- So it can run software:
 - Systems software
 - » Operating systems, network tools, software development tools (e.g. compilers).
 - Application software:
 - » Word processors, spreadsheets, databases, modelling and simulation.



Data Representation in Computer Memory

- Computer memory is made up of components that can be in one of two states i.e. on or off.
- Other binary methods of information include:
 - on and off (flashing light)
 - dot and dash (Morse)
 - yes and no (Ouija board – pronounced weeja)
 - north and south (magnet)
- Can be used in two ways:
 - Represent actual values in base 2
 - Hold an arbitrary series of states coded with particular meanings



Terminology

- Each binary digit is called a bit
- A group of eight (8) bits is a byte
- Memory is broken up into storage locations of a particular wordsize
- Wordsize is machine dependent and will be one or more bytes long
 - now usually 32 or 64
- Each memory location is located through its memory address
- All data and programs are stored in memory using various interpretations of these groups of 1's and 0's



Data types

- Manner of interpretation of the 1's and 0's varies for different data types stored
 - Addresses
 - Instructions
 - Integer values
 - Real values
 - Characters:
 - » Single characters
 - » Character strings.
 - Boolean



Software

- A program is a set of instructions to a computer.
- These instructions are written in a programming language.
- Each computer understands one language, its machine language, which is not (usually) human-readable.
- A program is written in a high level language (for humans) & then translated into machine code (for the computer).
- The purpose of you studying this unit is to design and write well structured, robust, maintainable software - not just to write a program that works some of the time!



Programming Languages

- 1940's Machine language
- Early '50s Assembly language
- Late '50s: High level languages & compilers introduced
 - Fortran
 - COBOL
- 70's & 80's Emergence of better structured languages
 - Pascal
 - Ada
 - C
- 90's Object Orientated Languages
 - C++ (or was it???)
 - Java
 - Perl
- 2000's
 - C#
 - Python
 - Ruby



To be OO or not to be OO?

- There are two basic paradigms for designing algorithms:
 - Non Object Oriented:
 - » Focus is on the steps required to perform the task.
 - » The design of the steps lead to the types of data structures that will be required.
 - Object Oriented:
 - » Focus is on the entities required. i.e. what are the things that need to be represented in the algorithm and:
 - » what functionality will each thing require.
 - » how these things will communicate with each other.
 - » Each entity will be represented as an object.
 - » The design of each object leads to the steps required.
- In this course we will be looking at the Object Oriented approach.
- The non Object Oriented approach will be discussed in UCP.



Software Development

- Problem Definition

Think/consult/revise

- High Level Design (typically UML)

Specify

- Algorithm Design

Convert

- High level programming language implementation

Translation

- Conversion to machine instructions

Test

- Execute and test that it actually does the required job



Problem Definition

- Must define and understand the problem requiring a solution
- Start with a "top level" general English description of the problem
- State what data is required to solve the problem
 - the input data
- State what results will be calculated
 - the output data
- What actions need to be taken to generate the results
- Crucial part of solution is to know what the problem is (!) BUT often ignored by poor software developers
- Need to consider security, readability and performance requirements



Algorithm

- An algorithm is a set of detailed, unambiguous, ordered steps specifying a solution to a problem
 - Steps must be stated precisely, without ambiguity
 - Enter at the start & exit at the bottom
 - English description independent of any programming language
 - Non-trivial problem will need several stages of refinement
 - Various methodologies available
 - Must be desk-checked for correctness



Pseudo Code

- Algorithms are expressed in Pseudo Code:
 - English phrases which describe the algorithm steps.
 - The pseudo code is evolved from a rough description to something which almost looks like a programming language.
 - Pseudo code development is about REFINEMENT.
 - » Developing an algorithm is a journey where the problem statement is the starting point and the final draft of pseudo code is the destination.
- Algorithm design is an art that takes LOTS of practice.
- You will need to read chapter 3 of OOPD110Book.pdf many times!



Simple Example

– Problem

- Write a program to calculate the sum of 2 numbers input from the keyboard (user) and output to the screen (user)

– Algorithm

MAIN

INPUT numOne

INPUT numTwo

sum = numOne + numTwo

OUTPUT sum



Conversion

- Convert algorithm description into implementation
HLL e.g. Java
 - known as coding
 - Programmer needs to know the semantics and the syntax of language
 - The files of High Level Language (HLL) statements are called the source files



Translation

- Process by which the “Language” translates source code into machine code
- Errors found are called *syntax* or *compile-time* errors
- Two variants
 - Languages that are compiled
 - » Translation is done before execution
 - » C, C++
 - Languages that are interpreted
 - » Translation is done during execution
 - » Ruby, Python
 - Discussed later



Java Implementation

```
/*
 *Author:
 *Date:
 *Purpose:
 */

import java.util.*;
public class MyFirstJavaApplication
{
    public static void main( String [] args)
    {
        int numOne, numTwo, sum;
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter 1st number");
        numOne = sc.nextInt();
        System.out.println("Enter 2nd number");
        numTwo = sc.nextInt();

        sum = numOne + numTwo;

        System.out.println("Sum is " + sum);
    }
}
```



Program Execution

- If no compile errors machine language file can be executed
- Errors during program's execution are referred to as run-time or execution errors
- Errors in a computer program are called bugs and the process of eliminating them as debugging.
- They are not really bugs they are MISTAKES!
 - These mistakes are found and eliminated by thorough testing
- When program is ready, it is "put into production" and will be amended and expanded over time.

