Introduction to Object Oriented Programming

Dr. Krishnendu Guha
Assistant Professor/ Lecturer
School of Computer Science and Information Technology
University College Cork
Email: kguha@ucc.ie

My Introduction

Dr. Krishnendu Guha

Assistant Professor/ Lecturer
School of Computer Science and Information Technology
University College Cork

Previous Affiliations

My Research and Teaching

Research Domains: Embedded Systems, Hardware Security, Quantum Computing

Teaching: Java, Web Systems, Embedded Systems and Hardware Security

Course Overview

CS2514: Introduction to Java

24x1=24 hrs lectures
6x2=12 hrs practicals

Timing Schedule:
Wednesday 2-3 pm, WGB 01
Thursday 5-6 pm, BHSCG05
Monday (practicals) 3-5pm, WGB 110

Total Marks 100
Written Examination 80 Marks - Semester 2 Written Exam - Summer
- Paper 1: 1.5hr paper - Written Questions (80 Marks)

Continuous Assessment 20 Marks
- Assignment - 3 x coding assignments (3, 2, & 2 marks) including correct styling (3 marks) (10 Marks)
- Assignment - **1 x coding assignment/ Innovative Project**, 10 marks (10 Marks)

Mode of Study: Interactive

If you don’t understand, stop me and ask questions

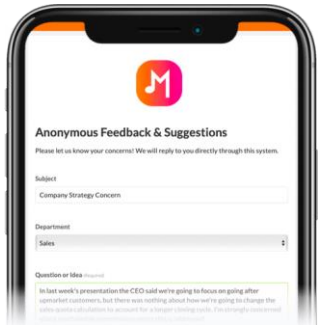
If you want to say something, say

- Email me
- Or use anonymous feedback form

If you want to make a suggestion/ comment

- Email me
- Or use anonymous feedback form

Provide Feedback



SIG Sponsorship for CS2514

The topper of CS2514 in 2024 will get 500 euros

SIG SUSQUEHANNA

Desired Learning Outcomes

Module Objective: To build on the foundation of CS1117, particularly in the areas of ***object-oriented concepts*** and library usage, in ***designing and implementing computer programs*** of increasing sophistication and complexity.

Module Content:

- Class definitions;
- Procedural abstraction and data abstraction;
- Associations between objects;
- Class hierarchies and inheritance;
- Polymorphism and dynamic method binding.

Resources

Internet

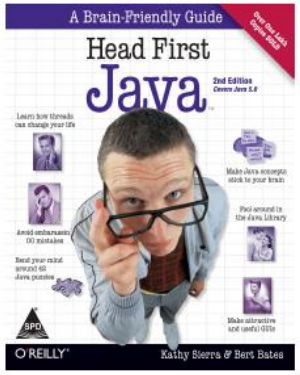
javaTpoint

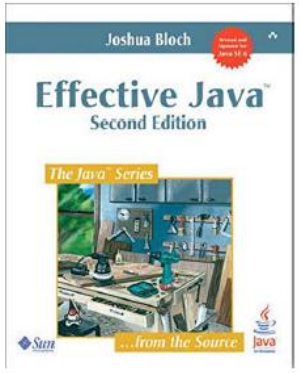
W3 schools

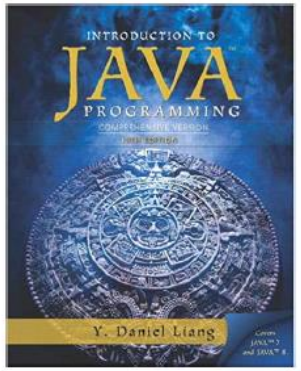
etc.

etc.

