Lecture-1

Rizwan Rashid

Outline

- Evolution of Object Oriented Programming (OOP)
- Difference between Object Oriented Approach & Modular/Structural Approach
- Object-Oriented Concepts and Principles

Evolution of Object Oriented Programming (OOP)

- Object-oriented programming, originating from the work on SIMULA by Ole-Johan Dahl and Kristen Nygaard
- The language introduced all elements of an object-oriented language such as
  - Encapsulation, inheritance, late binding, and dynamic object creation.

Evolution of Object Oriented Programming (OOP)

- SIMULA I in (1962-65) and SIMULA 67 in 1967
- Alan Kay integrated philosophy of SIMULA 67 in Smalltalk in 1970
- In 1980, Bjarne Stroustrup introduce the philosophy of SIMULA to C systems developers community.
  - C++ object-oriented language
- In 1990, James Gosling developed JAVA

Structured Languages

- Program consists of list of instruction
  - E.g. Get some input, add these numbers, divide by six, display that output
- NO organizing principle: complexity increases with increase in program size
- Example
  - C, Pascal, FORTRON

Structured Languages (Cont'd)

- Division into Functions
  - Large program divided into small functions
- A function has a clearly defined purpose and a clearly defined interface to the other functions
- Module: is grouping of functions togeather into larger entity called module

Structured Programming

- Dividing a program into functions and modules to improve clarity, quality of a program
- Two related problems with structured programs
  - Unrestricted Access
  - Unrelated data and procedure

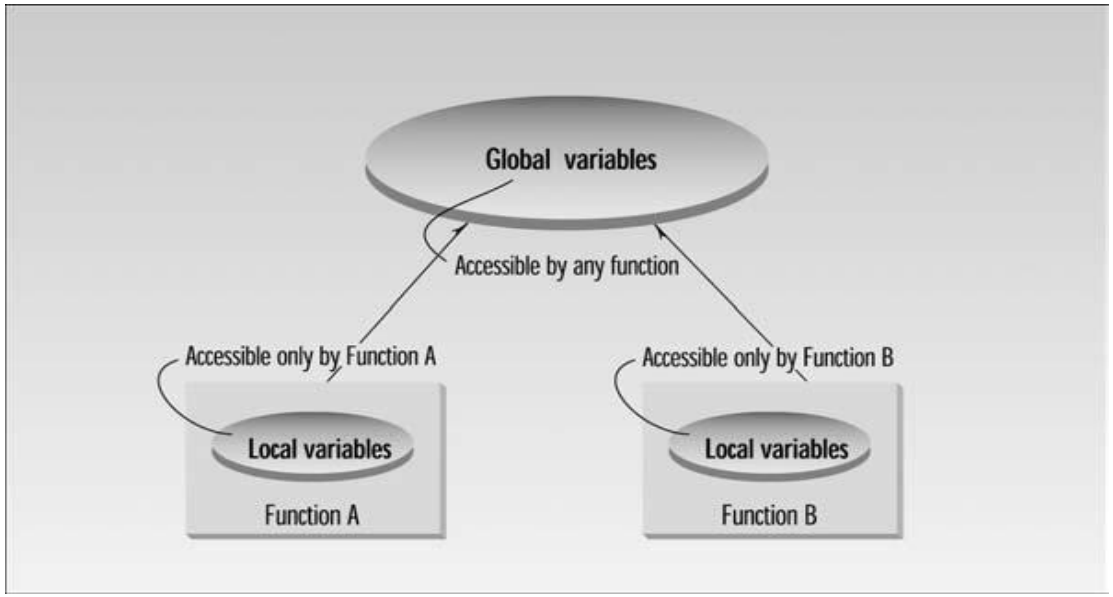
Unrestricted Access

- In structured language, two kinds of data
- Local data
  - Local access to function which is safe from modification by other functions

```
int AddNumber()
{
        int num1;
        int num2;
        int num3;
        num3 = num1 + num2;
        return num3;
}
```

Unrestricted Access (Cont'd)

- Global data
  - When one or more function must access same data
  - Can be accessed by any function in a program
  - Difficult to modify the program



