Software Engineering1 (Java)

CSY1019

Second Year - Computing

Level 5:	CS	MC	CN	C	IT	GV	SE	Level 5:
CSY2001 Computer Networks	20	20	20		20			CSY2001
CSY2002 Operating Systems	20	20	20	20				CSY2002
CSY2006 Software Engineering 2							20	CSY2006
CSY2008 Formal Specification of Software Systems							20	CSY2008
CSY2015 Microprocessor Systems	20							CSY2015
CSY2026 Modern Networks		20	20	20	20	20		CSY2026
CSY2027 Group Project	20	20	20	20	20	20	20	CSY2027
CSY2028 Web Programming	20	20	20	20	20	20	20	CSY2028
CSY2029 Databases 2			20	20	20	20	20	CSY2029
CSY2030 Systems Design & Development	20	20		20	20	20	20	CSY2030
CSY2033 Graphics 2D						20		CSY2033

Second Year – Business Computing

	BS	WD	
Level 5:			**Level 5:**
Group Project	20	20	CSY2027
Web Programming	20	20	CSY2028
Databases 2	20	20	CSY2029
Systems Design & Development	20		CSY2030
Quality and User-Centred Systems	20	20	CSY2041
Service Management	20		BUS2015
Web & Social Media Management		20	MKT2038
Creative Design for the Web		20	MKT2039

Games Development at UoN
School of the Arts
&
School of Science & Technology

Course Structure

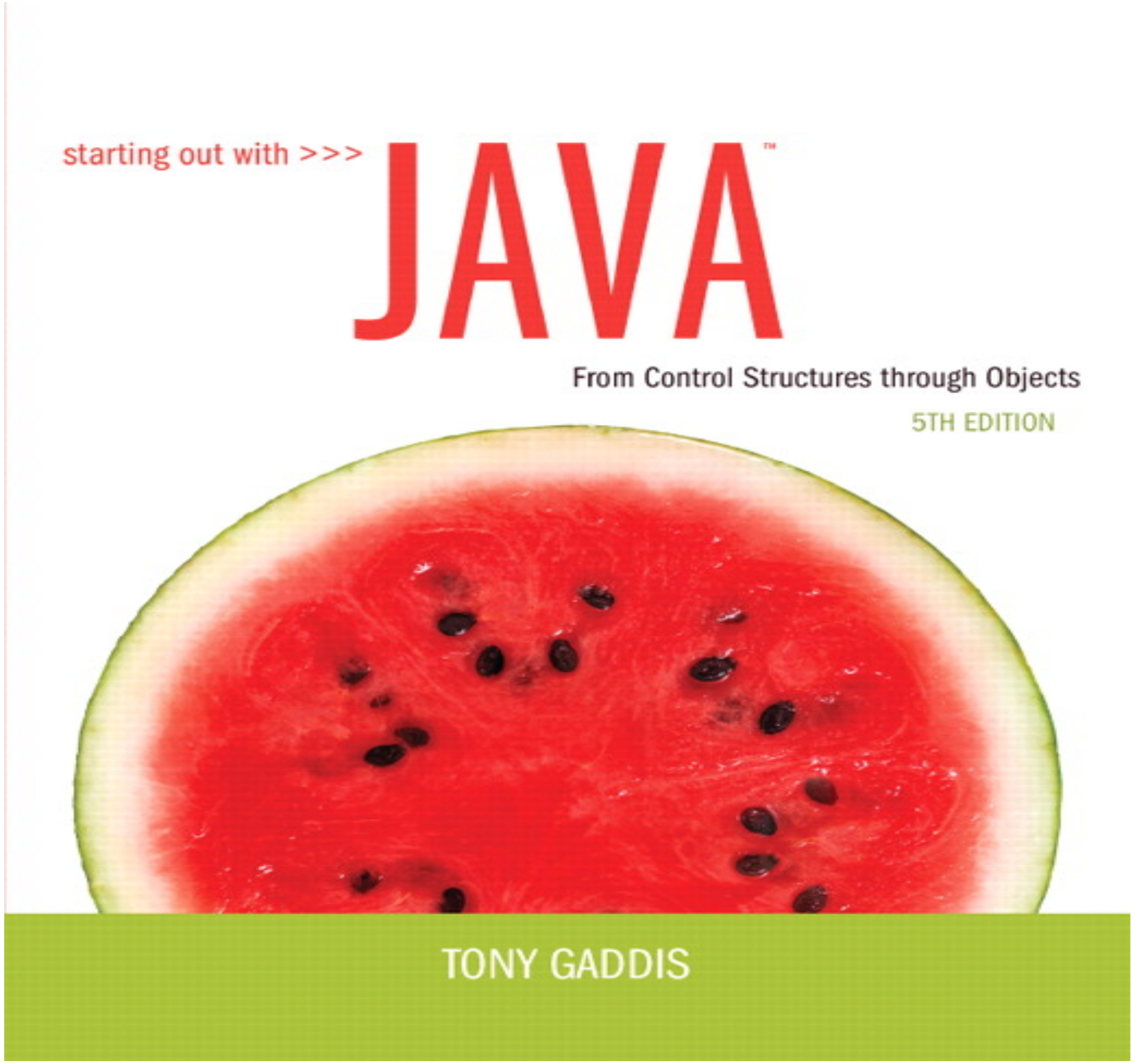
Level 4 (Year 1)		Level 5 (Year 2)		Level 6 (Year 3)	
BA Games Art	BSc/HND Computer Games Development	BA Games Art	BSc/HND Computer Games Development	BA Games Art	BSc Computer Games Development
3DD1007 Visual Studies 1	CSY1018 Internet Technology	3DD2028 Visual Studies 2	CSY2006 Software Engineering 2	3DD4009 Art Director Portfolio, Final Major Project	CSY4010 Computing Dissertation
3DD1055 2D Digital Practice	CSY1019 Software Engineering 1	3DD2063 3D Modelling, Technical Art	CSY2026 Modern Networks	3DD3038 Professional Practice	CSY3028 Graphics 3D
3DD1056 3D Modelling	CSY1020 Problem Solving & Programming	3DD2064 3D Organic Modelling	CSY2028 Internet Programming	3DD3037 Visual Studies 3	CSY3030 Games Techniques 3
	CSY1021 Database 1		CSY2033 Graphics 2D		CSY3029 Mobile Computing 2
	CSY1024 Games Techniques 1		CSY2034 Games Techniques 2		
CSY1025 Group Project 1 (Games)		CSY2035 Group Project 2 (Games)		CSY3031 Group Project 3 (Games)	