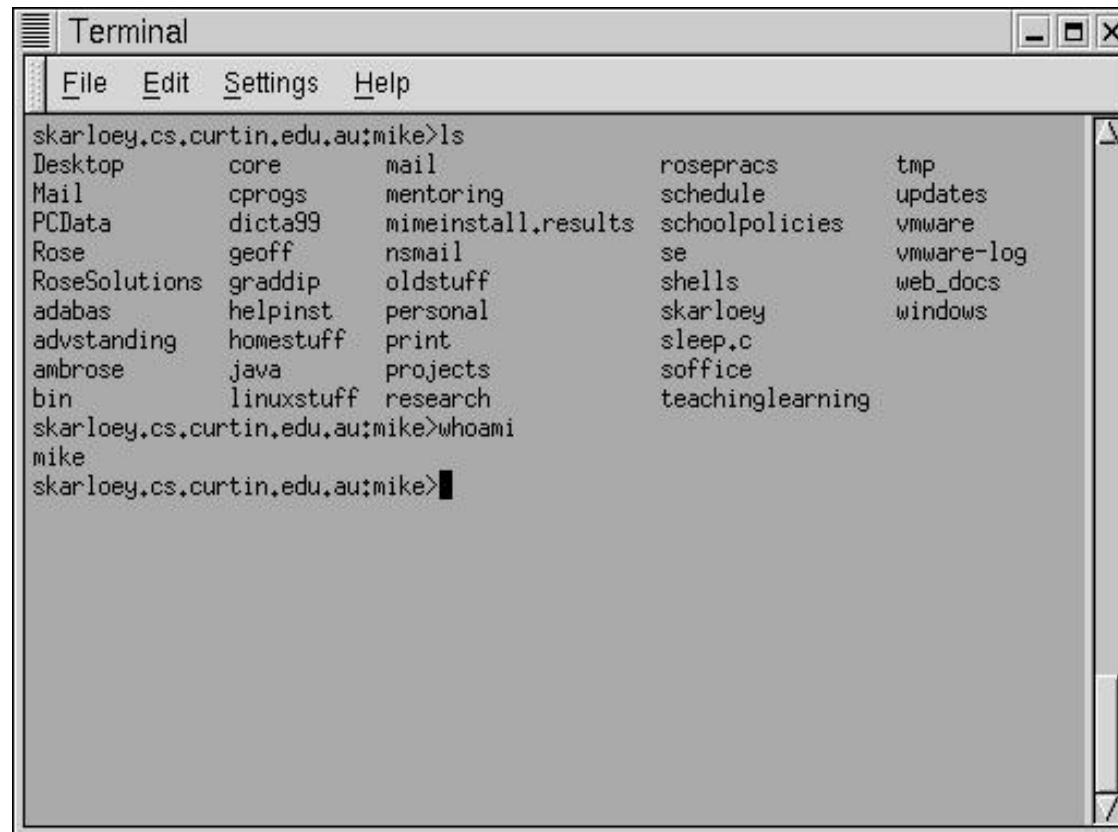


Introduction to Unix

- The operating system we will be using is called Linux.
 - Linux is a variant of Unix
- Unix is:
 - Totally different to what you might be used to.
 - An operating system which provides far greater scope in what can be accomplished but at a cost of a more difficult to learn user interface.
- As future computing professionals it is unacceptable for you not to be familiar with a Unix environment.

Command Line Interfaces

- A command line interface is composed of:
 - A prompt which signals the user when they can type commands.
 - A command line on which the typed commands appear.



The image shows a screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "Settings", and "Help". The terminal content shows a user prompt "skarloey.cs.curtin.edu.au:mike>" followed by a command "ls". The output of the command is a multi-column list of files and directories: Desktop, Mail, PCData, Rose, RoseSolutions, adabas, advstanding, ambrose, bin, skarloey.cs.curtin.edu.au:mike>whoami, mike, and skarloey.cs.curtin.edu.au:mike>. The prompt "skarloey.cs.curtin.edu.au:mike>" is repeated at the end of the output.

```
skarloey.cs.curtin.edu.au:mike>ls
Desktop      core         mail         rosepracs    tmp
Mail         cprogs      mentoring    schedule     updates
PCData      dicta99     mimeinstall.results schoolpolicies vmware
Rose        geoff       nsmail       se           vmware-log
RoseSolutions graddip     oldstuff     shells       web_docs
adabas      helpinst    personal     skarloey     windows
advstanding homestuff   print        sleep,c
ambrose     java        projects     soffice
bin         linuxstuff  research     teachinglearning
skarloey.cs.curtin.edu.au:mike>whoami
mike
skarloey.cs.curtin.edu.au:mike>
```



Introduction to Java

- Java is an Object Oriented (OO) language with its roots in C & C++
 - C is an imperative language, strongly associated with Unix operating system, designed in the early 70s
 - C's flexibility, which led to its popularity, also made it dangerous
 - C++ is basically C with OO features added which made it a "large" language
 - Quote from Stroustrup: "If you can shoot yourself in the foot using C then you will blow your whole leg off with C++!"
- In 1990-91 James Gosling designed Java as a "small" OO language
 - Initially aimed at information appliances but in 1993 adapted for animation & interaction on WWW
 - Introduction of 1995 Netscape allowing Java applets led to its current popularity.
- The emergence of Java and C# could mean the demise of C++ and its variants (e.g. visual c++).