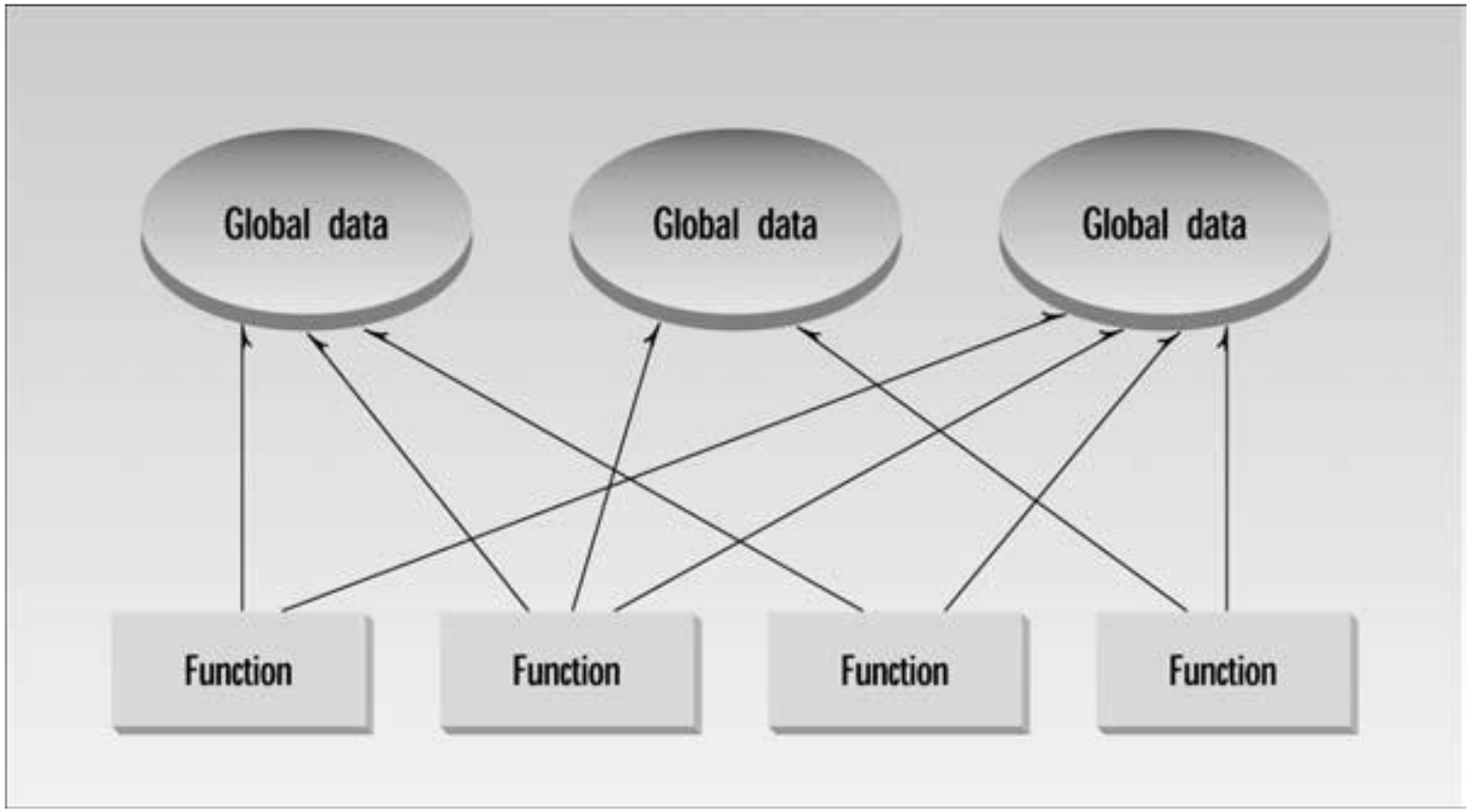
Example

```
int num1=2;
int num2=5;
int  num3;
Void AddNumber()
{
        num3 = num1 +  num2;
}
Void SubNumber()
{
        num3 = num2 – num1;
}
```

