Programming Technology

FOR THE STUDENTS OF BACHELOR IN COMPUTER ENGINEERING

Compiled by: Er. Shiva Ram Dam

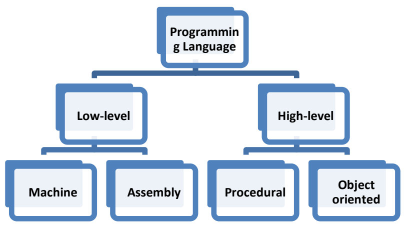
TABLE OF CONTENTS

Chapter 1: Introduction

[Credit: 5 hrs]

1.1. Review on Programming Languages

There are mainly two types of programming languages: High Level and Low level programming languages.



Assignment 1: Classify programming languages and explain in brief.

1.2. Programming Paradigms.

A programming paradigm is a style, or "way," of programming. It is a method to solve a problem using tools and techniques that are available to us following some approach. There are lots for programming language that are known but all of them need to follow some strategy when they are implemented and this methodology/strategy is paradigm. Apart from varieties of programming language there are lots of paradigms to fulfill each and every demand.

Some of the programming paradigms are:

i. Procedure oriented programming
ii. Object oriented programming
iii. Event oriented programming
iv. Aspect oriented programming
v. Subject oriented programming

i) Procedure Oriented Programming

- A computer program is a set of instructions for a computer to perform a specific task. The traditional programming languages like C, FORTRAN, Pascal, and Basic are Procedural Programming Languages.
- A procedural program is written as a list of instructions, telling the computer, step-by-step, what to do: Get two numbers, add them and display the sum.
- The problem is divided into a number of functions. The primary focus of programmer is on creating functions.
- While developing functions, very little attention is given to data that are being used by various functions.
- Procedural programming is fine for small projects. It does not model real world problems well. This is because functions are action oriented and does not really correspond to the elements of the problems.

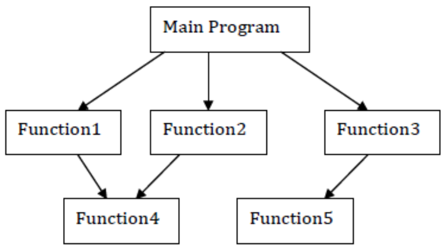


Figure: Typical structure of Procedural Oriented Program

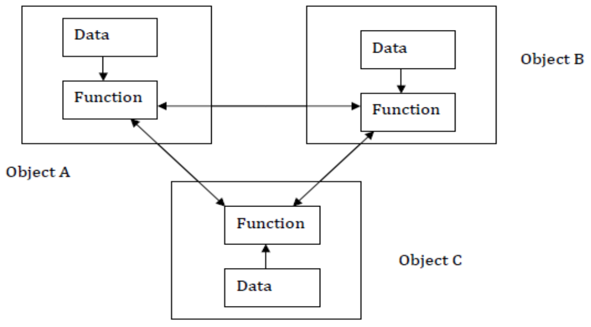
Features of Procedural/Structured Programming Language:

- Emphasis is on algorithm (step by step description of problem solving) to solve a problem. Each step of algorithm is converted into corresponding instruction or code in programming language.
- Large program is divided into a number of functions, with each function having a clearly defined purpose and a clearly defined interface to other functions in the program. The integration of these functions constitutes a whole program.
- Uses top-down approach in program design. In top-down programming, we conceive the program as a whole as a process to be decomposed.
- Most of the functions share Global data and these are more vulnerable to accidental changes.
- Data move openly around the system from function to function. Information is passed between functions using parameters or arguments

Assignment 2 : Enumerate the advantages and disadvantages of POP paradigm.

ii) Object Oriented Programming

- In Object-Oriented Programming, a program is decomposed into a number of entities called objects.
- *Data* and *the functions that operate on that data* are combined into that object. OOP mainly focus on data rather than procedure.
- It considers data as critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the functions that operate on it and protects it from accidental modification from outside functions.