Compiling

– Compilation:

- Process whereby the file of source code is translated into machine code.
- The source code is checked to ensure it conforms with grammar (syntax and semantics) rules.
- If no syntax errors found at compile time then machine code file is (eventually) created and can be executed.
- If even just one error then machine code file does not exist and there is no machine code to run.



Interpreting

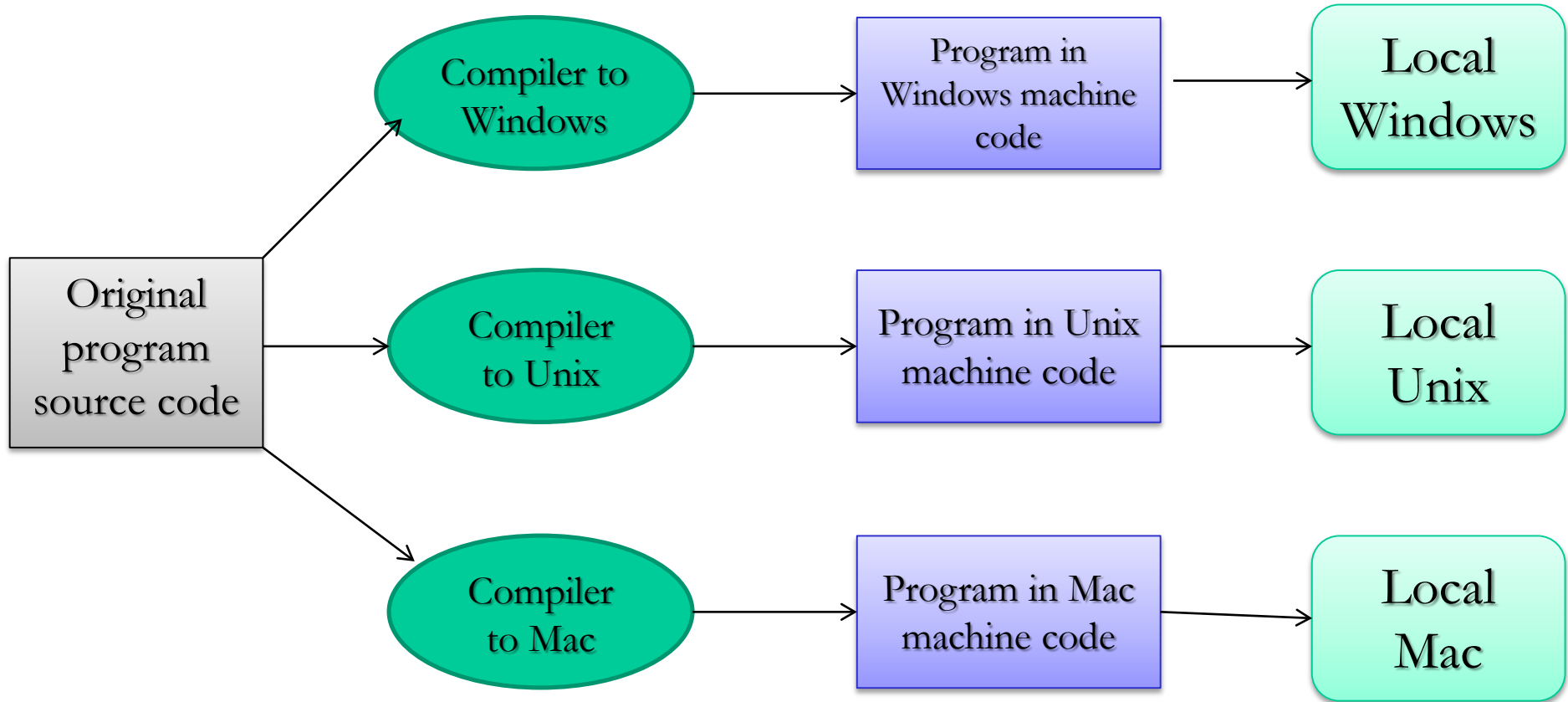
- Process whereby the file of source code is translated a line at a time into machine code instructions that can be executed by the machine.
 - If syntax errors exist then the program will be partially executed:
 - As soon as a syntax error encountered execution halts; note that with compilation syntax errors are eliminated prior to execution starting.
- Python and Ruby are both interpreted HLL .
- Interpreting code is slower than executing compiled code because syntax checking & translation has already been done with compiled code.
- If syntax errors can be eliminated before the source code is interpreted then the syntax error problem is avoided.