Unrelated data and procedure

- In a large program, there are many functions and many global data items
- The problem with the structured/modular paradigm is that this leads to larger number of potential connections between functions and data
  - It makes a program's structure difficult to conceptualize
  - It makes the program difficult to modify

Cont’d

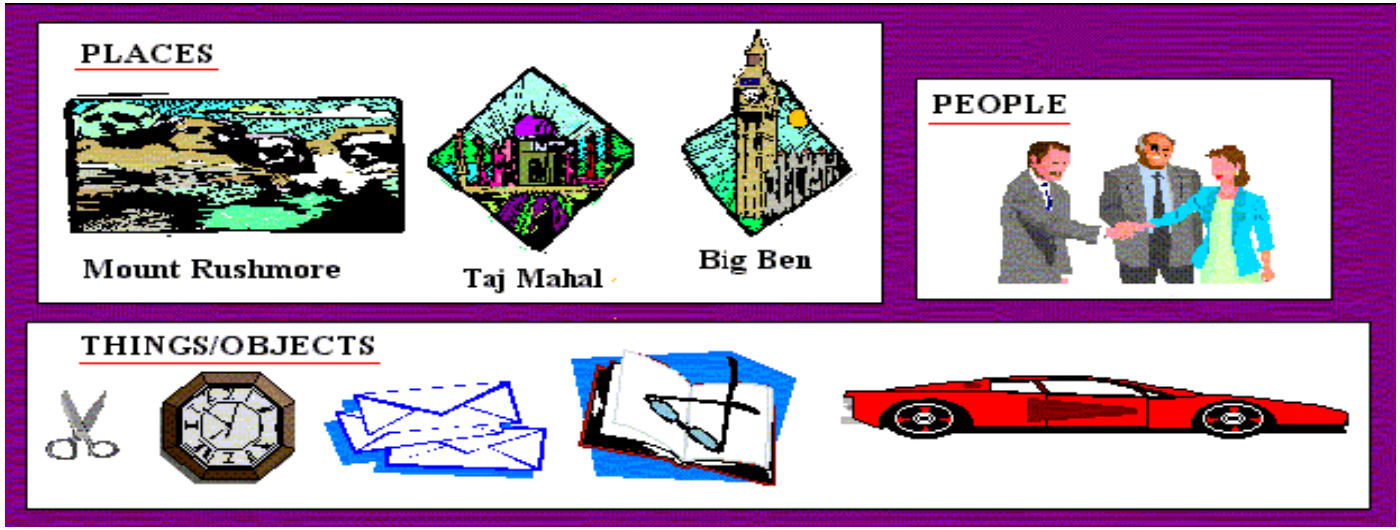

Global data
Global data
Global data
Function
Function
Function
Function

Real – World Modeling

- Physical World entities like car, people
- Focus on real objects to be mapped in computer program
- Objects has
  - Attributes – characteristics
    - For people; eye color, job title
    - For cars; horsepower, number of doors
  - Behavior
    - Is like function: call a function to do something

Object

- ***An Object*** is a computer representation of some real-world thing (i.e. person, place).
- Objects can have both *attributes* and *behaviors*



Object (Cont'd)

- When an object is mapped into software representation, it consists of two parts:
- **PRIVATE data structure**
  characteristics of private data structure are referred to as *ATTRIBUTES e.g. FirstName, LastName, Color etc.*
- **PROCESSES** that may correctly change the data structure
- Processes are referred to as *OPERATIONS* or *METHODS e.g. SetFirstName(), SetColor()*

Objects 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

Components in computer games
- Cars in an auto race
- Positions in a board game (chess, checkers)
- Animals in an ecological simulation
- Opponents and friends in adventure games

Elements of the computer-user environment
- Windows
- Menus
- Graphics objects (lines, rectangles, circles)
- The mouse, keyboard, disk drives, printer

Collections of data
- An inventory
- A personnel file
- A dictionary
- A table of the latitudes and longitudes of world cities