Module Information

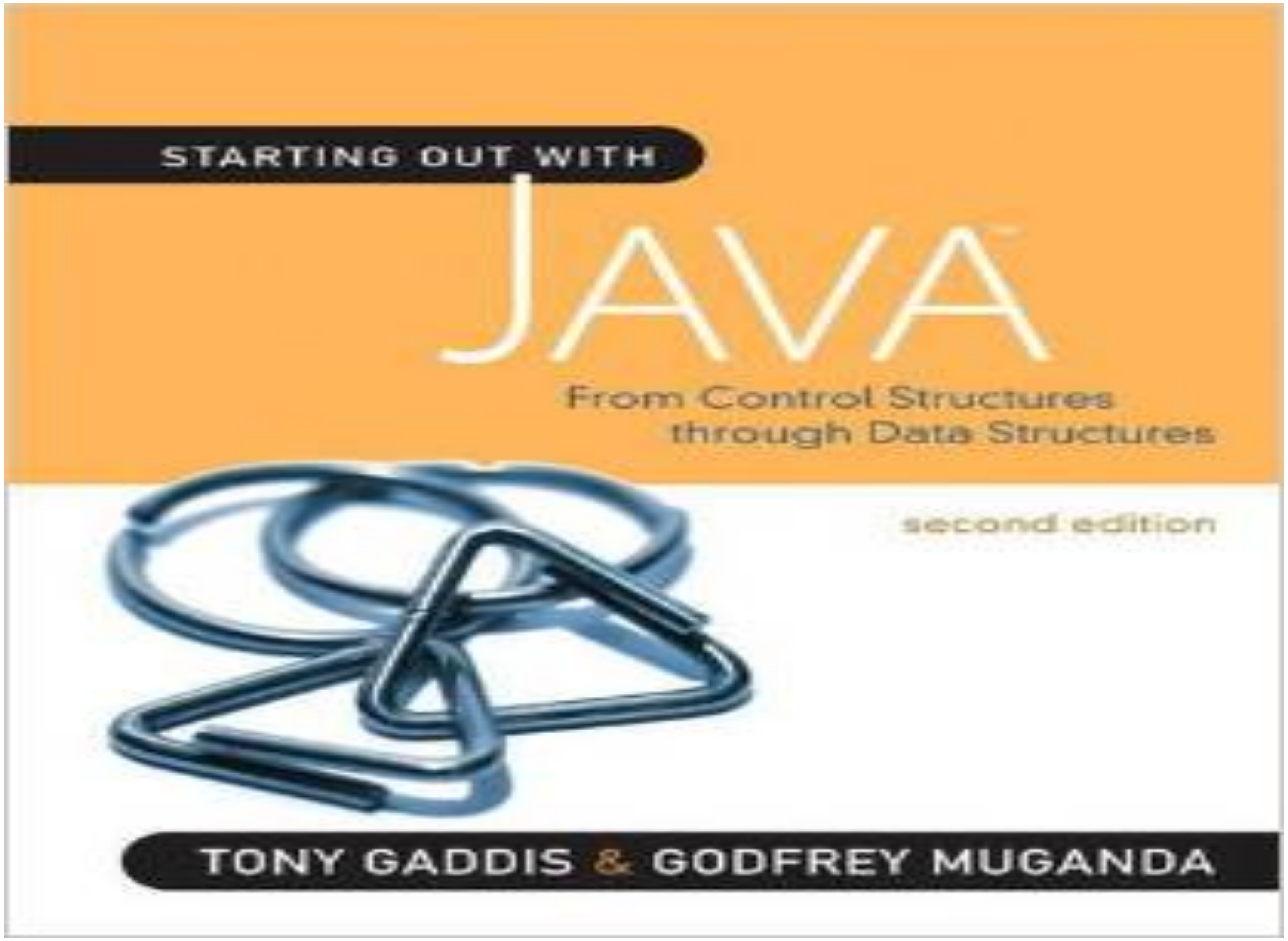
(Available on NILE)

- Module Content (Indicative)
- Late Objects Approach
- Assessment Strategy
- Module Materials
  - Lecture materials, Lab exercises, solutions
  - Useful e-resources (guides, tutorials)
- Reading List
- Contact Details
- Announcements (Keep an eye)

