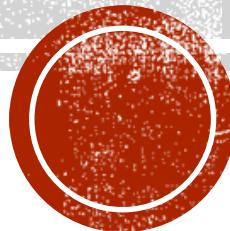




OBJECT ORIENTED PARADIGM

Lesson # 01

Lasanthi De Silva





PROGRAMMING INNOVATIONS

Improved software development processes

The Unified Modeling Language (UML)

Object-oriented programming (OOP)

PROCEDURAL LANGUAGES

- The problem is viewed as a sequence of things.
- A list of instructions.
- Example: C, Pascal, FORTRAN
- Each statement in the language tells the computer to do something:
 - Get some input,
 - add these numbers,
 - divide by six, and
 - display that output.

What happens when the Program becomes Larger?

A red circle with a white border and a white number '4' inside.

4

ACTIVITY

Write a simple C program that takes two integers as input, adds them, and then prints the result.

```
#include <stdio.h>

int main() {
    // Declare variables
    int num1, num2, sum;

    // Input
    printf("Enter first number: ");
    scanf("%d", &num1);

    printf("Enter second number: ");
    scanf("%d", &num2);

    // Add the numbers
    sum = num1 + num2;

    // Output
    printf("Sum: %d\n", sum);

    return 0;
}
```

PROCEDURAL LANGUAGES

- A program in a procedural language is a list of instructions.
- Example: C, Pascal, FORTRAN
- Each statement in the language tells the computer to do something:
 - Get some input,
 - add these numbers,
 - divide by six, and
 - display that output.
- A very small program
 - The programmer creates the list of instructions, and the computer carries them out.
 - No other organizing principle (a paradigm) is needed.

What happens when the Program becomes Larger?

DIVISION INTO FUNCTIONS

- When the program becomes Complex
 - The need arises to comprehend
- Also known as a subroutine, a subprogram, or a procedure
- Each function has a clearly defined purpose and a clearly defined interface to the other functions in the program
- Functions can be further grouped together to form Modules
- A grouping of components that execute lists of instructions
- Structured programming: Dividing a program into functions and modules

```
1 #include <stdio.h>
2
3 // Function prototype
4 int addNumbers(int a, int b);
5
6 int main() {
7     // Declare variables
8     int num1, num2, sum;
9
10    // Input
11    printf("Enter first number: ");
12    scanf("%d", &num1);
13
14    printf("Enter second number: ");
15    scanf("%d", &num2);
16
17    // Function call to add numbers
18    sum = addNumbers(num1, num2);
19
20    // Output
21    printf("Sum: %d\n", sum);
22
23    return 0;
24}
25
26 // Function definition to add two numbers
27 int addNumbers(int a, int b) {
28     return a + b;
29 }
```