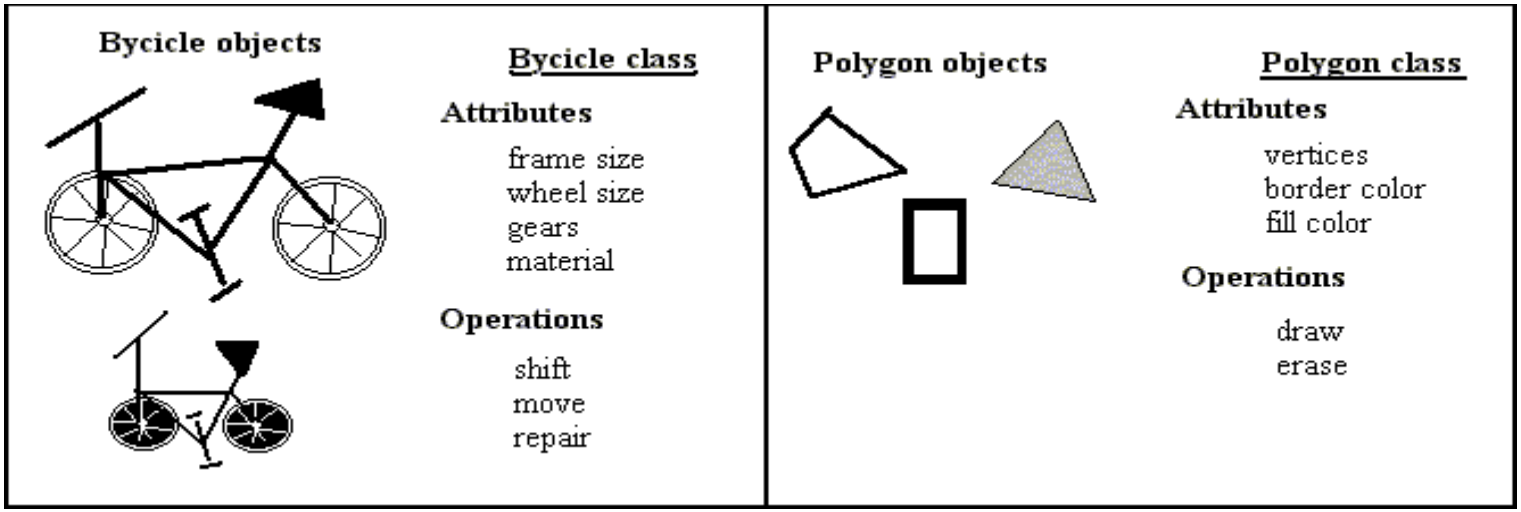
User-defined data types
- Time Angles
- Complex numbers
- Points on the plane

Data-storage constructs
- Customized arrays
- Stacks
- Linked lists
- Binary trees