Text Book

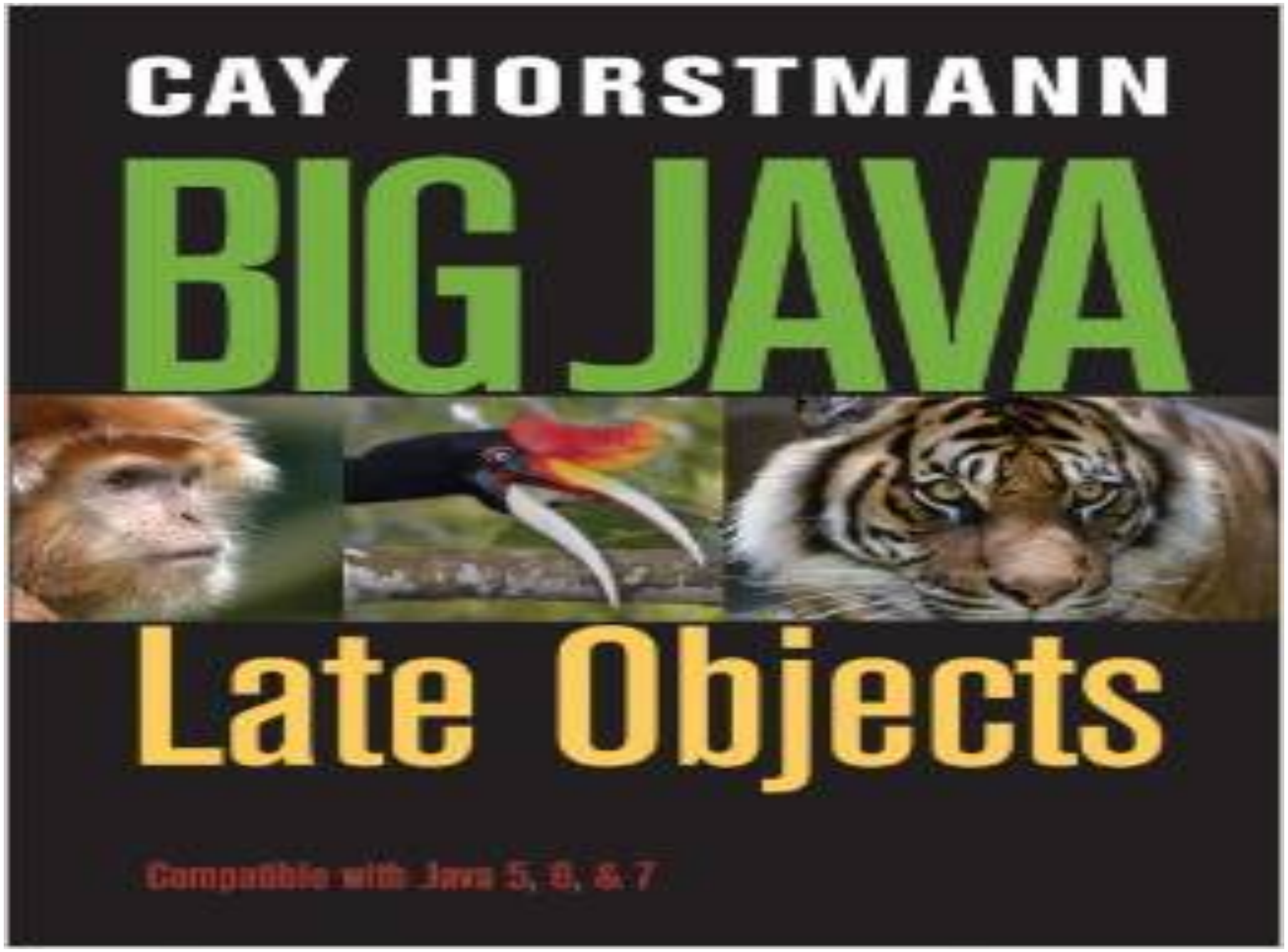

starting out with >>> JAVA™
From Control Structures through Objects
5TH EDITION
TONY GADDIS

Text Book (Alternative)



STARTING OUT WITH
JAVA
From Control Structures through Data Structures
second edition
TONY GADDIS & GODFREY MUGANDA

(For Challenging Exercises)



Basics

- Discipline
- Attendance (both lecture and lab)
- Additional Reading (not just lecture notes)
- Practise! Practise! Practise!
- Ask Yourself! (Why?, What?, How?, When?)

  e.g. when an apple is thrown up why does it fall down

  (Logical Reasoning is important)

Software Engineering

- The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software

  -IEEE

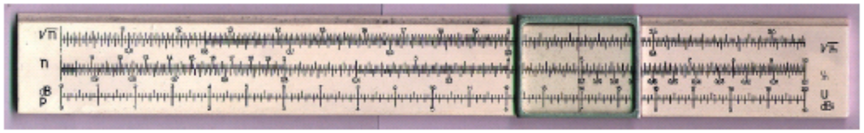
- It is the application of Engineering to software because it integrates significant mathematics, computer science and practices whose origins are in Engineering