Books





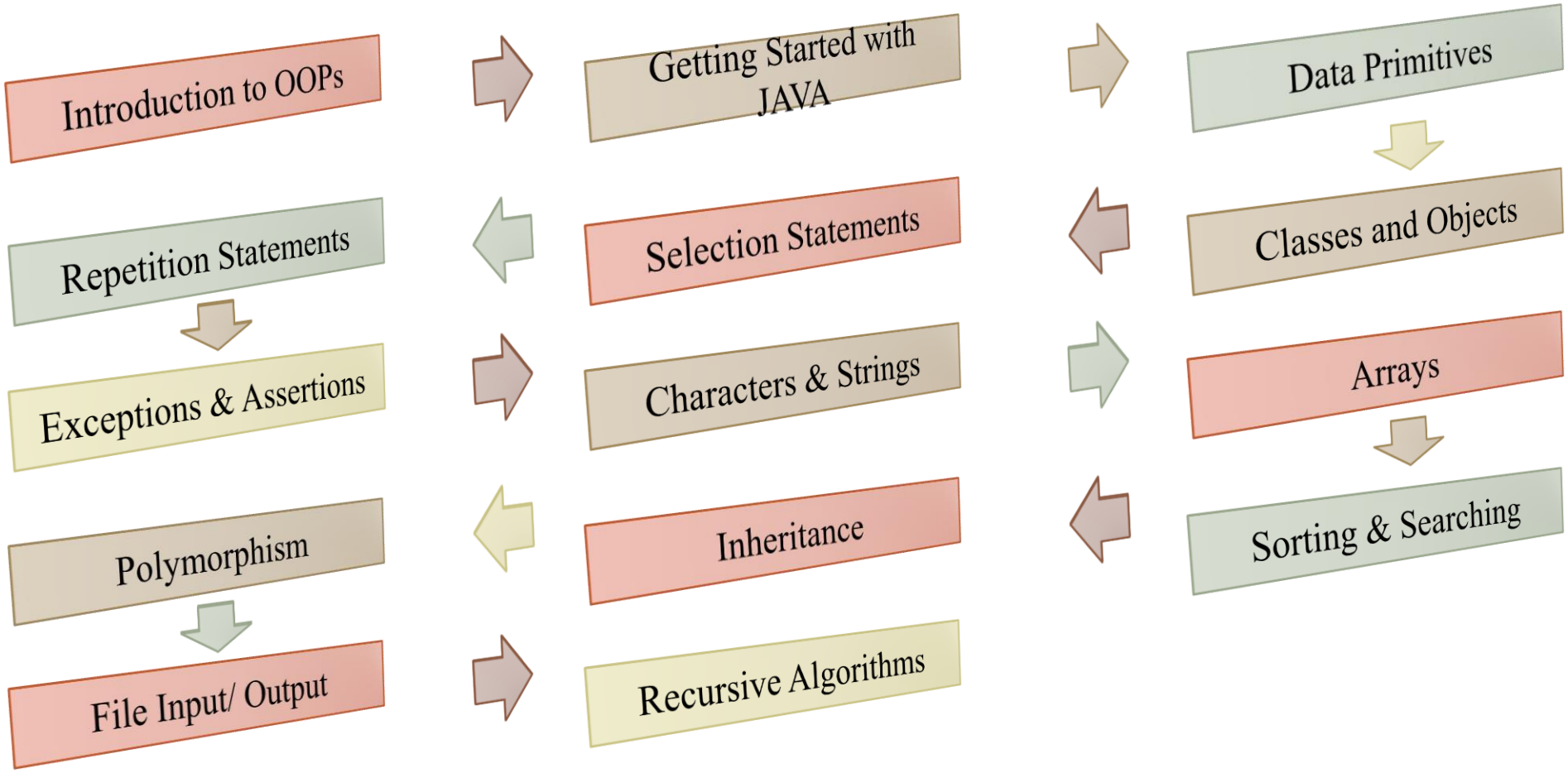


etc.

