EDIDIONG IME-ESSIEN

17/SCI01/041

CSC302

1.(i) Unstructured Programming Language -: they’re hard to read and it allows code duplication examples are; FOCA, MUMPS, COBOL etc.

(ii) structured Programming Language :- this programming language allows a programmer to divide the whole program into smaller units examples are; Ruby, C++, Java.

(iii) Modula Programming Language:- here less code has to be written and its pretty easy to comprehend examples are; COBOL, Morpho, Fortran, Erlang.

(iv) Object oriented Programming Language:- this programming language is more suitable for large objects and it has the feature of a memory management examples are; Python, Delphi, C#.

(v) Activity Programming Language

(vi) Aspect Programming Language:- this programming language provides privacy and reuse of codes and also allows programmers write modules examples are; Mix juice, Aspect Werka, JAC .

2. **Chronological Order of Programming Language**

**1840 – Analytical Engine Code**
The Analytical Engine was a theoretical (i.e., never built) mechanical general-purpose computer, created by British mathematician Charles Babbage. Ada Lovelace came across the idea, and created some code for the Analytical Engine. That’s why she’s considered the first programmer ever.

**1943 – ENIAC Coding System**
The ENIAC is regarded as the first electronic general-purpose computer. Both the computer and its coding were created by John von Neumann, John Mauchly, and J. Presper Eckert.

**1949 – Brief Code (Later Short Code)**
Initially proposed by John Mauchly, it was one of the first attempts of an assembly language.

**1954 – Fortran**
One of the most popular high-level programming languages. It was created by John W. Backus at IBM as an easier alternative to programming in assembly.

**1958 – LISP**
Created by John McCarthy, one of the pioneers of AI as well.

**1959 – COBOL**
The name stands for COmmon Business-Oriented Language, as the language was aimed mainly at banks, financial institutions and companies.

**1964 – BASIC**
Beginner’s All-purpose Symbolic Instruction Code, a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use.

**1970 – Pascal**
Pascal is an influential imperative and procedural programming language, designed in 1968–1969 and published in 1970 by Niklaus Wirth as a small and efficient language intended to encourage good programming practices using structured programming and data structuring.

**1972 – Smalltalk**
The language that started to inflate the popularity of object-oriented programming.

**1972 – C**
Created by Dennis Ritchie and Ken Thompson at the AT&T Bell Labs. It’s simplicity and efficiency made it one of the most popular languages around the world.

**1972 – SQL**
Created at IBM, it became the standard for dealing with databases.

**1983 – C++**
Originally named “C With Classes”, it brought object-orientation to C (which is technically a subset of C++).

**1987 – Perl**
Perl is a family of high-level, general-purpose, interpreted, dynamic programming languages.

**1991 – Python**
A high-level language that emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C.

**1995 – Java**
Java is the most popular object-oriented programming language around, and it was created to have as few implementation dependencies as possible. It’s widely used in commercial and business applications.

3. **Modular programming** (also called "top-down design" and "stepwise refinement") is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality. **Modular programming** just implies you have these two (or more) modules, but says nothing of how they achieve what they achieve. The modules can use object-oriented approaches or not at all and use procedural C-style programming. The way you described modular programming via classes is just a way of separating modules. You can separate them as classes, or you can separate them as functions across multiple compilation units, for example. It's your choice.

**Object-oriented programming** implies that your program is, well, **oriented towards objects**. It says nothing about modules within your application but demands that logical pieces that represent some ideas within the application are modeled via classes and objects.

As such, the two approaches can be used together, and when you decide to be modular, the object-oriented choice usually imposes on you that these modules are defined via classes and their relationships. An object-oriented program usually contains different types of objects, each corresponding to a particular kind of complex data to manage, or perhaps to a real-world object or concept such as a bank account, a hockey player, or a bulldozer.