Java Platform Independence

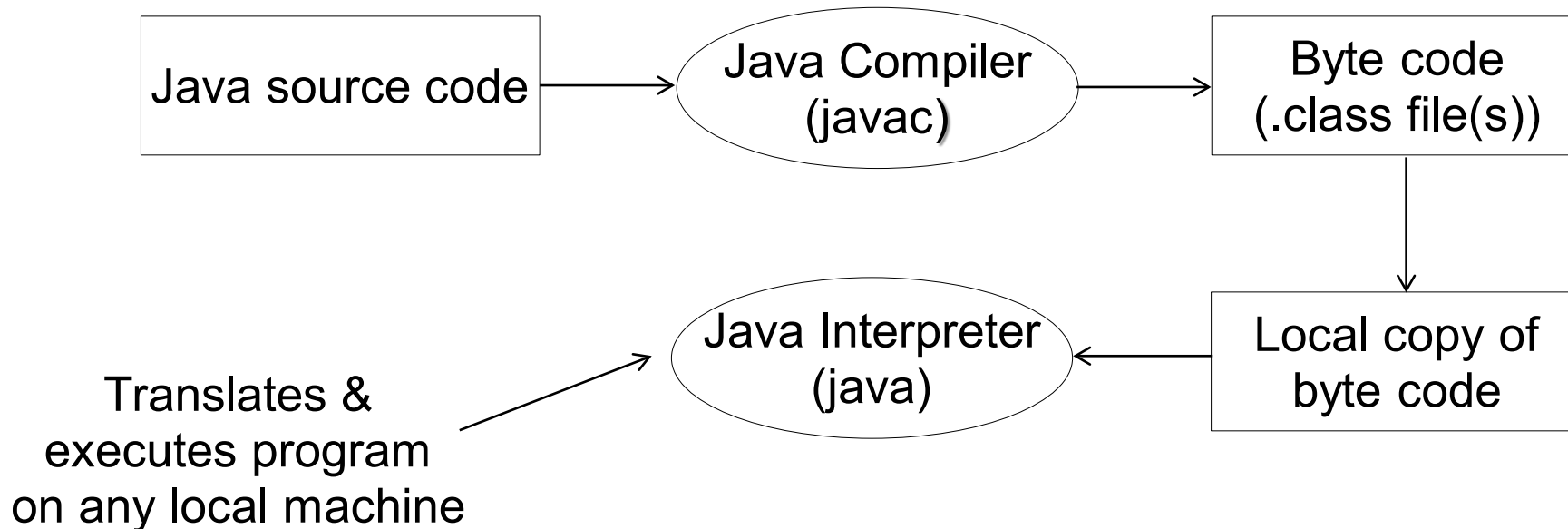
- Platform independence is achieved by running byte code on a Java virtual machine (jvm).
 - Byte code is a machine code for the jvm. A Java compiler compiles from source code to byte code.
- The jvm is itself a program whose job it is to interpret the Java byte code.
- Each machine actually needs code in its own particular native machine language: thus byte code needs to be interpreted to the local machine code to be able to execute locally.
- The overheads of interpreting byte code are lower than traditional interpreters because the conversion is from machine code (jvm) to machine code (native machine code).
 - Syntax checking has already been done at compile time.

Traditional Methods: Compile->Execute



Java Virtual Machine

- As long as local machine has the byte code interpreter it can download any Java program in byte code & execute it and yet:
 - Source code is secure.
 - Only one version of the byte code needs to exist





How to create a Java program

- Design and write the algorithm first!!!
- A text file with a file extension of .java must be created.
 - In this file (or files) you store the human readable Java program.
- The Java compiler (known as javac) is then used to compile the source code into byte code.
 - For each class (see later) defined in the source code the compiler will create a corresponding file of byte code.
 - The name of each byte code file will be the name of the class (as defined in the source code) with an extension of .class.
- The Java interpreter (known as java) is then used to translate the byte code and execute the resulting native machine code.



Applications and Applets

- A Java application is a program which is executed by directly invoking the Java interpreter.
 - Must contain a `main()` method
- Java servlets and applets are Java programs which are executed from within a web browser.
- OOPD will deal only with Java Applications.
- Graphical user interface issues will be dealt with in great detail in Computer Graphics and Human Computer Interfaces



Pseudocode

MAIN

message = "Welcome to Java"

OUTPUT message



A Simple Java Application

```
import java.util.*;
class MyFirstProgram
{
    public static void main( String [] args )
    {
        String message;

        message = new String("Welcome to Java!");
        System.out.println( message);
    }
}
```



Creating, Compiling and Running

- The Java code on the previous slide is entered into a text file using a text editor (under Linux we would use vim or gedit).
- The name of the .java file MUST be EXACTLY the same as the name of the class (i.e. `MyFirstProgram.java`)
- The .java file is then compiled into byte code:
- The command would be: `javac MyFirstProgram.java`
 - If the program contained errors then error messages are displayed and the byte code is NOT generated.
 - Otherwise the byte code is produced.
 - The byte code is stored in a file called `MyFirstProgram.class`
- The program can then be executed using the command:
`java MyFirstProgram`



The import statement

- Java comes with an extensive library of classes which can make the task of implementing algorithms much easier.
- It is very common for organisations to develop their own class libraries.
- In OOPD we do not have any class libraries, however you need to understand the import statement.
- The import statement tells the Java compiler that libraries are to be found by looking for the directory path specified.

```
import java.util.*;
```

- means all of the class files in the `util` directory.



