Software and programming Languages
Application Software

- Application software utilizes the capacities of a computer directly for a dedicated task.
- It is computer software designed to help the user to perform singular or multiple related specific tasks.

Examples:
- Enterprise software
- Accounting Software
- Graphics software
Two types of application software

(1) Generic (General Purpose) Software

- A generic software package is a package that can be put to a wide variety of uses.
- For example a spreadsheet package can be used for any task involving calculations or graph plotting.
- The most common generic software packages are:
  - Word Processor: Used to produce simple documents such as letters and essays.
Two types of application software

- Graphics Package: Used to produce pictures and diagrams.
- Spreadsheet: Used to perform calculations and draw graphs.
- Database: Used to store information so that it can be easily searched.
Two types of application software

(2) Application Specific Software

- An application specific package is produced to perform one specific task.
- For example a program written to produce invoices and manage stock levels for a garage.
  - Example: Accounting software
Available in variety of forms:
- Packaged software
- Freeware
- Shareware
(refer the previous note in chapter one)
The Evolution of Programming Languages
The Evolution of Programming Languages

- To build programs, people use languages that are similar to human language. The results are translated into machine code, which computers understand.
- Programming languages fall into three broad categories:
  - Machine languages
  - Assembly languages
  - Higher-level languages
The Evolution of Programming Languages

- Machine Language (First Generation)
  - Machine languages (first-generation languages) are the most basic type of computer languages, consisting of strings of numbers the computer's hardware can use.
Assembly Language (Second Generation)

• Assembly languages (second-generation languages) are only somewhat easier to work with than machine languages.
• To create programs in assembly language, developers use cryptic English-like phrases to represent strings of numbers.
• The code is then translated into object code, using a translator called an assembler.
The Evolution of Programming Languages

Assembly code

Assembler

Object code
Higher-Level Languages

Higher-level languages are more powerful than assembly language and allow the programmer to work in a more English-like environment.

Higher-level programming languages are divided into three "generations," each more powerful than the last:
- Third-generation languages
- Fourth-generation languages
- Fifth-generation languages
Third Generation Language

- Third-generation languages (3GLs) are the first to use true English-like phrasing, making them easier to use than previous languages.
- 3GLs are portable, meaning the object code created for one type of system can be translated for use on a different type of system.
- The following languages are 3GLs:
  - FORTAN
  - COBOL
  - BASIC
  - C
  - C++
  - Java
  - Pascal
Fourth Generation Language

- Fourth-generation languages (4GLs) are even easier to use than 3GLs.
- 4GLs may use a text-based environment (like a 3GL) or may allow the programmer to work in a visual environment, using graphical tools.
- The following languages are 4GLs:
  - Visual Basic (VB)
  - Visual Age
Fifth Generation Language

- A fifth generation programming language is a programming language designed to make the computer solve the problem of you.
- Fifth generation language are used mainly in artificial intelligence research.
- Prolog, OPS5 and Mercury are the best known fifth generation languages.
Commonly use programming languages

**Worldwide**, May 2016 compared to a year ago:

<table>
<thead>
<tr>
<th>Rank</th>
<th>Change</th>
<th>Language</th>
<th>Share</th>
<th>Trend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Java</td>
<td>24.0%</td>
<td>-0.2%</td>
</tr>
<tr>
<td>2</td>
<td>↑</td>
<td>Python</td>
<td>12.4%</td>
<td>+1.8%</td>
</tr>
<tr>
<td>3</td>
<td>↓</td>
<td>PHP</td>
<td>10.6%</td>
<td>-0.8%</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>C#</td>
<td>8.9%</td>
<td>-0.4%</td>
</tr>
<tr>
<td>5</td>
<td>↑↑↑</td>
<td>Javascript</td>
<td>7.5%</td>
<td>+0.5%</td>
</tr>
<tr>
<td>6</td>
<td>↓</td>
<td>C++</td>
<td>7.4%</td>
<td>-0.3%</td>
</tr>
<tr>
<td>7</td>
<td>↓</td>
<td>C</td>
<td>7.2%</td>
<td>+0.1%</td>
</tr>
</tbody>
</table>
What is object oriented programming?

- A programming approach which uses objects
- An object is a software component which has properties and behaviour
- When a program runs objects are created and work together to perform the program’s tasks
- Most modern programming languages support object orientation
  - C#, Java, VB.NET, C++, PHP, etc.
Where should we use it?

- Object-orientation offers some key benefits:
  - Code re-use – DRY principle
  - Ability to model real-world environments
  - Understand ability

- Many kinds of software application can benefit from object oriented approach

- GUI applications, web applications, games, etc.
An entity, or *thing*, is represented as an **object** in the program

- e.g. an object representing an Employee in a company

- Objects have **attributes** to represent **state** of object, e.g. *name*, *location* of an Employee

- Objects have **methods** to define the actions, or **behaviour**, which object can perform, e.g. an Employee could *record that he or she worked some overtime hours*
Thank You..