Organization of data and functions in OOP

Features of Object Oriented Programming Language

- Emphasis is on the data rather than procedure. It focuses on security of data from unauthorized modification in the program.
- A program is divided into a number of objects. Each object has its own data and functions. These functions are called member functions and data are known as member data.
- Data is hidden and cannot be accessed by external functions.
- Follows bottom-up approach in program design. The methodology for constructing an object-oriented program is to discover the related objects first. Then, appropriate class is formed for similar objects. These different classes and their objects constitute a program.
- Object may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.

- Supports reusability of code. Once a class has been written, created and debugged, it can be used in a number of programs without modification i.e. a same code can be reused. It is similar to the way a library functions in a procedural language.
- Supports polymorphism.

Basic Characteristics of Object-oriented Languages

- Class and Objects
- Data abstraction
- Encapsulation
- Inheritance
- Reusability
- Polymorphism and overloading

Assignment 3: Explain the characteristics of Object-oriented languages.

Applications of using OOP:

Main application areas of OOP are:

- User Interface design such as windows, menus.
- Real time systems
- Simulation and modeling
- Object oriented databases
- AI and Expert Systems
- Neural Networks and Parallel processing
- Decision Support and Office Automation System, etc.

Benefits of OOP:

The main advantages are:

- It is easy to model a real system as programming objects in OOP represents real objects. The objects are processes by their member data and functions. It is easy to analyze the user requirements.
- With the help of inheritance, we can reuse the existing class to derive a new class such that redundant code is eliminated and the use of existing class is extended. This saves time and cost of program,
- In OOP, data can be made private to a class such that only member functions of the class can access the data. This principle of data hiding helps the programmer to build a secure program that cannot be invaded by code in other part of the program.
- With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.
- Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
- It is possible to have multiple instances of an object to co-exist without any interference i.e. each object has its own separate member data and function.

Assignment 4:

1. *Mention the advantages and disadvantages of OOP paradigm.*
2. *Differentiate between POP and OOP*

iii) Event Oriented Programming

An Event is a signal that informs an application that something important has occurred. For example, when a user clicks control on a form, the form can raise a click event and call a procedure that handle the event. There are various types of event associated with a form like: Click, Double Click, Close, Load, Resize, etc.

Event oriented Programming means that the program executes everything as a response to an event, instead of a top down program where the ordering of the code determines the order of execution.

Program code is based on what happens when the user does something. Events can be mouse moves, mouse clicks or double-clicks, keystrokes, changes made in textboxes, timing based on a timer, etc. Note that this is not an exhaustive list of events, just a small sampling. It is quite a bit different from top-down coding. An event-oriented language implies that an application (the computer program) waits for an event to occur before taking any action.

Assignment 5:

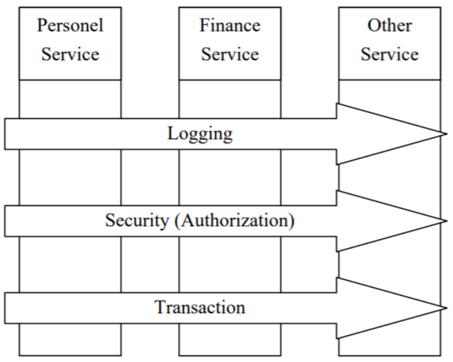
Mention the features, advantages and disadvantages of EOP

iv) Aspect Oriented Programming

Lets us consider an abstract class ***Dog.*** We can inherit its behavior using inheritance concept. Let us create a derived class ***Poodle***, this inherits Dog. Again let us define another unique behavior (method) and label as an ***ObedientDog***. Surely, not all Dogs are obedient. So ***Dog*** class cannot contain the Obedience behavior. So, this ***obedience*** is an aspect. An aspect is a common feature that's typically scattered across methods, classes, object hierarchies, or even entire object models.