The CLASSPATH Environment Variable

- The question is where to look for the top level Java directory?
- The CLASSPATH environment variable contains the directories that should be searched.
- The Java class libraries are stored in a default area when Java is installed. Under Linux it will be somewhere like:

`/usr/local/java/jdk1.6.1/lib`

- The CLASSPATH environment variable does not have to be set if the only place your Java program needs to look is the standard one.
- If you have libraries, you must specify where they are:
- Previously our CLASSPATH Variable was set as below:

```
"$CLASSPATH:/usr/java/latest/lib:/usr/units/st151/classes:."
```

 - Note that ST151 is the old name for OOPD.
- Each path is delimited by a colon. The first tells java where the classes directory is located. The second specifies the standard location and the period says look in the current directory as well.



Setting the CLASSPATH Variable

- Under Unix to set the CLASSPATH variable we use the export command.

```
export CLASSPATH="$CLASSPATH:/usr/java/latest/lib:/usr/units/st151/classes:."
```

- If we do this interactively we will have to type this command every time we login.
 - A better way is to add this line to a special file in our home directory called the .bashrc
-
- Under microsoft O/S:
 - Set an environment variable via GUI:

```
set CLASSPATH=C:\jdk1.6\lib;C:\myjava\st151;.
```
 - Where myjava is the folder where you have placed other non standard libraries.
 - By going to the control panel on windows you can set the CLASSPATH variable so that it will always be in effect. Please see the appropriate howto on blackboard for instructions on that.



Java Classes and Objects

- A Java application consists of a series of classes.
 - Everything in Java is encapsulated within a class.
- Within a class will be a number of:
 - Methods
 - Variables
 - » A method is a set of Java instructions (more later).
 - » A variable is a piece of memory used to store data (more later).
- An Object is an instantiation of a class:
 - A variable is declared to be of a particular class
 - It is initialised by allocating memory and creating an instance of the class in memory.
 - The memory location of the object is then placed in the object variable.

The main() method

- A Java program starts by executing the main() method.
 - The main method provides the starting point for the program.
 - Within the main() method:
 - » Other methods will be invoked.
 - » Objects will be created and
 - » methods within those objects invoked.

```
public static void main ( String [] args)
```

↑ ↑ ↑

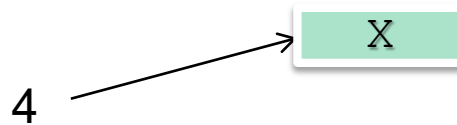
type of method method name method parameters

- For the moment do not worry about the parameters to main(). This will be covered in DSA

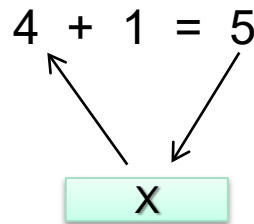
Variables and the Assignment Statement

- Variables are where data is stored in a program.
- Information is placed in a variable using an assignment statement.
- Example:

– $x = 4;$



– $x = x + 1;$





Assignment Statement

- In mathematics the equals sign is a statement of fact.
 - The left hand side is the same as the right hand side.
- In programming the equals sign is performing the action of place the value on the right hand side in the variable specified on the left hand side.
 - Change the left hand side to be the same as the right hand side



Java: Good and Bad

- Good:
 - Platform independent execution.
 - Platform independent binary data (files etc).
 - Robust
 - Does not allow operator overloading. Some people regard this as a limitation. Others think operator overloading is not a good idea anyway!
 - Comes with a huge class library which allow:
 - » File input/output
 - » Graphics
 - » Event trapping/handling
 - » 3D modeling.
- Bad:
 - Syntax is adopted from C. This means that some control structures are primitive and unstructured.
- As you work through the rest of this unit (and future units) , the good and bad will make more sense to you.



Data Types

- As was stated earlier, the manner of interpretation of the 1's and 0's varies for different data types stored
- We need to specify the type of each piece of data held in memory.
- Syntax of declarations is easy; challenge is identifying the correct type.
 - » triangleSideA this is obviously a real
 - » yearOfBirth this is obviously an integer
 - » studentName this is obviously a string of characters
 - but what about?
 - » dateOfBirth single 6 (8?) digit integer
 - or 3 separate integers
 - » age single 6 digit integer
 - or 3 separate integers
 - or a single real
 - » phoneNo single integer
 - or string of characters
- In algebra we often "assume" what the type (and domain) is but machines need the type unambiguously defined.



Variables, Constants and Literal Values

- A variable is a piece of memory in which data can be stored and retrieved. It has a name associated with it and must be declared.

```
int    thisIsAnInteger;
```

- A constant is similar to a variable except that its value is set initially and can never be modified.