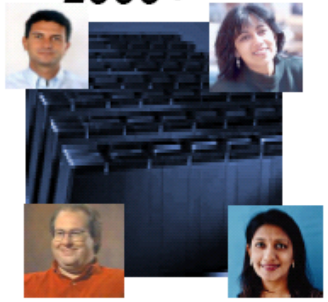
  - ACM

… much more than just programming

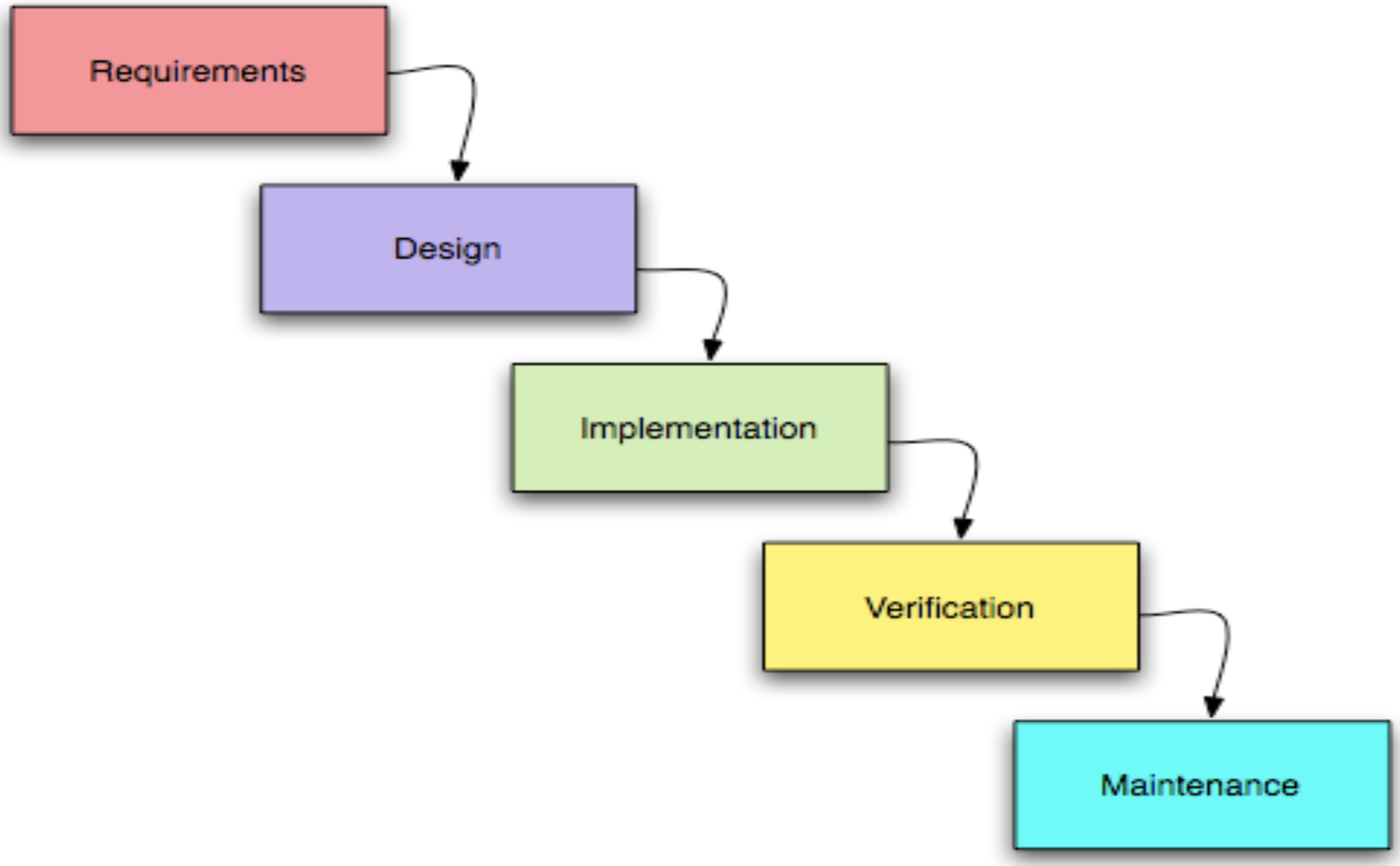
Augustine's Law – Growth of Software: Order of Magnitude Every 10 Years