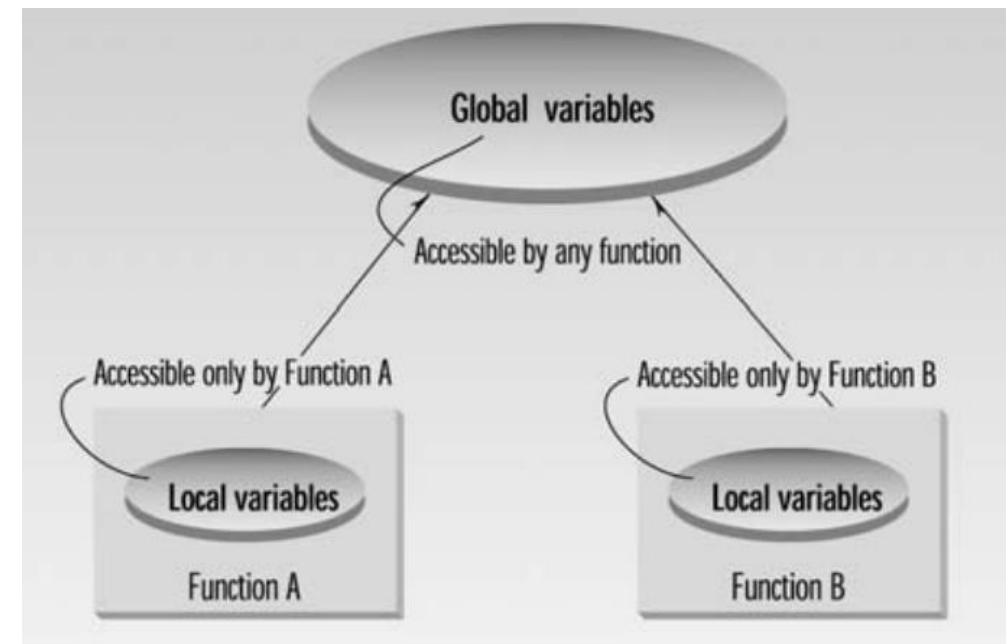
PROGRAMMING PARADIGMS

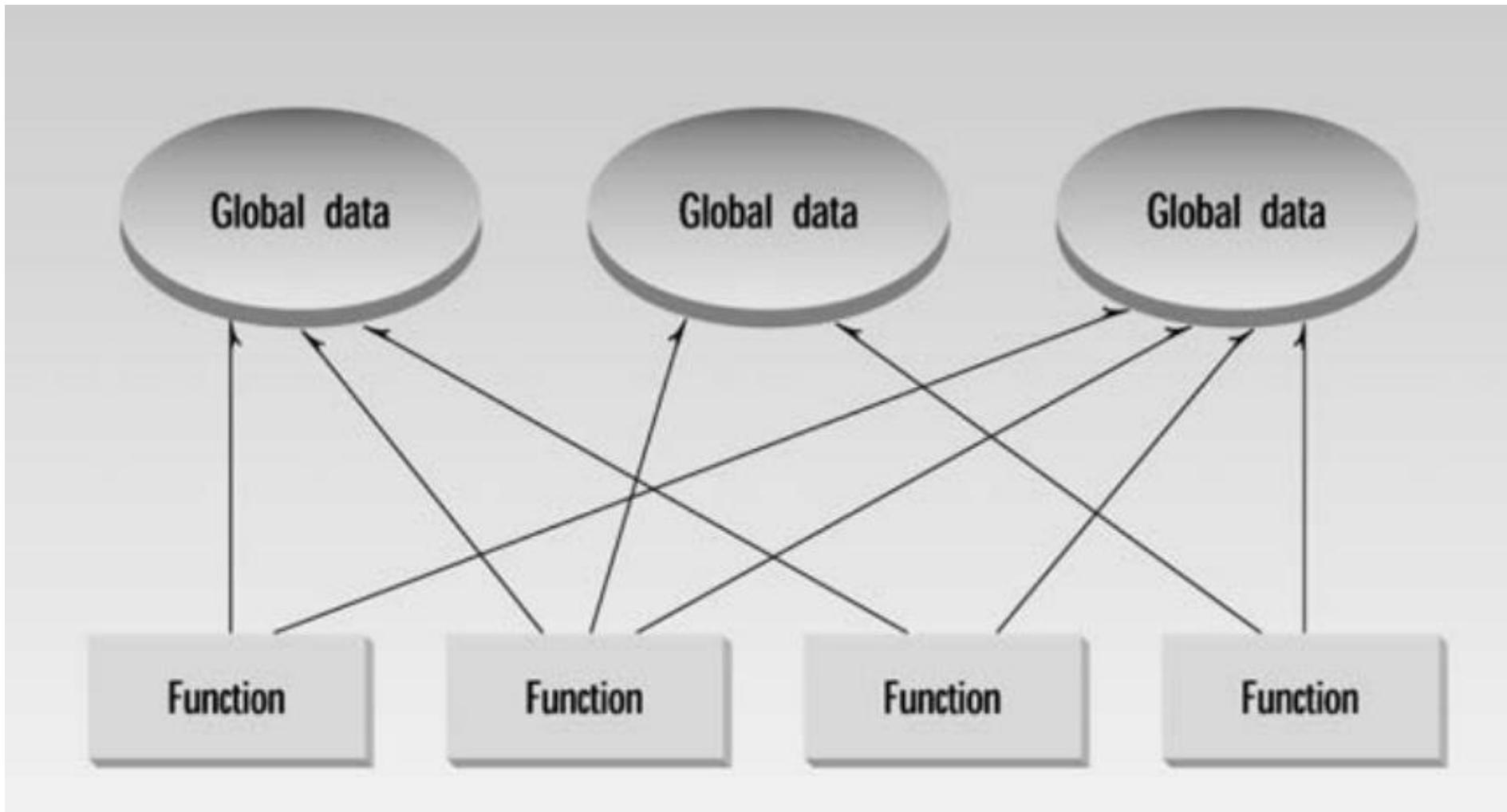
- The way you write the program
- Procedural vs Object Oriented Programming
- Procedural Programming
 - A programming model derived from structured programming
 - Divide the program into series of steps.
 - Called based on the concept of procedure calls.
 - Procedures - routines, subroutines or functions (consist of a series of computational steps to be carried out).