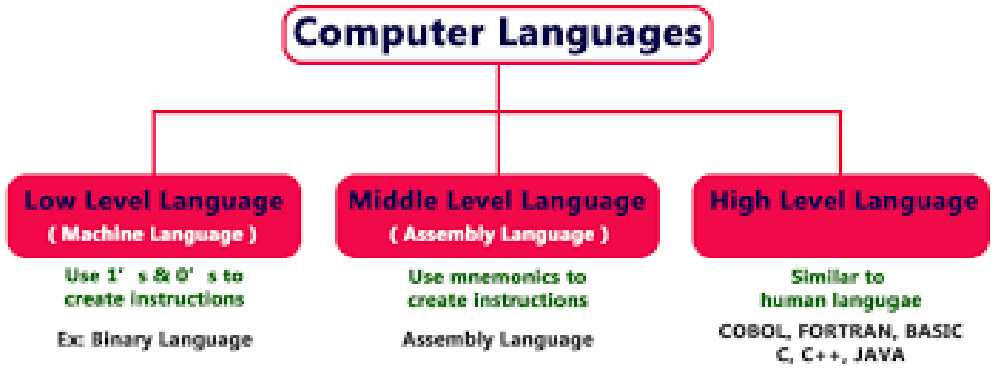
etc.

The lecture material is designed to be used in class as part of a guided discovery sequence.
This is not a comprehensive source.
Please use this for revision purposes, attend the labs regularly, and work on your continuous assessments.
You must be actively thinking, questioning, and practicing to learn.

Topics Overview

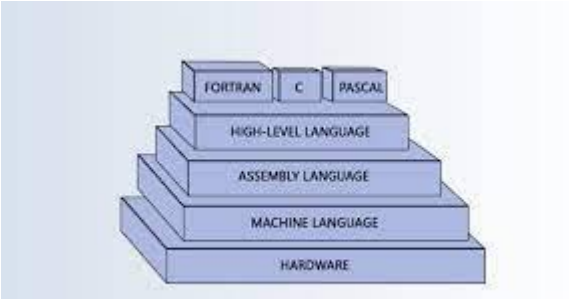


Programming Languages



Programming languages are broadly classified into three levels:

- machine languages,
- assembly languages,
- and high-level languages.



Machine language is the only programming language the CPU understands

Machine-language instructions are **binary-coded** and very low level

```
10110011 00011001
01111010 11010001 10010100
10011111 00011001
01011100 11010001 10010000
10111011 11010001 10010110
```

One machine instruction may transfer the contents of one memory location into a CPU register or add numbers in two registers

Thus, we **must provide many machine-language instructions to accomplish a simple task** such as finding the average of 20 numbers

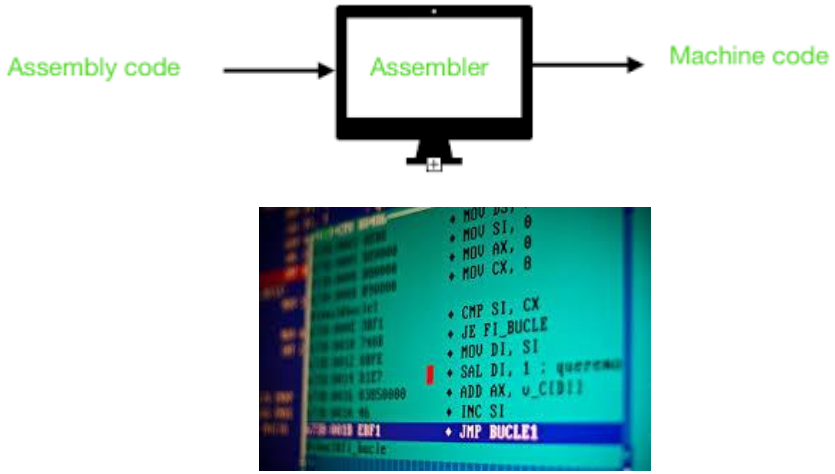
One level above machine language is ***assembly language***, which allows "higher-level" **symbolic programming**.