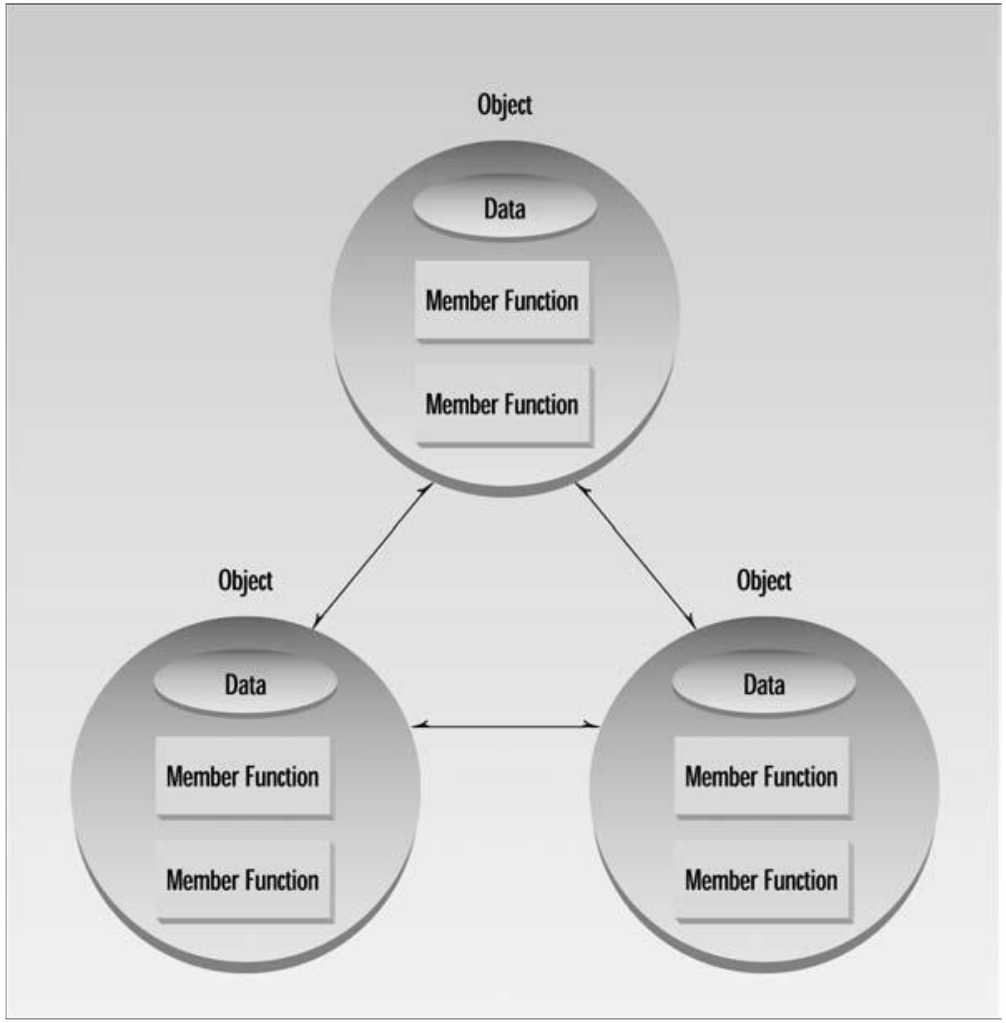
Human entities
- Employees
- Students
- Customers
- Salespeople

Class

- Objects with the same data structure ***(Attributes)*** and behavior ***(Methods or Operations)*** are grouped together called a ***class***
- Multiple objects can be created from the same class