LANGUAGES: FORTRAN, ALGOL, COBOL, BASIC, Pascal and C.

WEAKNESSES IN THE PROCEDURAL PARADIGM

1. Functions have unrestricted access to global data.
2. Unrelated functions and data, the basis of the procedural paradigm, provide a poor model of the real world.





WEAKNESSES IN THE PROCEDURAL PARADIGM

- A large number of connections makes the program more complex
- A program's structure is difficult to conceptualize
- It makes the program difficult to modify
- A change made in a global data item may necessitate rewriting all the functions that access that item
- Difficulty in modeling Real-World Problems

Complex real-world objects have both attributes and behavior.

PROGRAMMING PARADIGMS

Object Oriented Programming

- An approach that provides a way of modularizing programs by creating partitioned memory areas for both data and functions that can be used as templates for creating copies of such modules on demand
- Combine into a single unit both data and the functions that operate on that data.
- A unit called an object.

LANGUAGES: Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Objective-C, Dart, Swift, Scala.

14

ACTIVITY: COMPARE & CONTRAST

Procedural vs Object-Oriented Programming

PROCEDURAL ORIENTED PROGRAMMING

In procedural programming, program is divided into small parts called **functions**.

Procedural programming follows **top down approach**.

There is no access specifier in procedural programming.

Adding new data and function is not easy.

Procedural programming does not have any proper way for hiding data so it is **less secure**.

In procedural programming, overloading is not possible.

In procedural programming, function is more important than data.

Procedural programming is based on **unreal world**.

Example: C, FORTRAN, Pascal, Basic etc.

OBJECT ORIENTED PROGRAMMING

In object-oriented programming, program is divided into small parts called **objects**.

Object oriented programming follows **bottom up approach**.

Object oriented programming have access specifiers like private, public, protected etc.

Adding new data and function is easy.

Object oriented programming provides data hiding so it is **more secure**.

Overloading is possible in object-oriented programming.

In object-oriented programming, data is more important than function.

Object oriented programming is based on **real world**.

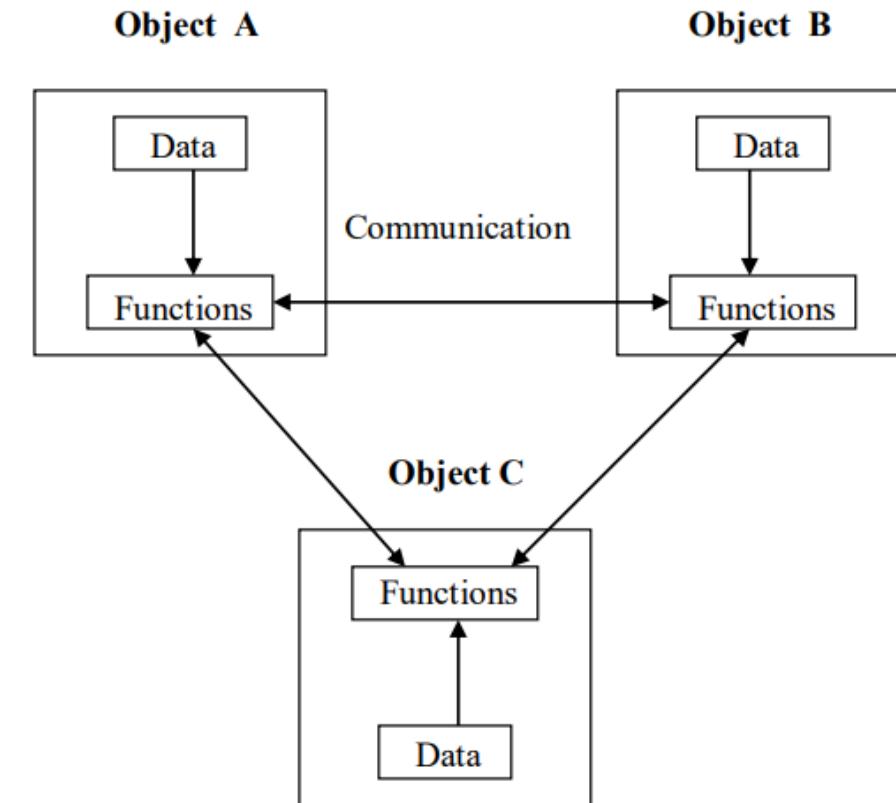
Example: C++, Java, Python, C# etc.

