

Chapter One: Introduction

1.1 Chapter objectives

By the end of this, learners should be able to:

- i). Understand what an Object-oriented programming is
- ii). Differentiate between OOP and procedural/structured programming language
- iii). Understand the core basic features of OOP languages
- iv). Understand the benefits associated with OOP

1.2 Introduction to OOP

Object oriented programming (OOP) is a fairly new way to approach the task of programming aimed at dealing with the limitations associated with structural programming.

It allows programmers to adopt best ideas from structured programming as well as incorporate new concepts.

OOP is a programming methodology that helps to organize complex

programs through use of OOP related features i.e., **inheritance**, **abstraction**, **encapsulation** and **polymorphism**.

Email: Kioko@cuea.edu / **Mobile No:** ++254-725-695-782

© By Edward Kioko. All rights reserved for this module.

OOP overtakes the so called procedural/structured programming languages like COBOL, Pascal or C, that have been around since the 1960s.

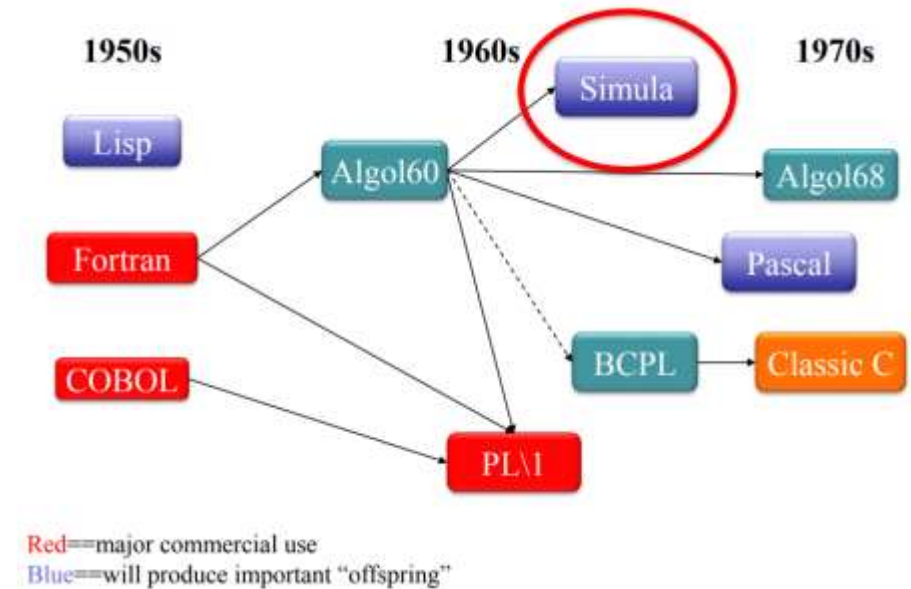


Fig 1.0-Early programming languages

The following is an illustration showing modern programming languages

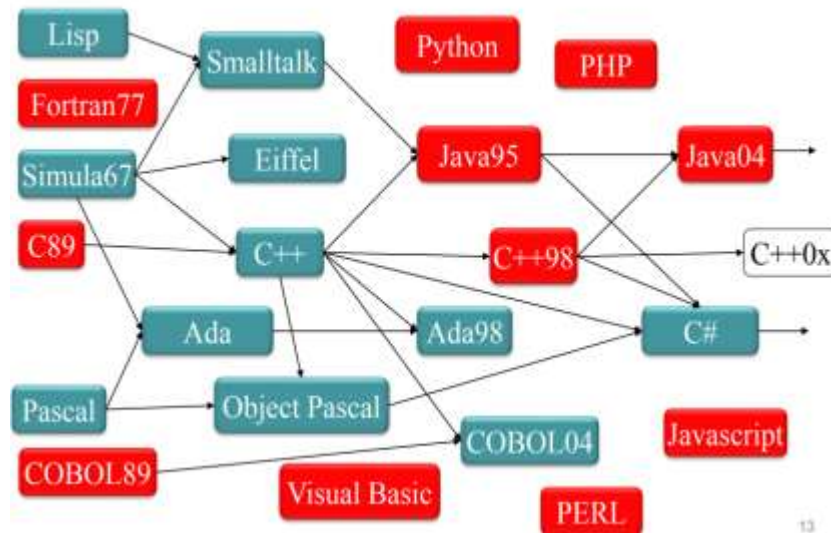


Fig 1.1-Modern programming Languages

OOP is a programming philosophy whose programs are written on basis of “objects”. The real world can be “accurately” described as a collection of objects that interact

In OOP, the problem is usually decomposed into related subgroups, where each subgroup becomes a self-contained

object that contains its own instructions and data that relate to that object.