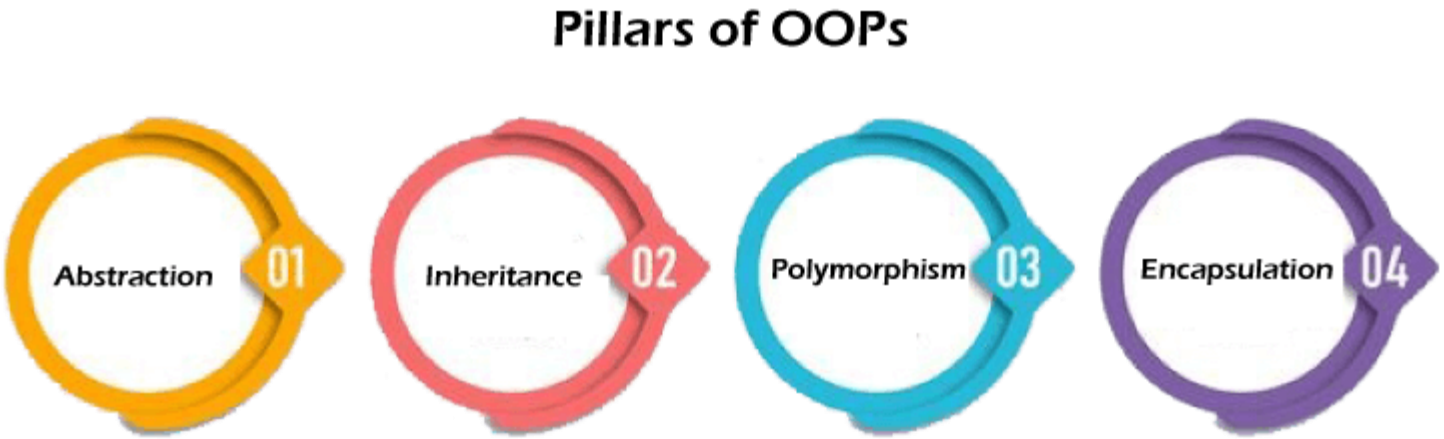


The object-oriented paradigm



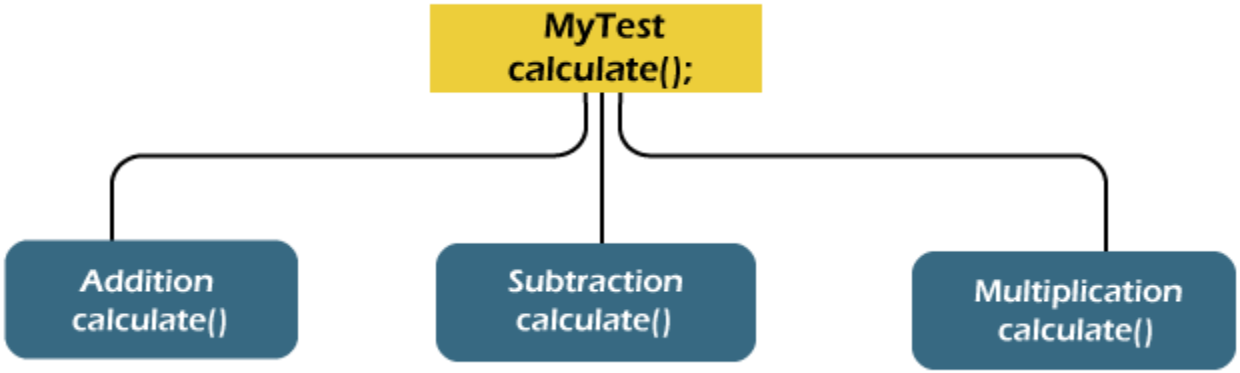
Pillars of OOPs

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism



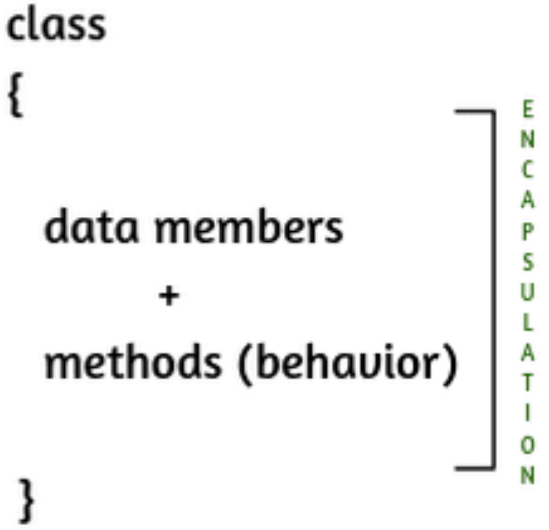
Pillars of OOPs

- Abstraction
  - Hide the implementation from the user but shows only essential information to the user



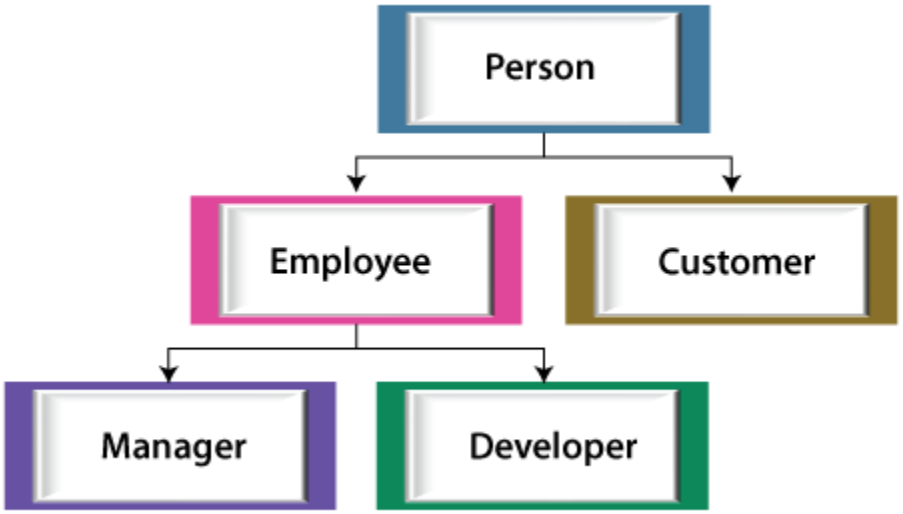
Pillars of OOPs

- Encapsulation
  - Mechanism that allows to bind data and functions of a class into an entity