ADVANTAGES OF OOP

- Thinking in terms of Objects
- Faster and easier to execute
- Provides a clear structure for the programs
- Easier to understand
- Makes the code easier to maintain, modify and debug
- Makes it possible to create reusable applications with less code and shorter development time.

INTRODUCTION TO OOP

- Using a modular, object-oriented design-and implementation approach can make software development groups much more productive than was possible with earlier techniques.



INTRODUCTION OF OOP

- Refers to languages that use objects in programming.
- Create many *software objects* that model *things* in the real world.
- Any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating).
- Implement real-world entities like inheritance, hiding, polymorphism, etc
- Bind together the data and the functions that operate on entities so that no other part of the code can access this data except that function.

OOPs Concepts:

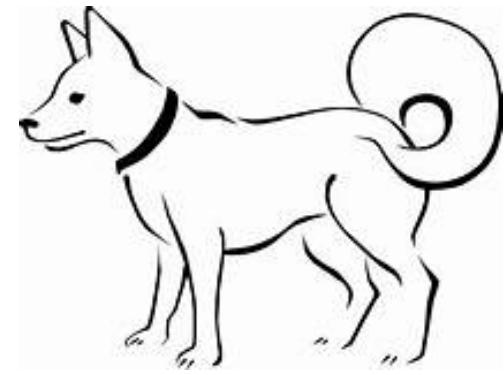
Class, Objects, Data Abstraction, Encapsulation, Inheritance, Polymorphism.



SCS1310



INTRODUCTION TO OOP CONCEPTS

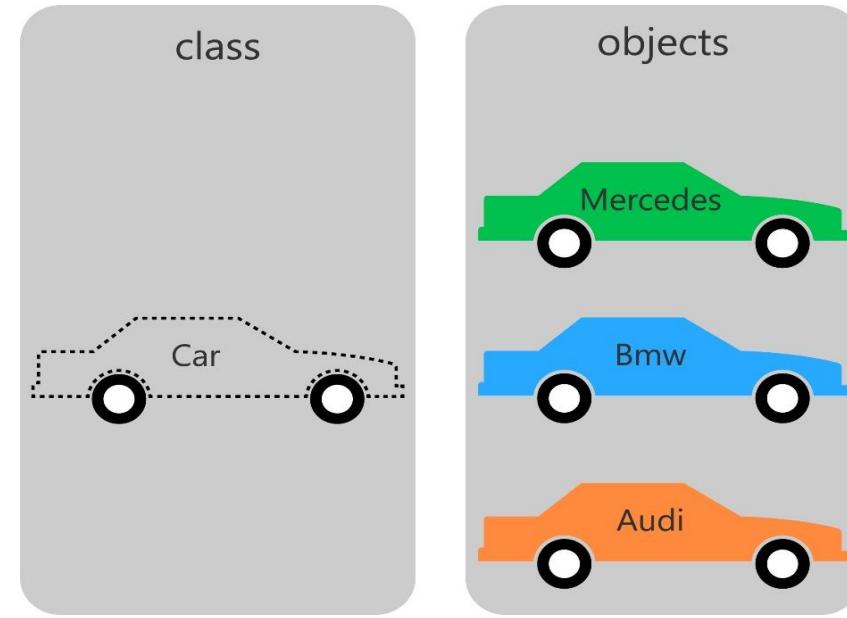


OOP CONCEPTS: OBJECT

- Basic unit of Object-Oriented Programming
- Represents the real-life entities
- An instance of a Class.
- Has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

Example: “My Car”

- Characteristics like color, Manufactured year, price, Drive, and Break.