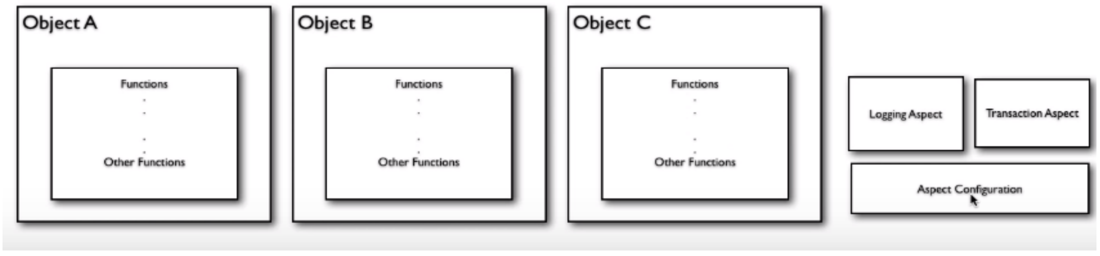
Aspect-Oriented Programming (AOP) is a new programming concept that was developed due to some shortcomings of object-oriented programming (OOP). The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects. (Such concerns are often termed crosscutting concerns in AOP literature.)

Although the OOP provides a rich set of tools for abstraction and modularization, it cannot address the problem with so called cross-cutting concerns. One can think the cross-cutting concerns as functionality that when implemented, will scatter around the final product in different components. Since this kind of functionality will cut through the basic functionality of the system, it is hard to model even with the OOP. Good examples of the cross-cutting concerns are authorization, synchronization, error handling and transaction management.



AOP tries to address the problem by modularizing the crosscutting functionality into more manageable modules – aspects. Unlike the OOP, AOP does not replace previous programming paradigms. AOPers generally say that OOP solves 90% of problems and AOP solves the 10% of problems that OOP isn't good at. Therefore it can be seen as a complementary to the object-oriented paradigm rather than a replacement. In aspect-oriented programming the system is divided into a two halves: the base program and the aspect program. The base program will contain the main functionality of the system and can be implemented using object-oriented programming. The aspect program will consist of the cross-cutting functionality that has been modularized away from the base program. This leads to a more concise structure since the functionality of the cross-cutting concerns is contained within well-defined modules.

There are several frameworks to implement aspect-oriented programming in Java world: Spring and AspectJ



For further learning:

https://www.youtube.com/watch?v=QdyLsX0nG30&list=PLE37064DE302862F8

https://www.youtube.com/watch?v=aNcTL9e7luM

https://docs.jboss.org/aop/1.0/aspect-framework/userguide/en/html/index.html

https://www.cs.helsinki.fi/u/paakki/laukkanen.pdf

https://flowframework.readthedocs.io/en/stable/TheDefinitiveGuide/PartIII/AspectOrientedProgramming.html

v) Subject Oriented Programming

In computing, SOP is an object-oriented software paradigm in which the state (i.e. fields) and behavior (i.e. methods) of objects are not seen as intrinsic to the objects themselves, but are provided by various subjective perceptions ("subjects") of the objects.