In The Beginning



1960's	1970's	1980's	1990's	2000+
				
F-4A 1000 LOC	F-15A 50,000 LOC	F-16C 300K LOC	F-22 1.7M LOC	F-35 >6M LOC

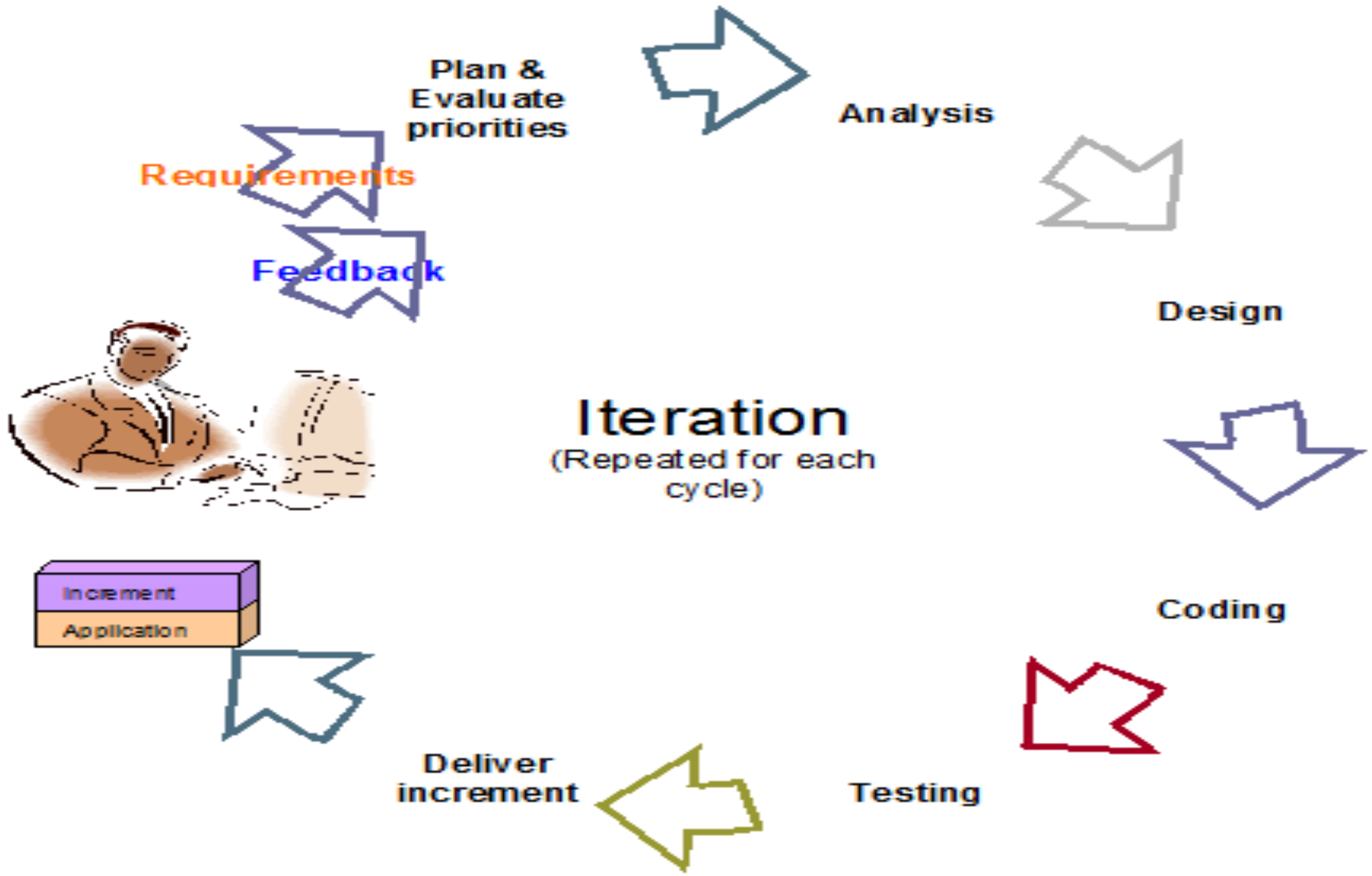
Traditional Software Development Process



Software Development Process

- *Software requirements* specify *what* a program must accomplish. Requirements are expressed in a document called a *Software Requirements Specification*
- A *software design* indicates how a program will accomplish its requirements
- *Implementation* is the process of writing the source code that will solve the problem
- *Verification/Testing* is the act of ensuring that a program will solve the intended problem given all of the constraints under which it must perform
- *Maintenance* is the act of improving software programs after delivery for reusing it in the future

Agile Software Development



What is a program?

- Instructions
  e.g. recipe to make a curry

A computer program is a sequence of instructions written to perform a specified task with a computer

Algorithm

- **Algorithm**: A list of steps for solving a problem.

- **How does one bake sugar cookies?**
  (what is the "bake sugar cookies" algorithm?)
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.
  - Set the oven for the appropriate temperature.
  - Set the timer.
  - Place the cookies into the oven.
  - Allow the cookies to bake.
  - Mix the ingredients for the frosting.
  - Spread frosting and sprinkles onto the cookies.
  - ...

Programming Languages

- Machine Language consisting of binary code (0s and 1s). Computers understand only this. It is processor dependent.
- Assembly Language consisting of mnemonics (symbols) to make programming less tedious and faster. (e.g. Load A, Add B, Store C). An *Assembler* converts assembly language into machine language. It is also processor dependent.
- High-level Language consisting of English-like instructions to make programming simpler and faster. (e.g. C= A+B). A compiler/interpreter helps convert high-level language into machine language. It is not processor dependent.

Some High-level languages

- *procedural languages*: programs are a series of commands
  - **Pascal** (1970): designed for education
  - **C** (1972): operating systems and device drivers
- *functional programming*: functions map inputs to outputs
  - **Lisp** (1958) / **Scheme** (1975), **Haskell** (1990)
- *object-oriented languages*: programs use interacting "objects"
  - **Smalltalk** (1980): first major object-oriented language
  - **C++** (1985): "object-oriented" improvements to C
  - **Java** (1995): general purpose language and world's most widely used computer programming language. (Deitel & Deitel)