Instead of writing programs as a sequence of bits, assembly language allows programmers to write programs by **using symbolic operation codes**

For example, **instead of 10110011, we use MV** to move the contents of a memory cell into a register
We also can use **symbolic, or mnemonic, names for registers and memory cells**

A program written in assembly language might look like this:

```
MV    O,    SUM
MV    NUM,  AC
ADD   SUM,  AC
STO   SUM,  TOT
```

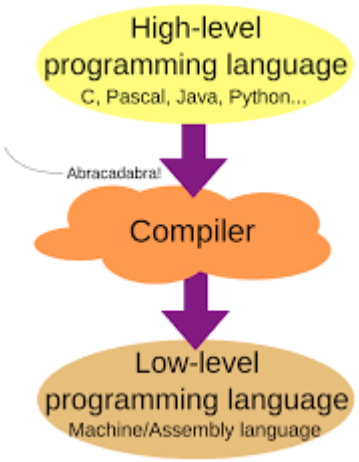


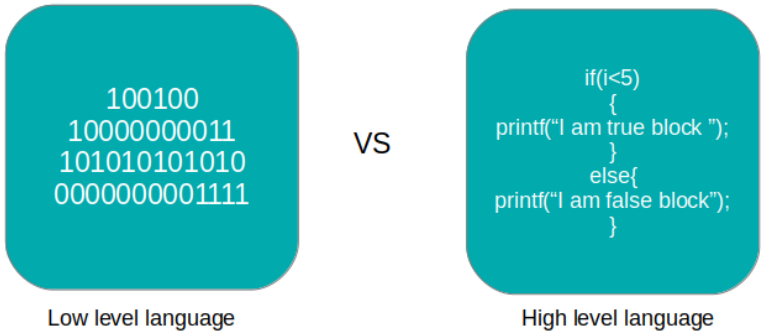
ASSEMBLER:

Since programs written in assembly language are not recognized by the CPU, we use an ***assembler*** to **translate programs written in assembly language into machine-language equivalents**

Compared to writing programs in machine language, writing programs in assembly language is **much faster, but not fast enough for writing complex programs**

High-level languages were developed to **enable programmers to write programs faster than when using assembly languages**

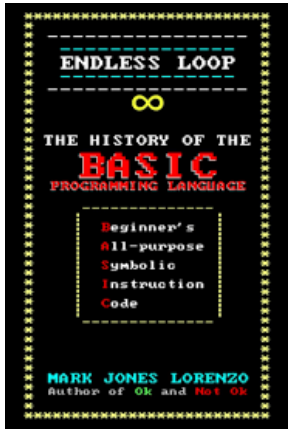




Since programs written in a high-level language are not recognized by the CPU, we must use a ***compiler*** to translate them to assembly language equivalents

Examples:

BASIC (Beginners All-purpose Symbolic Instructional Code) was developed specifically as an easy language for students to learn and use




```
10 INPUT "Please enter your name", A $
20 PRINT "Good day", A $
30 INPUT "How many stars do you want?"; S
35 S $ = ""
40 FOR I = 1 TO S
50 S $ = S $ + "*"
55 NEXT I
60 PRINT S $
70 INPUT "Do you want more stars?"; Q $
80 IF LEN (Q $) = 0 THEN GOTO 70
90 L $ = LEFT $ (Q $, 1)
100 IF 30 (L $ = "Y") OR (L $ = "y") THEN GOTO
110 PRINT "Goodbye";
120 FOR I = 1 TO 200
130 PRINT A $; "";
140 NEXT I
150 PRINT
```

FORTRAN and COBOL were developed in the late 1950s and early 1960s and are still in use.

FORTRAN (FORmula TRANslator), a programming language intended for ***mathematical computation***, allows programmers to express numerical equations directly as: X = (Y + Z) / 2

COBOL (COmmon Business-Oriented Language) is a programming language intended for business data processing applications.

The **programming language C** was developed in the early 1970s at **AT&T Bell Labs.**

The **C++ programming language** was developed as a successor of C in the early 1980s to add support for **object-oriented programming.**

Programming Paradigms

- **Imperative**: the ***programmer provides instructions*** to the machine how to change its state. Ex: C
- **Procedural**: also imperative. Allows ***splitting the instructions into procedures***. Ex: Fortran
- **Object-oriented**: *uses classes to pass messages to objects*, *facilitates reuse through components and inheritance*. Ex: Java, C++, Python
- **Declarative**: the programmer **expresses the logic of a computation**, but **not how to compute it**. Ex: Prolog, SQL
- **Functional**: the programmer *composes a program of short functions within a function*. Ex: Haskell
- **Logical**: A **declarative rule-based approach to problem-solving**, where a *program is specified as a set of facts and rules*. Ex: Prolog

Why OOP?