In OOP, we do not only define the data type of a data structure, but also the types of operations (methods/functions) to be applied to the data structure. Data structure becomes an object including both data and functions (methods) in one unit.

1.3 Procedure oriented programming

This is also known as structured programming

The main emphasis is on doing things i.e., algorithms. Ideally large programs are divided into smaller programs known as functions/procedures/subroutines. The functions will share global data and these data will normally move freely around the system from one function to another with the functions transforming the data from one form to another. Procedure oriented programming employs top-down approach for program design

Though structured programming has yielded excellent results when applied to moderately complex programs, it fails when a program complexity increases.

1.4 Object Oriented Programming Paradigm

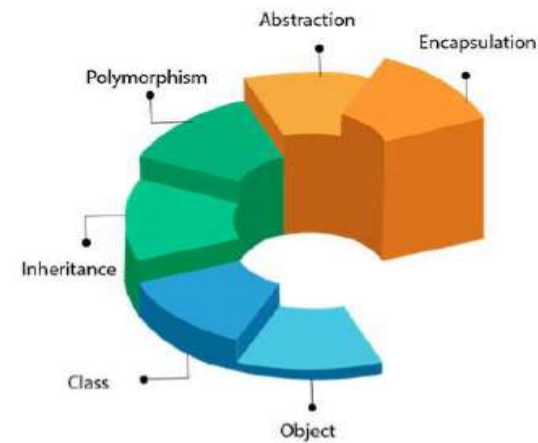
Emphasis is kept on data rather procedure. Large and complex programs are divided into what are known as objects. Data structures are designed in such a way that they characterize the objects. Objects that operate on the data of objects are tied together in the data structures.

Data is always hidden and cannot be accessed by external functions. If objects need to communicate with one another, they will do so through message passing. New data and functions can be added wherever necessary. OOP follows the bottom-up approach of program design.

1.5 OOP Concepts

In structural programming, structures will contain only data, and programmers separately create functions to act on them. Objects contain both data and functions (methods) to operate upon them. OOP is associated with four major concepts; ***inheritance, polymorphism, abstraction and encapsulation***

Other characteristics associated with OOP include ***classes, objects and message passing***



- **Objects** -> real world entities
- **Classes** -> templates or blueprints
- **Methods** -> Object actions
- **Data abstraction** -> abstraction controls
- **Encapsulation** -> binding code & data
- **Inheritance** -> Acquiring other object properties
- **Polymorphism** -> many forms

Fig 1.3-OOP concepts

1.5.1 Objects

Email: Kioko@cuea.edu / **Mobile No:** ++254-725-695-782

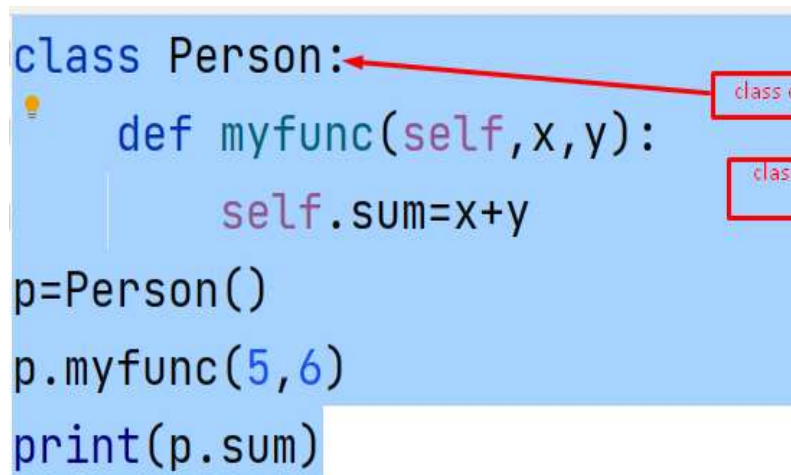
© By Edward Kioko. All rights reserved for this module.