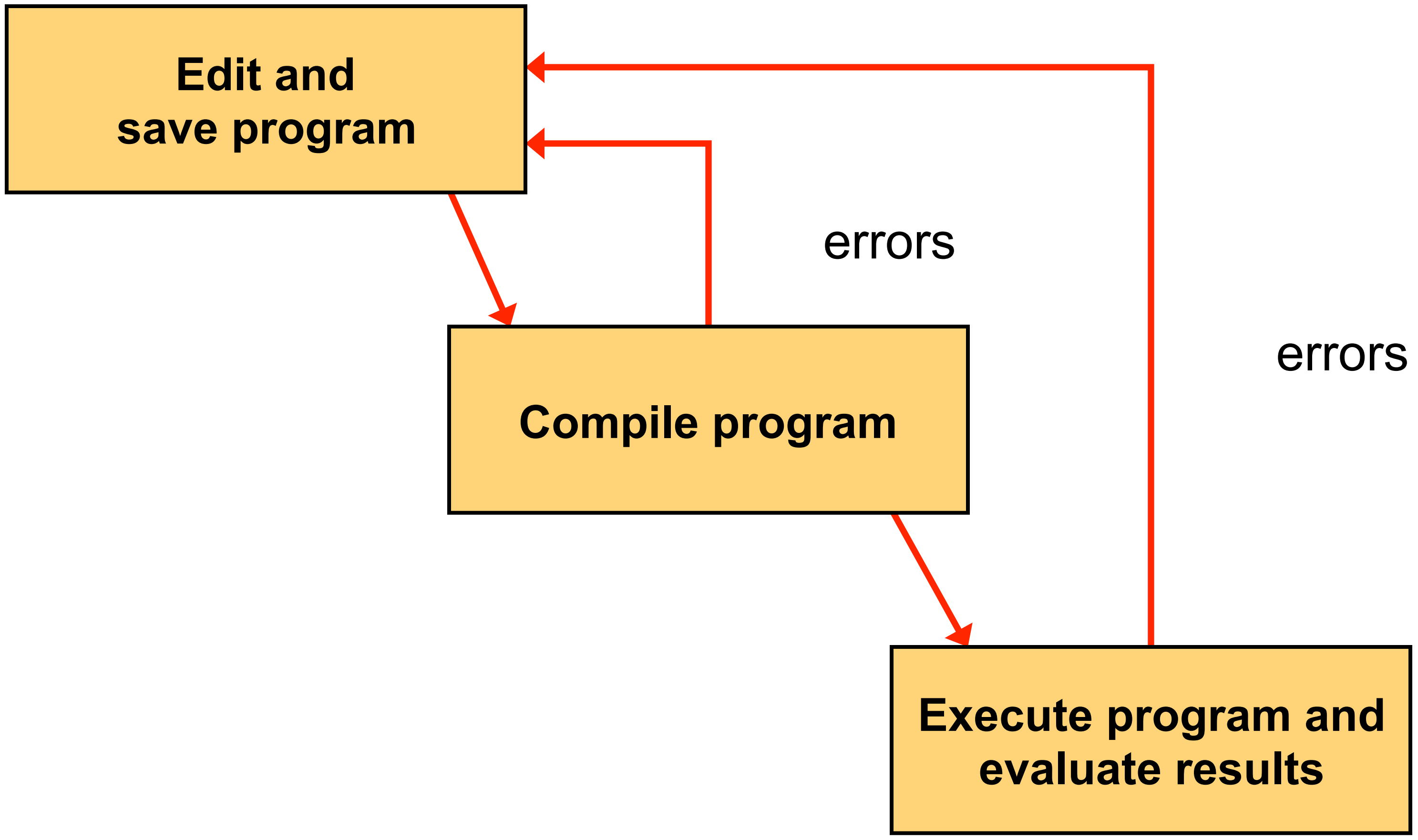
Syntax and Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we <u>meant</u> to tell it to do