- `public static final int MYCONST = 42;`

- A literal value refers to the symbols which can be used to represent the possible values available for a particular data type.

integer	-12, 42, 0
real	-10.2, 56.8, 0.0



Java's Primitive Data Types

– Java defines 8 primitive types

Java Type	Memory format	Range/Domain	Range/Domain
byte	8 bit integer	-2^7 to $2^7 - 1$	-128 to 127
short	16 bit integer	-2^{15} to $2^{15} - 1$	-32768 to 32767
int	32 bit integer	-2^{31} to $2^{31} - 1$	-2147483648 to 2147483647
long	64 bit integer	-2^{63} to $2^{63} - 1$	$\pm 9.22337\text{E}+18$
float	32 bit floating point	± 6 sig. digits ($10^{-46}, 10^{38}$)	
double	64 bit floating point	± 15 sig. digits ($10^{-324}, 10^{308}$)	
char	16 bit Unicode character	All Characters	
boolean	boolean	true, false	



Integer Data Types

- Integer: positive or negative value that consists of whole number
- The Java primitive types byte, short, int & long are abstractions of integers from the mathematical world.
- The range of integers is determined by the amount of storage available (memory) for a particular data type.
- The accuracy is guaranteed.
 - stored as the exact base2 equivalent of the base10 integer



Range of Integers

- Determined by how many distinct base 2 values can be stored in the given number of bits: every additional bit doubles the size of the range
- For N bits you always need one bit for the sign and the remaining N-1 bits can represent 2^{N-1} different bit combinations that directly relate to their binary value
- Note that the lack of symmetry is because of the need to represent zero as one of the 2^{N-1} values i.e.

2^{N-1} negative values, 0, $2^{N-1}-1$ positive values

negative values stored as the 2's complement of the number

- When an attempt to store a number which is larger/smaller than the maximum/minimum value then Integer Overflow occurs.



Real Numbers

- Positive or negative value that consists of whole number plus fractional part (expressed in floating point, or scientific notation)
- The Java types float & double are an abstraction of the real numbers that exist in the mathematical world
- The range and accuracy of real numbers are limited in any computing system (why?)
 - Hint – how would you store $\frac{1}{3}$ or $\sqrt{2}$



Range and Accuracy of Real Numbers

- Determined by number of bits & the split up of mantissa & exponent
- There has to be a limit on the range: by definition you need an infinite number of bits to represent infinity!
- Accuracy is obviously limited
 - The number of significant digits is limited.
 - » There are an infinite number of real values between any two points on the number line.
 - Irrational numbers.
 - Recurring decimals
 - IEEE 754 form (binary conversion)



Real and Integer Expressions

- Real operands used with $+$ $-$ $*$ $/$ produce real results

Expression	Result
$27.3 + 8.4$	35.7
$7.0 - 10.0$	-3.0
$3.0 * 5.0$	15.0
$11.0 / 4.0$	2.75

- Integer operands used with $+$ $-$ $*$ $/$ $\%$ produce integer results

Expression	Result
$27 + 8$	35
$7 - 10$	-3
$3 * 5$	15
$11 / 4$	2
$11 \% 4$	3
$10 \% 2$	0

Integer Arithmetic

- the integer truncation feature of `/` (`div`) and the remainder operator `%` (`mod`) are powerful and very useful tools
 - think of long division here

$$\begin{array}{r} 2 \\ 4 \overline{) 11} \\ \underline{8} \\ 3 \end{array}$$

Diagram illustrating integer division and remainder. The quotient 2 is labeled "Div" and the remainder 3 is labeled "Mod".

- Assume Year holds 4 digit year e.g. 1998
 - » $(\text{Year} / 100) + 1$ evaluates to century
- Other examples
 - » $\text{pageNo} = (\text{lineNo} / \text{noLinesPerPage}) + 1$
 - » $\text{hours} = \text{hhmm} / 100$
 - » $\text{minutes} = \text{hhmm} \% 100$



Mixed Mode Arithmetic

- Mixed mode arithmetic occurs when a numeric expression contains a mixture of integer and reals.
 - » $y = 3 + 4.5;$
 - » $z = 2 / 3.0;$
- Programming languages always have a set of rules for evaluating mixed mode expressions, but:
 - Not the same across different programming languages
 - Not always supported by compiler.
- Errors caused by mixed mode arithmetic in program code are extremely difficult to find.
- The rule is NEVER use mixed mode arithmetic



Type Casting

- To convert from one data type to another a type cast is used.
- The syntax is:

`(NewDataType) (expression)`

- Examples:
 - Assume a, b, c, d are of type int and x, y, z and average are of type double
 - `average = (double) (a + b + c + d) / 4.0;`
 - » a + b + c + d are added first, the value is converted to a double, then divided by 4.0, then assigned to average
 - `z = (double) (a + b);`
 - » a and b are added, the value is converted to a double, then assigned to z
 - `a = (int)y;`
 - » the value of y is truncated to an int, then assigned to a (y is NOT changed)
 - `x = (double) (a / b);`
 - » this is a div b, then converted to double and assigned to x
 - » if a is 5 and b is 2, x is assigned 2.0
 - » `x = (double) (5/2);`
 - `y = (double)a / (double)b;`
 - » this is convert both the values of a and b to doubles, then normal division
 - » same as `y = 5.0/2.0;`
- Note that conversion from a real data type to an integer data type involves truncating the real value (i.e. not rounding!).