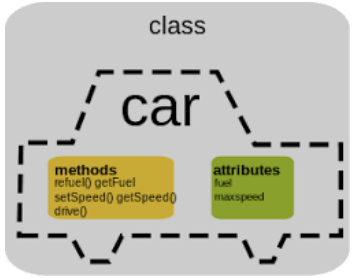
Object Oriented Programming (OOP) is a paradigm in which **real-world objects are each viewed as separate entities** having their **own state** which is *modified only by built in procedures, called **methods***.

Object-oriented programming helps us to

- de-couple parts of our programs from each other
- minimise redundancy
- reuse and extend existing code

But it needs more up-front planning (design!)

- so it is more suitable for long-term, large-scale project development



Object Oriented Programming Brief Background

Object-oriented programming is a style of programming gaining wider acceptance today

Although the concept of object-oriented programming is old (the first object-oriented programming language, **Simula**, was developed in the late 1960s), its significance wasn't realized until the early 1980s

Smalltalk, developed at **Xerox PARC**, is another well-known object-oriented programming language

The programming language we use in this book is **Java**, the newest object-oriented programming language, developed at **Sun Microsystems**

Why Java ?

Java is a programming language designed to be concurrent, class-based, and object-oriented as well as a computing platform first released by Sun Microsystems in 1995.

Plenty of applications and websites will not work unless you have Java installed.

Java has been tested, refined, extended, and proven by a dedicated community of Java developers, architects and enthusiasts.

Some interesting facts:

- 97% of Enterprise Desktops run Java.
- 89% of Desktops (or Computers) in the U.S. run Java.
- 9 Million Java Developers worldwide.
- 3 billion mobile phones run Java.
- 125 million TV devices run Java.

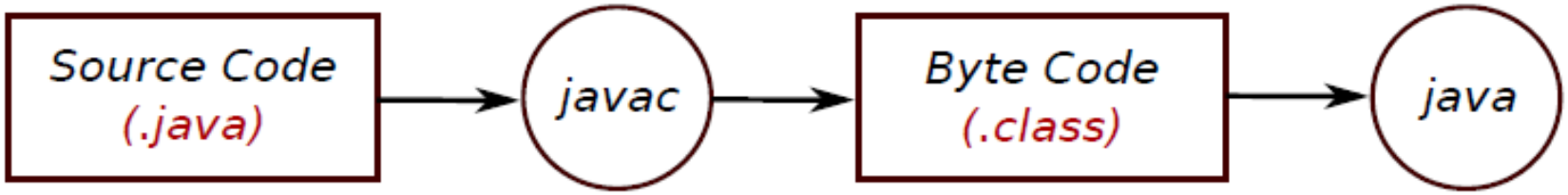
Note: Java and JavaScript are different languages.

Compiling Java

Java runs on a virtual machine (JVM).

1 Java program (source-code) is compiled and converted into bytecodes.

2 Bytecodes are platform independent instructions for the JVM to interpret and run the program.




thank you
shutterstock.com · 735649525

Plagiarism

1. Plagiarism is presenting someone else's work as your own. It is a violation of UCC Policy and there are strict and severe penalties.
2. You must read and comply with the UCC Policy on Plagiarism www.ucc.ie/en/exams/procedures-regulations/
3. The Policy applies to *all* work submitted, including software.
4. You can expect that your work will be checked for evidence of plagiarism or collusion.
5. In some circumstances it may be acceptable to reuse a small amount of work by others, but *only* if you provide explicit acknowledgement and justification.
6. If in doubt ask your module lecturer *prior* to submission. Better safe than sorry!