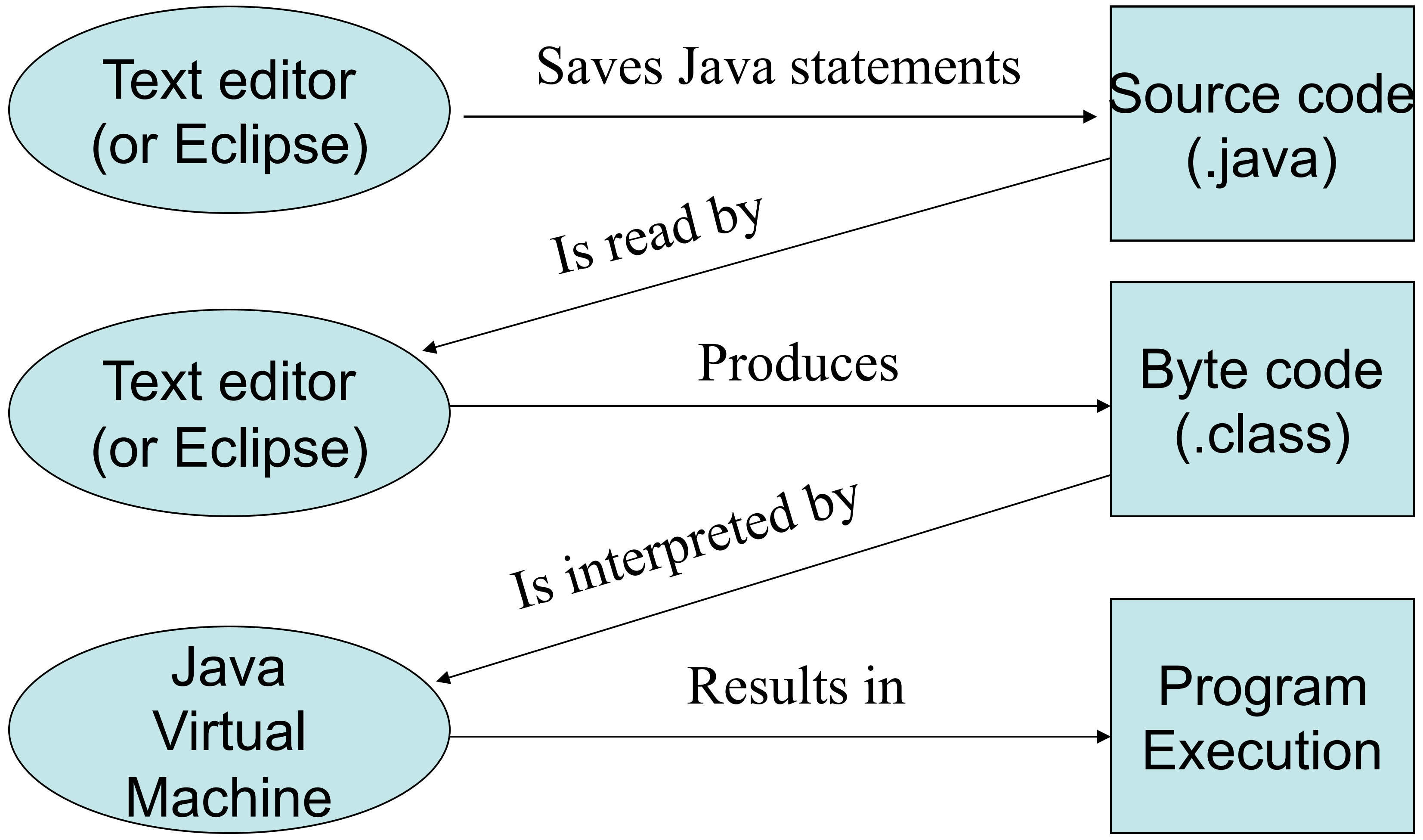
Types of Errors

- A program can have three types of errors
  - The compiler will find syntax errors and other basic problems (*compile-time errors*)
    - If compile-time errors exist, an executable version of the program is not created
  - A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
  - A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

Basic Program Development



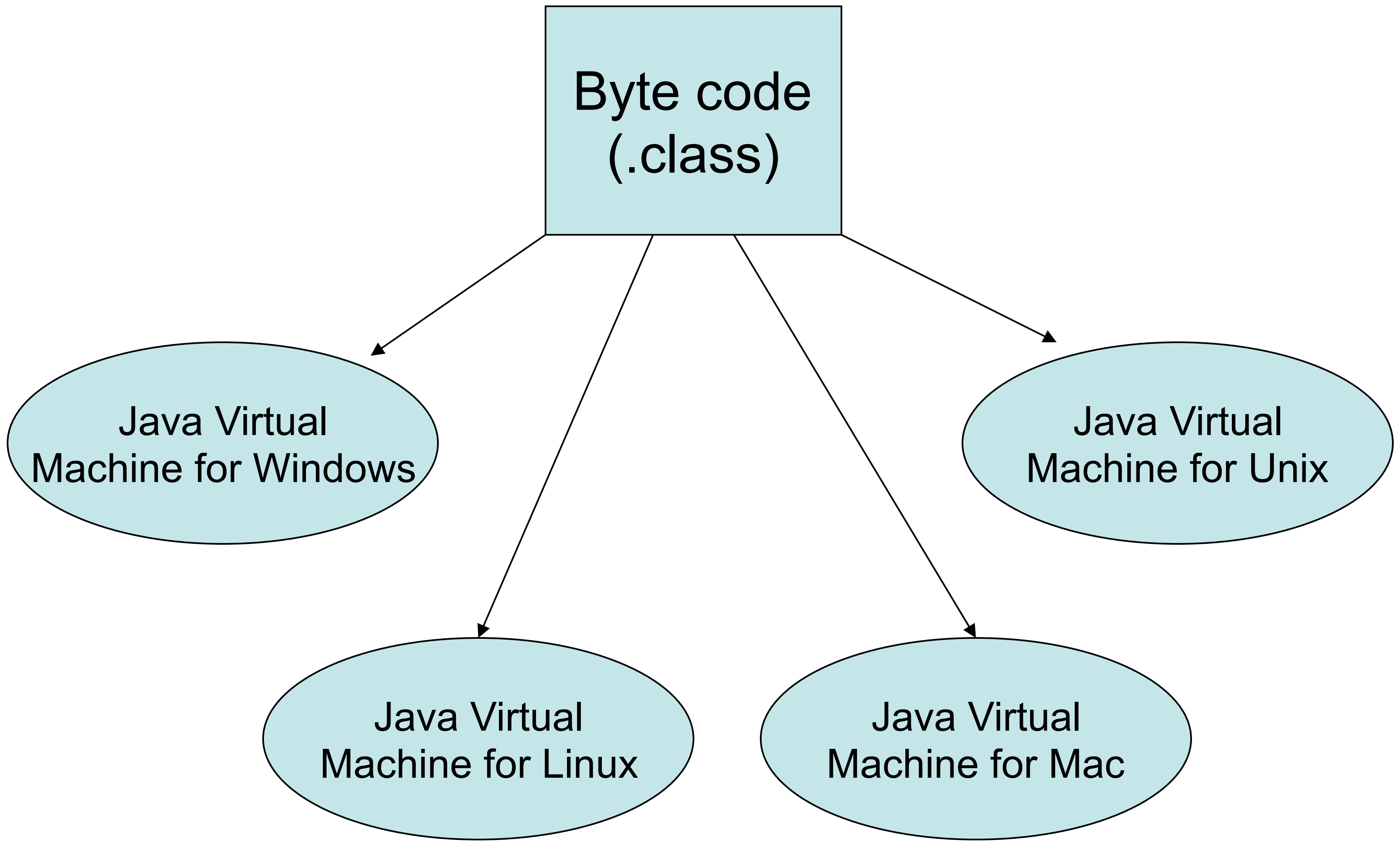
Java Program Development



Java : Portability and Platform Independence

- *Portable* means that a program may be written on one type of computer and then run on a wide variety of computers, with little or no modification.
- Java byte code runs on the JVM and not on any particular CPU; therefore, compiled Java programs are highly portable.
- JVMs exist on many platforms:
  - Windows
  - Mac
  - Linux
  - Unix
  - BSD
  - Etc.