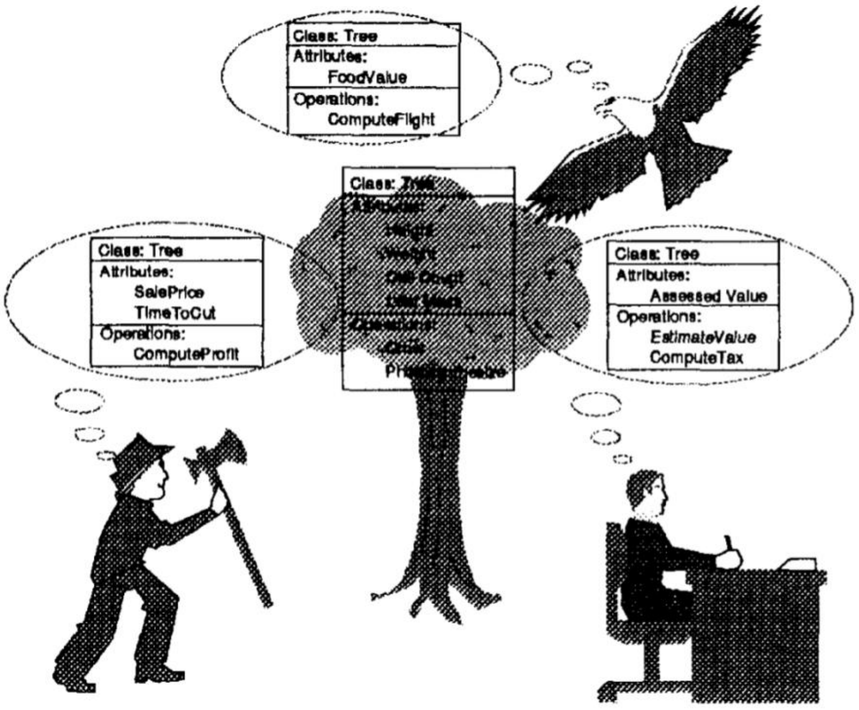


Figure: Many Subjective Views of an Object-Oriented Tree

In the real-world, properties of a tree like its height, cell-count, density, leaf-mass, etc. are intrinsic properties. From the point of view of a bird, a tree may also have measures of relative value for food or nesting purposes. From the point of view of a tax-assessor, it may have a certain taxable value in a given year. From the point of view of a woodsman, it might be different.

Subject-oriented programming is an enhancement of object-oriented programming that allows decentralized class definition. An application developer who needs new operations associated with classes can implement them him/herself, not by editing existing code for the classes, but as a separate collection of class definitions called a subject.

SOP allows OO systems to be built by flexible composition of components, which we call "subjects." A subject is a collection of classes, defining a particular view of a domain and/or providing a coherent set of functionality. Without eliminating the advantages of encapsulation, this approach eliminates the need for class ownership. An application developer can write all the code needed for the application, irrespective of which classes are involved, without leaving development requirements on others.

The overall goal of subject-oriented programming is to facilitate the development and evolution of suites of cooperating applications.

1.3. Integrated development environment, components of visual programming

An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor; build automation tools and a debugger. Several modern IDEs integrate with Intellisense coding features.

Some examples of IDEs are: Microsoft Visual Studio, Eclipse, NetBeans, Delphi, JBuilder, Borland C++ Builders, FrontPage, Dreamweaver etc.

Benefits of using IDEs

- Faster setup
- Faster development tasks:
- Continual learning:
- Standardization:

For more understanding:

https://www.veracode.com/security/integrated-development-environment

https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

Visual Programming:

A visual programming language (VPL) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually.

Visual programming lets humans describe processes using illustration. Whereas a typical text-based programming language makes the programmer think like a computer, a visual programming language lets the programmer describe the process in terms that make sense to humans.

Examples of visual programming languages include: Alice, GameMaker, Kodu, Lego, Scratch, etc.

A programming language that a visual representation (such as graphics, drawings, animation, buttons or icons etc.) is known as visual programming.

The term visual programming refers to creating or developing windows based applications or graphical user interface GUI applications. Different graphical objects are used to create the graphical user interface. These objects include windows, buttons, list boxes, and menus etc. these objects are provided by the visual programming languages as built in components.

Advantages of VP

Following are the most important advantages of visual programming languages:

- These languages are easy to learn and use.
- These languages provide many built-in objects that can be used in developing the new programs. New objects can also be created.
- The user-interface can be designed very easily by using mouse. The components are placed on the main interface component like forms. These components can be resized and moved easily.
- These languages provide facility to attach code to each interface component. The attached code is executed when the user interacts with the interface component.
- The users can use the visual applications very easily.