EXAMPLES

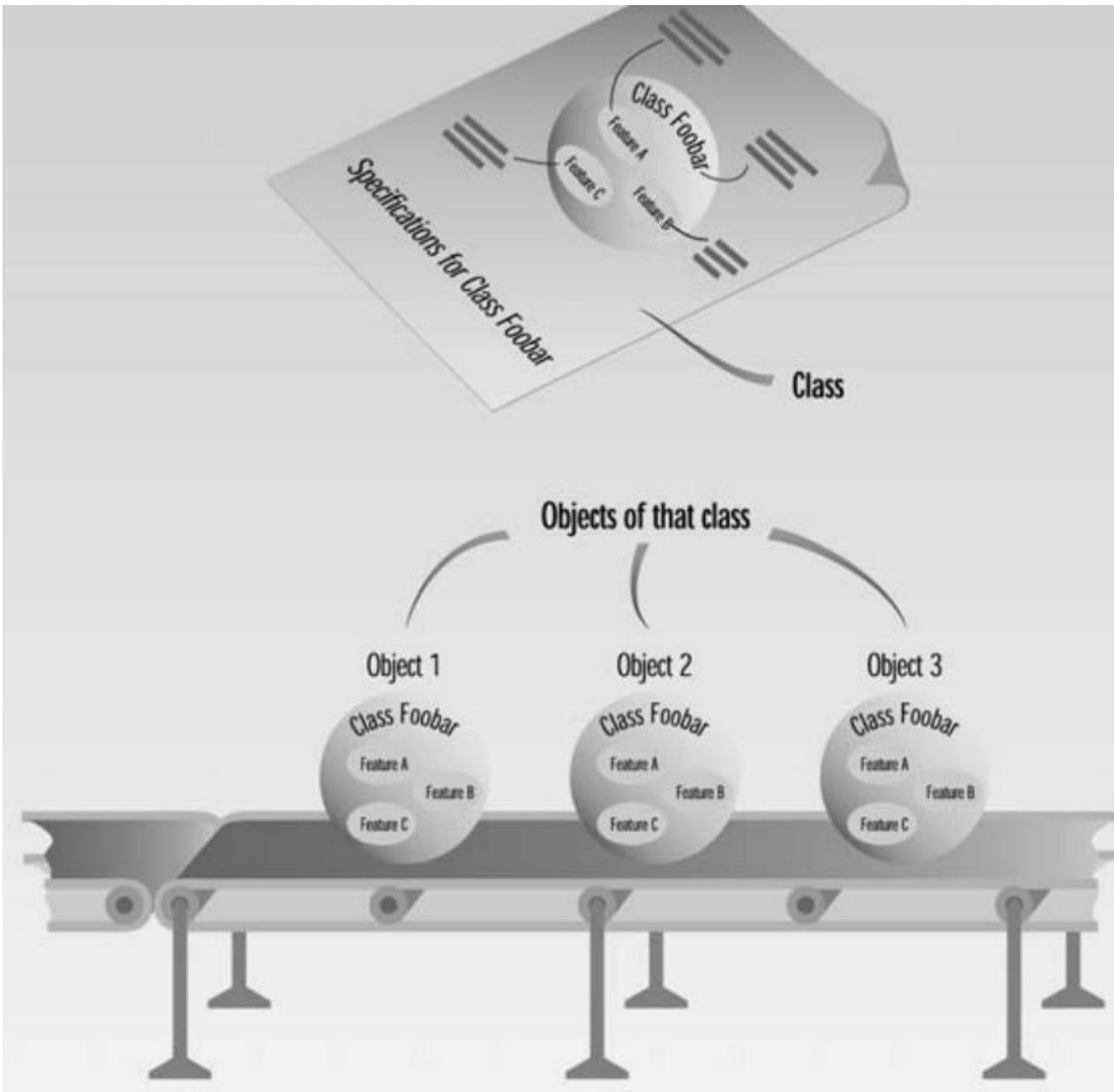
- Physical objects
 - Automobiles in a traffic-flow simulation
 - Electrical components in a circuit-design program
 - Countries in an economics model
 - Aircraft in an air traffic control system
- Elements of the computer-user environment
 - Windows
 - Menus
 - Graphics objects (lines, rectangles, circles)
 - The mouse, keyboard, disk drives, printer
- Data-storage constructs
 - Customized arrays
 - Stacks
 - Linked lists
 - Binary trees
- Human entities
 - Employees
 - Students
 - Customers
 - Salespeople
- Collections of data
 - An inventory
 - A personnel file
 - A dictionary
 - A table of the latitudes and longitudes of world cities
- User-defined data types
 - Time
 - Angles
 - Complex numbers
 - Points on the plane