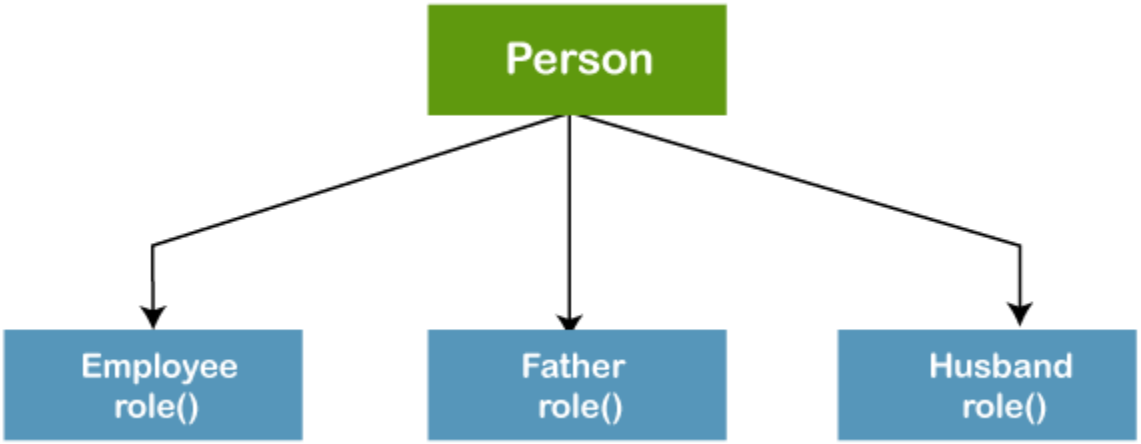
Pillars of OOPs

- Inheritance
  - Inherit or acquire the properties of an existing class (parent class) into a newly created class (child class)



Pillars of OOPs

- Polymorphism
  - Derived from the two words i.e. **ploy** and **morphs**. Poly means many and morphs means forms.
  - Create methods with the same name but different method signatures



Difference between Structured and OOP

Structured Programming

- Divides the code into modules or function
- Focuses on dividing the program into a set of functions in which each function works as a subprogram
- Main method communicates with the functions by calling those functions in the main program

Object Oriented Programming

- based on the concept of objects, contain both data and methods
- Focuses on representing a program using a set of objects which encapsulates data and object
- The objects communicate with each other by passing messages

Difference between Structured and OOP

Structured Programming

- No access specifiers
- **Data** is not secure
- Difficult to reuse code
- No way of data hiding

Object Oriented Programming

- Access specifiers such as private, public and protected
- **Data** is secured
- Easy to reuse code. E.g. inheritence
- Hide data is more secure

Procedural Approach	Object Oriented Approach

Public class **Circle**{
int radius;
Public void setRadius(int r)
{ radius = r;}
Public void showCircumference()
{
double c = 2*3.14*radius;
System.out.println("Circumferenceis"+ c);
}
Public static void main()
{
setRadius(5);
showCircumference();
//output would be 31.4
setRadius(10);
showCircumference();
// output would be 62.8
}
}
	Public class **Circle**{
	Private int radius;
	Public void setRadius(int r)
	{ radius = r;}
	Public void showCircumference()
	{
	double c = 2*3.14* radius;
	System.out.println("Circumference is"+
	c);
	}
	}
	Public class **runner**
	{
	Public static void main()
	{
	Circle c1= new circle();
	c1.setRadius(5);
	c1.showCircumference();
	//output would be 31.4; it belongs to c1
	Circle c2= new circle();
	c2.setRadius(10);
	c2.showCircumference();
	//output would be 62.8; it belongs to c2
	}
	}