Operator Precedence

- below are the Java operations listed from higher to lower precedence. Note: there are many more.

Operator	java	Associativity
ops on references	<code>.</code> <code>[]</code>	L => R
unary	<code>-</code> <code>++</code> <code>--</code>	L => R
multiplicative	<code>*</code> <code>/</code> <code>%</code>	L => R
additive	<code>+</code> <code>-</code>	L => R
relational	<code>></code> <code>>=</code> <code><</code> <code><=</code>	L => R
equality	<code>==</code> <code>!=</code>	L => R
logical AND	<code>&&</code>	L => R
logical OR	<code> </code>	L => R
assignment	<code>=</code>	R => L



Examples

Expression	Result
$7 + 23 * 6$	$= 7 + 138 = 145$
$3 * 2 + 4 * 5$	$= 6 + 20 = 26$
$-6 * 2$	$= -12$
$3 + 5 * 6 / 4 + 2$	$= 3 + 30/4 + 2$ $= 3 + 7 + 2 = 12$
$3.0 + 5.0 * 6.0/4.0 + 2.0$	$= 3.0 + 30.0/4.0 + 2.0$ $= 3.0 + 7.5 + 2.0 = 12.5$
$-6 * 2 + 3 / 4$	$= -12 + 0 = -12$
$2 * 5 \% 2$	$= 10 \% 2 = 0$



Expression Guidelines

- Never use mixed mode arithmetic.
 - Use type casting to avoid mixed mode arithmetic.
- Precedence rules are the same as in mathematics.
- Use parentheses to simplify readability of complex expressions.
- Use intermediate steps to split complex expressions into explicitly separate steps.
- Don't over-parenthesise simple expressions.
- Beware of algebraic simplicity:
 - $$x = \frac{y - p}{z - q}$$
 - is written in Java as: `x = (y - p) / (z - q);`



Assignment Operators

- Short hand way of modifying the contents of a variable.

- Traditional

```
x = x + 5;
```

```
x = x - 32;
```

```
fred = fred * 2;
```

```
ralph = ralph / 6;
```

- Alternative

```
x += 5;
```

```
x -= 32;
```

```
fred *= 2;
```

```
ralph /= 6;
```

- Must be careful though:

```
y *= x - 2;
```

- is the same as:

```
y = y * ( x - 2 );
```

- but not the same as:

```
y = y * x - 2;    or    y = (y * x) - 2;
```



The Increment/Decrement Operator

- Increment ($++$)/ Decrement ($--$)
 - $x++;$ is the same as $x = x + 1;$
- and
 - $x--;$ is the same as $x = x - 1;$
- Must be careful:
 - $x = x++;$ is nonsense!
- Also $++x$ and $--x$
 - work differently in expressions
 - do not use in an expression!



Character data types

- Java provides two character data types:
- char:
 - Single character
 - primitive data type
 - Constants are enclosed in single quotes (e.g. 'a').
- String:
 - Zero or more characters
 - Strings are objects not primitive data types
 - Constants are enclosed in double quotes (e.g. “a”).



char

- char stores a single character e.g.: 'a', 'A', '6', '&', etc.
- stored in Unicode, a standard that arbitrarily designates a bit pattern to represent a particular character symbol
- if the character is a decimal digit e.g. '8' can't do arithmetic with it: '8' + '6' can't possibly be expected to be meaningful
- a character occupies 16 bits & is coded according to the Unicode standard thus there are >32,000 different possible combinations to represent different all the possible characters ... more than enough!
 - the lower (rightmost) 8 bits is identical to the ASCII system
- order of the characters is determined by the codes:

'A' < 'B' ... < 'Z' < ... < 'a' < 'b' ... < 'z'

refer to www.asciitable.com



The Java String class

- Generally a string is a collection of 0 (empty) or more characters.
- The Java String class provides us with the facility to handle strings.
- String variables are objects but they can be used as if they were primitive variables.

```
String unitName = "Programming";
```

- Can also be treated as an object:

```
String unitName = new String("Programming");
```



The Java String class

- Must never forget strings are objects.

- What is wrong with:

```
(nameOne == nameTwo)
```

- Strings can also be concatenated using the plus operator.