1.5.1 Classes

It's a user defined data type (ADT) which encapsulates related data into a single entity. It defines how an object of this type will look and behave. It is simply a prototype, idea, and blueprint for creating objects

A class has three data elements; its name, attributes/properties & member functions. These elements together are known as fields.

An example of class definition in python is shown below



```
class Person:
    def myfunc(self, x, y):
        self.sum = x + y
p = Person()
p.myfunc(5, 6)
print(p.sum)
```

The image shows a Python code snippet on a light blue background. The code defines a class 'Person' with a method 'myfunc' that takes 'self', 'x', and 'y' as arguments and sets 'self.sum' to 'x + y'. It then creates an instance 'p' of the 'Person' class and calls 'p.myfunc(5, 6)', finally printing 'p.sum'. A red arrow points from the text 'class definition' to the 'class Person:' line. Another red arrow points from the text 'object creation' to the 'p = Person()' line. There are also two small red boxes, one containing 'class d' and the other 'class', with arrows pointing to the 'class' keyword in the class definition and the 'Person' argument in the object creation respectively.

A class doesn't occupy memory hence it can be manipulated. A class is a data type (Abstract Data Type)

1.5.2 Objects

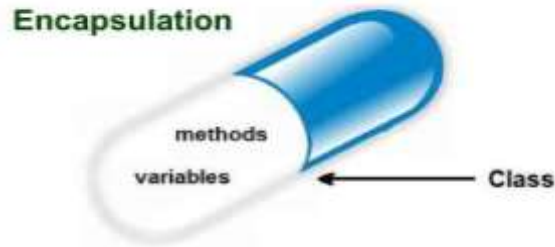
In OOP, we refer to object as an instance of a class. These are basic run-time entities in an OOP system. Objects can be *a place, thing, system device, organization, event, structure* e.t.c.

In the above example, p is an object of type class person. That means the object can have access to method **myfunc ()** and attribute **sum** defined in the class

An object gives life to class. An object occupies memory as opposed to a class hence it can be manipulated

1.5.3 encapsulation

It's an OOP mechanism that binds code and data together and keeps them safe from outside interference and misuse. A good way to think about encapsulation is to think about capsulation as a protective wrapper that prevents code and data from being accessed outside the wrapper. It is achieved by using a class with the class acting a container/cell or capsule



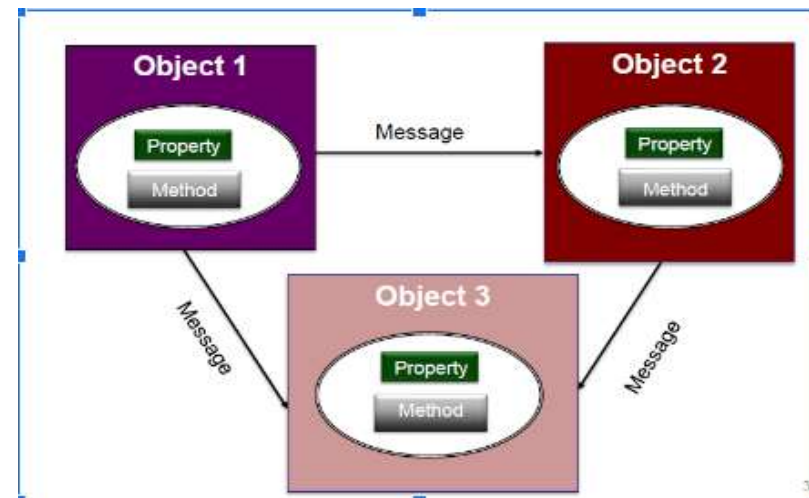
This feature of Oop allows a class to change its internal implementation without hurting the overall functioning of the system. It also hides how a class does its operations while allowing requesting its operations

Benefits associated with encapsulation

1. Ensures that structural changes remain local:
 - Changing the class internals does not affect any code outside of the class
 - Changing methods' implementation does not reflect the clients using them
2. Encapsulation allows adding some logic when accessing client's data E.g., validation on modifying a property value
3. Hiding implementation details reduces complexity (easier maintenance)