Java Portability


Byte code
(.class)
Java Virtual Machine for Windows
Java Virtual Machine for Linux
Java Virtual Machine for Mac
Java Virtual Machine for Unix

Structure of a Java program

class: a program

```
public class name {
public static void main(String[] args){
            statement;
            statement;
            ...
            statement;
      }
}
```

method: a named group of statements

statement: a command to be executed

- Every executable Java program consists of a **class**,
  - that contains a **method** named `main`,
    - that contains the **statements** (commands) to be executed.
    - each statement ends with semicolon

Sample Program

```
public class HelloPrinter {
    public static void main(String[] args) {
        // Display a greeting in the console window
        System.out.println("Hello, World!");
    }
}
```

Program Run:

```
Hello, World!
```

Code Convention is important

Parts of a Java Program

- Comments
  - The line is ignored by the compiler.
  - The comment in the example is a single-line comment.
- Class Header
  - The class header tells the compiler things about the class such as what other classes can use it (public) and that it is a Java class (class), and the name of that class (HelloPrinter).
- Curly Braces
  - When associated with the class header, they define the scope of the class.
  - When associated with a method, they define the scope of the method.

Short Review

- Java is a case-sensitive language.
- All Java programs must be stored in a file with a .java file extension.
- Comments are ignored by the compiler.
- A .java file may contain many classes but may only have one public class.
- If a .java file has a public class, the class must have the same name as the file.

Names and Identifiers

- You must give your program a name.

```
public class HelloPrinter {
```

  - Naming convention: capitalize each word (e.g. `MyClassName`)
  - Your program's file must match exactly (`HelloPrinter.java`)
    - includes capitalization (Java is "case-sensitive")

- **identifier**: A name given to an item in your program.
  - must start with a letter or _ or `$`
  - subsequent characters can be any of those or a number
    - legal: `_myName` `TheCure` `ANSWER_IS_42` `$bling$`
    - illegal: `me+u` `49ers` `side-swipe` `Ph.D's`

Keywords: words with predefined meaning

`abstract`	`else`	`interface`	`switch`
`assert`	`enum`	`long`	`synchronized`
`boolean`	`extends`	`native`	`this`
`break`	`false`	`new`	`throw`
`byte`	`final`	`null`	`throws`
`case`	`finally`	`package`	`transient`
`catch`	`float`	`private`	`true`
`char`	`for`	`protected`	`try`
`class`	`goto`	`public`	`void`
`const`	`if`	`return`	`volatile`
`continue`	`implements`	`short`	`while`
`default`	`import`	`static`	
`do`	`instanceof`	`strictfp`	
`double`	`int`	`super`	

Comments

- They should be included to explain the purpose of the program and describe processing steps
- They are not executed when your program runs
- Java comments can take three forms:

```
// this comment runs to the end of the line


/*  this comment runs to the terminating
    symbol, even across line breaks          */


/** this is a javadoc comment   */
```

Escape Sequence

- **In Java any character that is preceded by a backslash (\) is known as escape sequence, which has special meaning.** An *escape sequence* inserts a special character into a println statement. **Following is a list of Java escape sequences**

`\n`	new line character (goes to next line)
`\"`	double quotation mark character
`\\`	backslash character
`\t`	tab character (indents output by ~8 spaces)

Example:

```
System.out.println("\\hello\nhow\tare\"you\"?\\\\");
```

Escape Sequence

- Try:

```
System.out.println("Name\nRollNo\nAddress");
System.out.println("Name\tRollNo\tAddress");
System.out.println("Name\"RollNo\"Address");
System.out.println("Name\\RollNo\\Address");
```

How many Lines of output?

```
public class Test {
    public static void main(String[ ] args) {
        System.out.println("Testing, testing,");
        System.out.println("one two three.");
        System.out.println();

        System.out.println("How much output");
        System.out.println();
        System.out.println("will there be?");
    }
}
```

Answer: 6 lines (Blank lines do not count)

(Some) Common Syntax Errors

- File Name not matching Class name
- Misspelled Words (or wrong case)
- Forgetting a semicolon
- Forgetting a required keyword
- Not closing a string literal or comment
- Missing dot
- Not closing braces { }, ( )

This program contains 11 errors

```
public class Buggy
    public static main(String args) {
        System.out.println(Hello world);
        system.out.Pritnln("Do you like this program"?);
        System.out.println()

        System.println("I wrote it myself.";
    {
}
```

Syntax Errors

- line 1: missing { after Tricky
- line 2: missing void before main
- line 2: missing [] after String
- line 3: missing " marks around Hello world
- line 4: system should be System (uppercase S)
- line 4: Pritnln should be println (lowercase P and fixed spelling)
- line 4: ? should be before "
- line 5: missing semicolon after ()
- line 7: missing) after "
- line 8: System.println should be System.out.println
- line 8: { should be }

Corrected Version

```
public class Buggy {
    public static void main(String[] args) {
        System.out.println("Hello world");
        System.out.println("Do you like this program?");
        System.out.println();

        System.out.println("I wrote it myself.");
    }
}
```