OOP CONCEPTS: CLASS

- A blueprint for an object.
- A user-defined data type.
- Consists of data members and member functions.
- Accessed and used by creating an instance of the class.
- Represents the set of properties or methods that are common to all objects of one type.

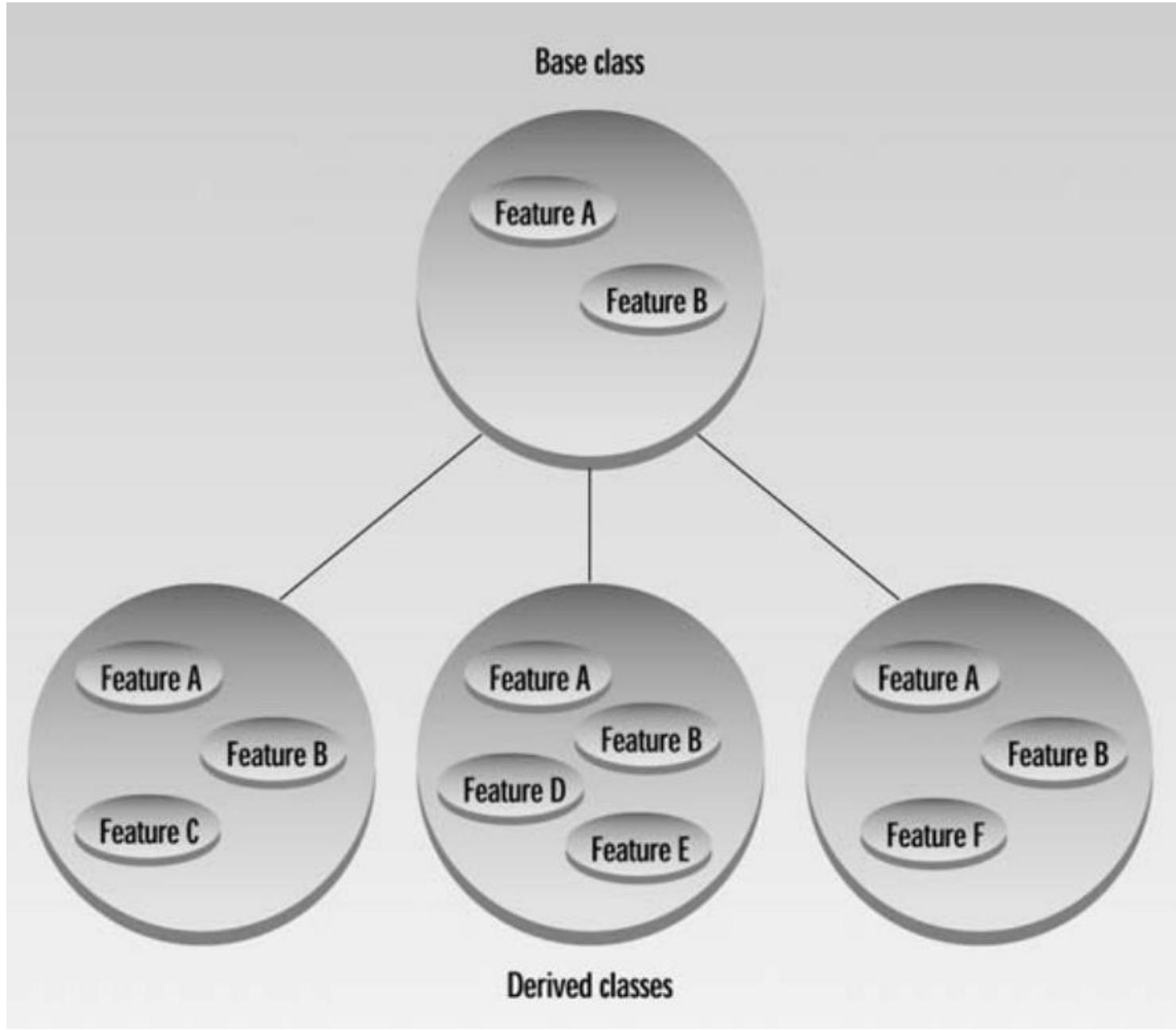
Example: Car

- Cars have different names and brands, but all share some common properties (4 wheels, Speed Limit, Mileage range, etc)



OOP CONCEPTS: INHERITANCE

- Classes divided into subclasses.
 - The Animals can be divided into mammals, amphibians, insects, birds, etc.
 - The vehicles can be divided into cars, trucks, buses, motorcycles, etc.
- A new class of object derives properties and characteristics from another class.
- It inherits properties from other classes.
- No need to write all the properties and functions again and again, as these can be inherited from another class that possesses it.
- Allows the user to reuse the code whenever possible and reduce its redundancy.



OOP CONCEPTS: DATA ABSTRACTION

- Provide only essential information about the data to the outside world.
- Hide the background details or implementation.

Example: A man driving the car

- Knows by pressing the accelerators will increase the speed/applying brakes will stop the car BUT NOT how.

OOP CONCEPTS: ENCAPSULATION

- Wrapping up of data under a single unit.
- An object's attributes and member functions are intimately related.
- Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented.
- Implementation details are *hidden* within the objects themselves.
- This is also known as **information hiding**.

OOP CONCEPTS: POLYMORPHISM

- Polymorphism means having many forms.
- Ability of a message to be displayed in more than one form.



OOP USING C++

General-purpose programming language developed as an enhancement of the C language to include object-oriented paradigm.

FEATURES OF C++

Object-Oriented Programming

- C++ is an Object-Oriented Programming Language.
- C is a procedural programming language.
- It can create/destroy objects while programming.

Machine Independent/Platform Dependent

- C++ executable is not platform-independent
 - Compiled programs on Linux will not run on Windows
- However they are machine independent.
 - Source code can be ported from one machine to another.

FEATURES OF C++

SIMPLE Language

- Programs can be broken down into logical units and parts.
- Has rich library support, and a variety of data-types.

Compiler Based

- Compiled and runs the executable file.

Case-sensitive

- **cin/Cin**

Dynamic Memory Allocation

- Allows to allocate the memory of a variable or an array in run time.