1.5.4 Message passing

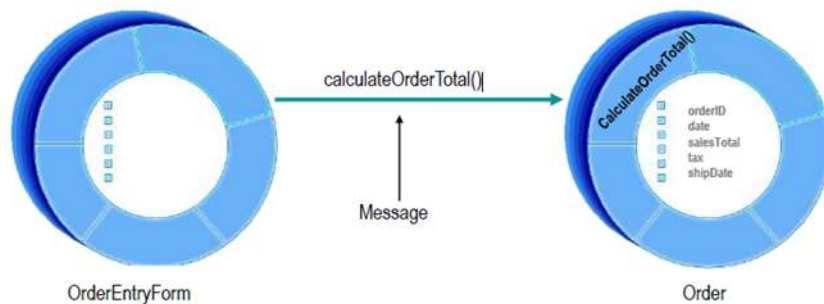
Objects are useless unless they can collaborate together to solve a problem. Each object is responsible for its own behavior and status. However, no one object can carry out every responsibility on its own. Objects interact with each other through messages, a mechanism we refer to as message passing in OOP



Let's look at the following example

The OrderEntryForm wants Order to calculate the total NOK value for the order.

The class Order has the responsibility to calculate the total NOK value. That means object OrderEntryForm will send a message to order which in turn will compute the total and return it to OrderEntryForm as shown below



1.5.5 Inheritance

It's a concept of OOP which allows classes to inherit properties and methods from another class.

Inheritance allows child/derived classes to inherit the characteristics of existing parent/base/super class i.e. attributes

(fields and properties) & Operations (methods)

Child class can extend the parent class by

1. adding new fields and/or methods
2. Redefine methods (modify existing behavior)

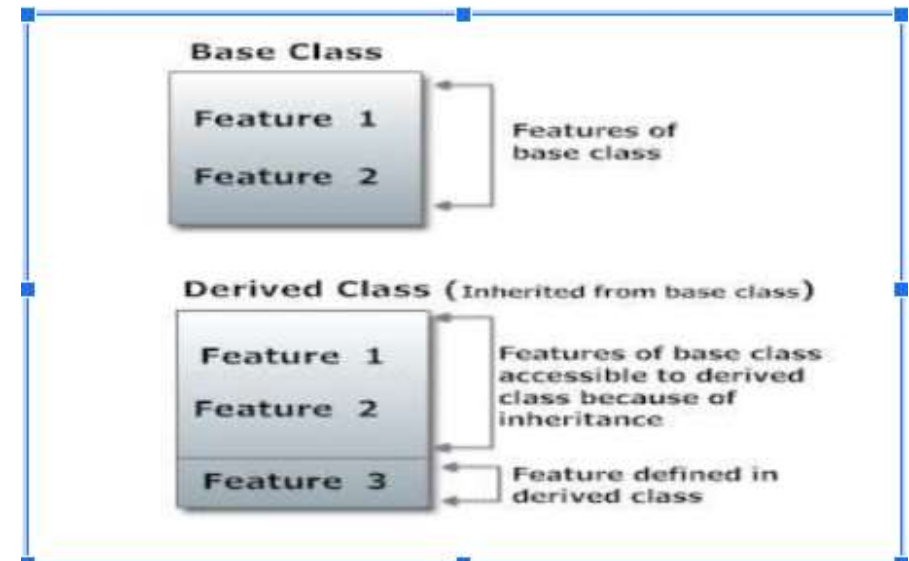
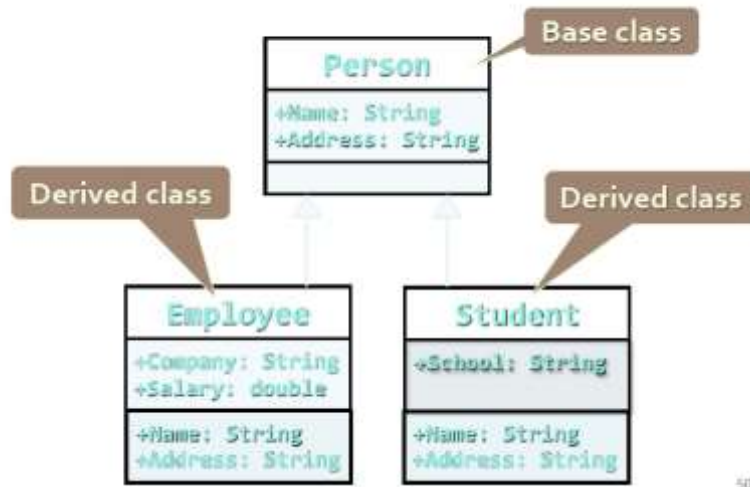
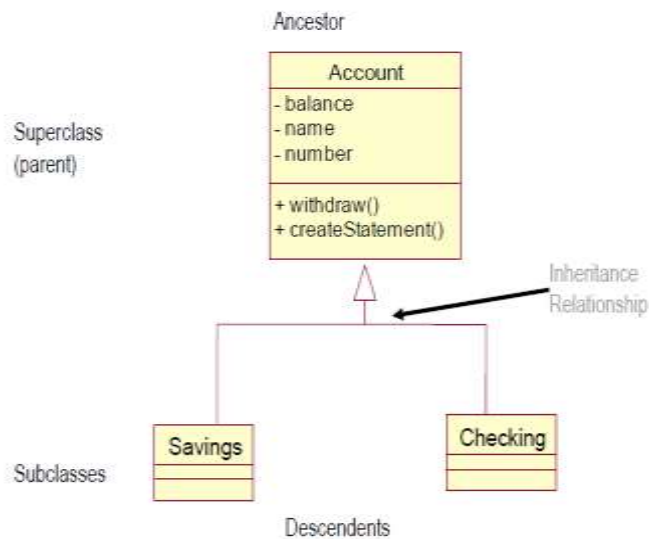