- Example

```
int x = 42;
```

```
String message;
```

```
message = "x = " + x;
```

- Java converts the value of x (42) to a String (“42”) and joins it to the end of "x = " to make "x = 42"



Java toString() method

- A convention is that all Java objects should have a toString() method.
 - This method is used to provide a String representation of the data stored in the object.
 - » covered later



Arrays

- The variables that we have seen so far represent a single item.
 - eg: `int numTiles` is a single integer number
- But we also often work with *sets* of similar data
 - eg: the list of student marks in OOPD. How to handle?
 - » `double student1Mark, student2Mark, student3Mark, ...?`
 - Clumsy!
 - » Variable names defined at compile time – ‘hard-coding’:
program can never change the number of students
 - » Calculating the average involves a massive amount of typing
 - » Can’t conveniently pass the set of students around



Arrays

- Arrays are a solution to this problem
- Simplest kind of data structure for storing sets of data
 - Arrays are built-in to *all* programming languages
 - Instead of just one element, an array is a variable that contains *many* elements
 - The array variable itself is a reference to the first element of the array
 - » Java: the array variable also knows how large the array is
 - » C: doesn't store the array length – you have to do it yourself!



Array Properties

- Elements are located sequentially in memory
 - ie: the array is a *contiguous* block of memory
- All elements must have same data type
 - eg: double
- Arrays can be initialised to any size
 - within memory limits
- However, once initialised they cannot be resized
 - Must create a new array and copy over the contents of the old array in order to ‘resize’ an array

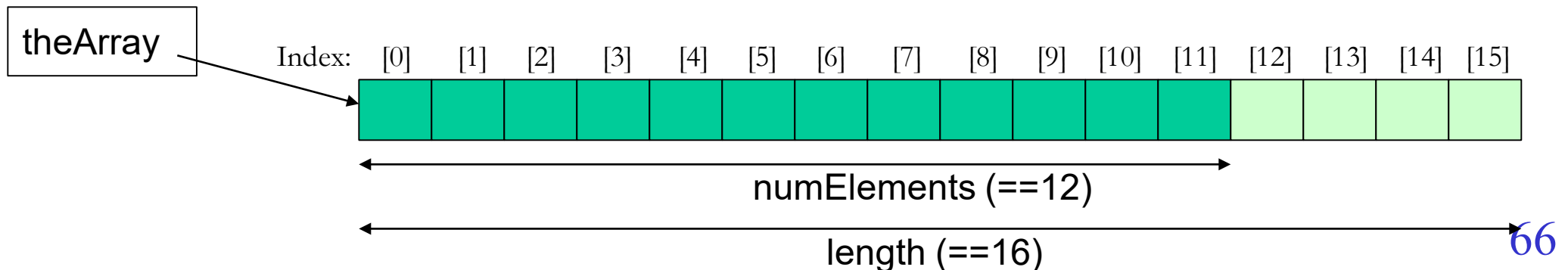


Array Properties

- Array capacity (length) vs actually used elements
 - Initialising an array to (say) length=20 doesn't *set* 20 elements, it merely *reserves space* for 20 elements
 - » Hence initialisation is also referred to as *allocation*
 - You therefore typically need to keep track of how many elements you have *actually used* in the array
 - » ie: the count of elements, as distinct from the array capacity
 - » It is typical that you allocate more space than you initially need, since arrays have a fixed capacity and cannot be resized

Arrays – Accessing Elements

- Once you have allocated an array, you need to be able to work with the elements inside the array
- Elements are accessed via an *index* (or *subscript*)
 - The index is the element number in the array
 - » 0,1, 2, 3, 4, ... N-1, where N is the allocated size (length)
 - » the index is an *offset* from the first element





Arrays In Code (Java)

- Declaring: put '[' on the end of the data type
 - » `double[] theArray;`
 - » Any data type can be used with arrays, including classes
- Allocating: use new keyword with special [] syntax
 - » `theArray = new double[100];`
- Indexing: `theArray[index]`, index must be an int
 - » Negative indexes or indexes that are past the end of the array
 - » (ie: \geq length) will cause an error during runtime
- Assignment: `sameArray = theArray;`
 - » Assignment doesn't copy the array *contents*, it only makes the l.h.s. variable point at the *same array* as the r.h.s.
 - » Same with passing an array as a parameter to a method



Primitive or Object?

- All of the primitive data types have object equivalents.

Class	Primitive
Integer	byte, int, short, long
Float	float
Double	double
Character	char
Boolean	boolean



Initialising Variables

- What is stored in a variable when it is created?
- Java auto-initialises variables:
 - Primitive Variables:
 - » Numeric set to zero
 - » char set to blank
 - » boolean set to false
 - Object variables:
 - » set to null.
 - » null represents an invalid memory address.
- Not all programming languages auto-initialise so it is extremely poor programming style to rely on auto-initialisation.
- You should always explicitly initialise variables.



Summary

- We have covered a lot today!
 - It is extremely important to consolidate this lecture
 - Watch the iLecture a couple of times
- Attend the practical sessions
 - Start in week 1
 - Compulsory (Sign-offs)
 - Linux labs are building 314, rooms 218, 219, 220, 221 & 232
 - Use labs outside of your prac time if there is no class or there are free computers (check with tutor first)
 - Building and labs are 24/7 access
- Install Java so you can work at home
 - You can also remote login to machines using ssh



Next Module

The next module will address the following issues:

- Program documentation
- Testing
- Boolean Operations
- Control structures.
 - » Selection
 - The If-Then-Else Statement.
 - The Case Statement