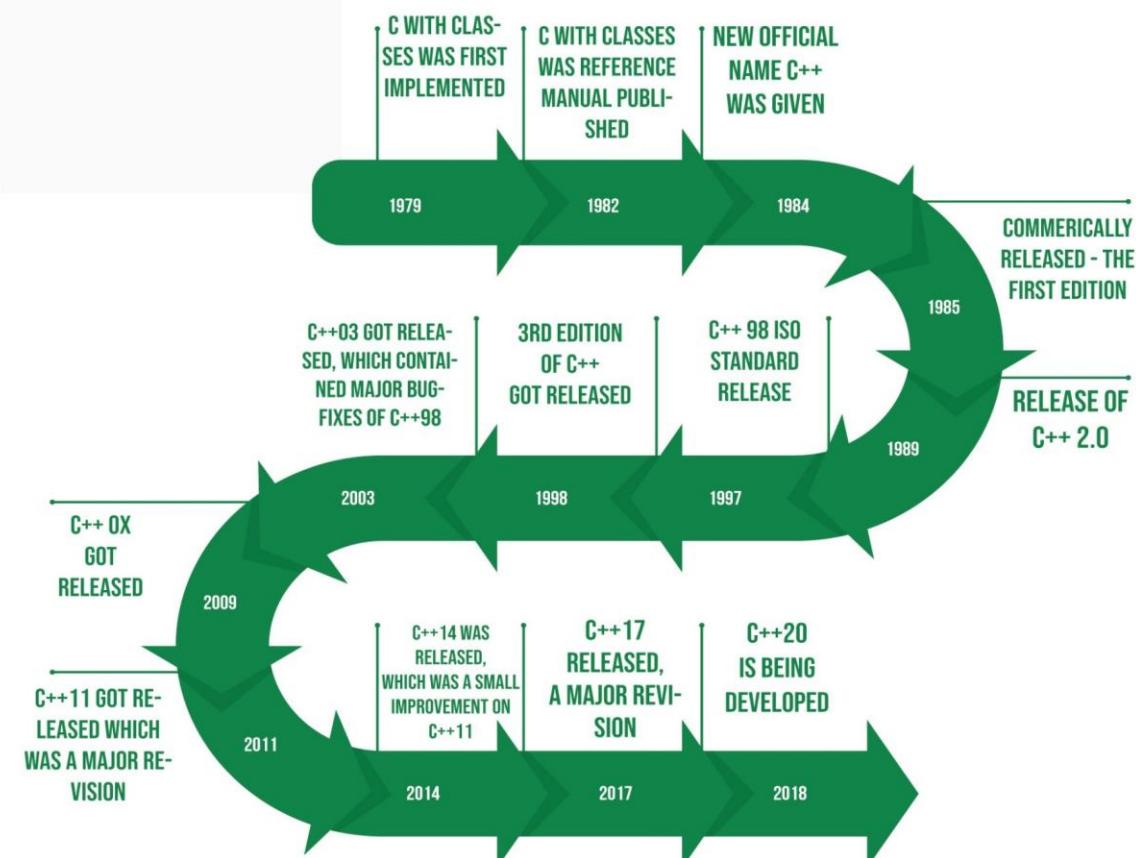
32

A BRIEF HISTORY

C++

SCS1310

History of C++



A BRIEF HISTORY TO C++

- Developed by Bjarne Stroustrup, as an extension to the C language (**1979**).
- Danish Computer Scientist at Bell Telephone Laboratories (Nokia Bell Labs) in New Jersey
- Simula (Used for simulations) : primary language to support the object-oriented programming- too slow for practice & practical use.
- Worked on “**C with Classes**” to get advanced object-oriented programming, into the C language
- In 1983, the name Change from C with categories to C++.
 - The ++ operator for incrementing a variable
- The language was updated 3 major times in 2011, 2014, and 2017 to C++11, C++14, and C++17.

35

C++ BASICS

Getting Familiar

SCS1310

GETTING STARTED: C++

- An editor to write C++ code
- A compiler to translate the C++ code into a language the computer understands

IDEs (Integrated Development Environment) to edit AND compile

- Eclipse
- Visual Studio
- **Dev C++**

EXERCISE 1

Print “Hello World”

```
#include <iostream>
int main()
{
    std::cout<<"Hello World";
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"Hello World";
    return 0;
}
```

EXERCISE 2

Print in multiple lines

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"Hello World\n";
    cout<<"This is my first C++ code\n";
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"Hello World"<<endl;
    cout<<"This is my first C++code"<<endl;
    return 0;
}
```

EXERCISE 3

```
#include <iostream>
using namespace std;
int main(){
    int varNum = 456;
    double varFloatNum = 65.459;
    char varChara = 'X';
    string varText = "Hello world";
    bool varBoolean = true;
    cout<<varNum<<endl;
    cout<<varFloatNum<<endl;
    cout<<varChara<<endl;
    cout<<varText<<endl;
    cout<<varBoolean<<endl;
}
```

Declaring Variables

Syntax:

type variable = value;

EXERCISE 4

User Inputs

```
#include <iostream>
using namespace std;
int main()
{
    int x;
    int y;
    cout<<"Enter X = ";
    cin>>x;
    cout<<"Enter Y = ";
    cin>>y;
    cout<<"SUM of X and Y is "<<x+y;
}
```