Fig 1.4: Typical illustration of inheritance



Below is an example of inheritance for a bank operation



1.5.6 Polymorphism

Encapsulation, Inheritance, and Abstraction concepts are very related to Polymorphism. Poly morphism means the “ability of objects of different types to respond to functions of the same form”.

The user does not have to know the exact type of the object in advance.

The behavior of the object can be implemented at run time.

It is achieved by using many different techniques; *method overloading, operator overloading, and method overriding* “

1.6 Benefits of OOP

1. Through inheritance, it's possible to eliminate redundant code and extend the use of existing classes
2. The principle of data hiding helps programmers to build more secure programs that cannot be invaded by code in other parts of the program.
3. It's possible to have multiple instances of an object to co-exist without interference.
4. Software complexity can be easily managed.
5. It's possible to map objects in the problem domain to those objects in the program

Email: Kioko@cuea.edu / **Mobile No:** --+254-725-695-782

© By Edward Kioko. All rights reserved for this module.

6. It's easy to partition the work in a program based on projects
7. Message passing techniques for communication between objects makes the interface description with internal system less complex.
8. The data centered design approach enables us to capture more details of a model in implementation form.
9. Object oriented systems can be easily upgraded from small to large systems

1.7 Object Oriented Languages

Programming languages need to support a number of OOP concepts to claim that they are objected oriented. They are categorized as follows as follows

- **Object-based programming languages**

This is the kind of programming that primarily supports encapsulation and object identity. Major features required include data encapsulation, data hiding, and automatic initialization and clear up of objects, operator overloading. However, they don't support inheritance and dynamic binding. Classical examples include ADA

- **Object-oriented Programming Languages**

They incorporate all object-based programming features plus features of inheritance and dynamic binding. classical examples include C++, python, Java ++, C#, small talk, Object Pascal e.t.c

1.8 Application of OOP

Some of the classical application of OOP includes

1. Real time systems
2. Simulation and modelling
3. Object-oriented database
4. Hypertext, hypermedia and hypertext
5. Expert systems
6. Neural networks and parallel programming · Decision support and office automation systems 13
7. CAM/CAD systems

The most popular application of OOP has been the user interface design such as windows. There are a hundred of windowing systems developed using OOP techniques. OOP is useful in developing complex problem since it simplifies tasks