Disadvantages of VP

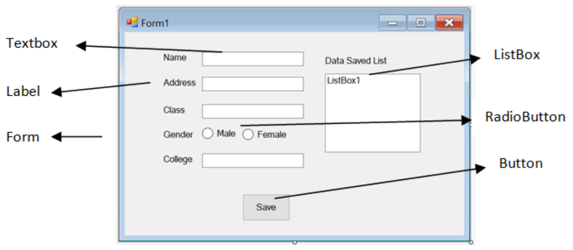
Following are some disadvantages of visual programming languages:

- These languages require computer with more memory, high storage capacity of hard disk, and faster processor.
- These languages can only be implemented on graphical operating systems like Linux and windows.

Components of Graphical Visual Interface

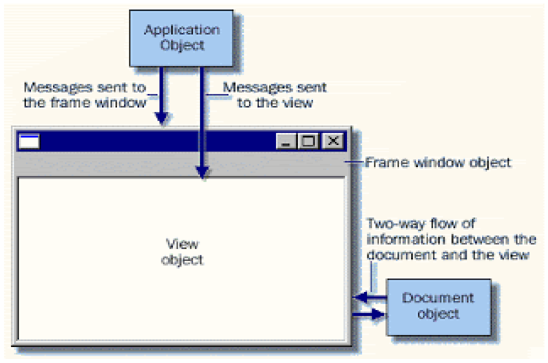
Following are the popular components of Visual programming. There are much more than these.

1. **Window:** A window sometimes also called a form is the most important of all the visual interface components. A window plays the role of the canvas in a painting. Without the canvas there is no painting. Similarly, without the window there is no user interface. The components that make up the user interface have to be placed in the window and cannot exist independent of the window. Simply, Forms are the windows which hold the various controls (buttons, text boxes, etc.) which make up our application.
2. **Buttons:** A button is used to initiate an action. The text on the button is indictive of the action that it will initiate. Clicking on a button initiate the action associated with the button.
3. **Text boxes**: Text box is a control that allows entering text on a form at runtime. Text boxes are used to accept information from the user. The user interface will display one text for each piece of information
4. **List Boxes or Pop-up Lists**: List boxes are used to present the user with the possible options. The user can select one or more of the listed options.
5. **Label**: The label is used to place text in a window. Typically label is used to identify controls that do not have caption properties of their own.
6. **Radio button or Option Button**: The radio buttons, also referred as option buttons, are used when the user can select only one of multiple options.
7. **Check boxes**: A Check box allows the user to select one or more items by checking the check box/check boxes concerned. The CheckBox control allows the user to set true/false or yes/no type options.



1.4. Application object, main window object, view object, document object

In the early days of Microsoft Foundation Class (MFC), applications were built very mush like the sample programs. An application had two principal components: application object and window object. MFC gradually enhanced to MFC 2.0 using document/view architecture and then to MFC 4.0.


Application Object
Messages sent to the frame window
Messages sent to the view
Frame window object
View object
Two-way flow of information between the document and the view
Document object

An **Application object** represents the application itself. Its primary job is to create the windows object, and the window object, in turn, processes messages.

The frame **window object** is the application's top-level window and usually includes a resizing border, a caption bar, a system menu, and Minimize, Maximize, and Close buttons. The arrows in the illustration show the direction of data flow during various application operations.

The **view object** is a child window sized to fit the frame window and serves as the client area for the parent. One or more View objects represent views of that data.

The application's data is stored in the **document object,** which is displayed in the view.

1.5. Document-view architecture and its advantages

The **Document/View architecture** is the foundation used to create applications based on the Microsoft Foundation Classes library. It allows us to make logical separation between different parts that compose a computer program (views and documents). **View** is what the user sees as part of your application and the **document** is where a user would work on.

MFC Document View Architecture is a framework to manage user module data and view of the module separately. The design it follows is known as model-view architecture. Data part is managed by model and in MFC, this is known as Document, and display part is managed by view class.

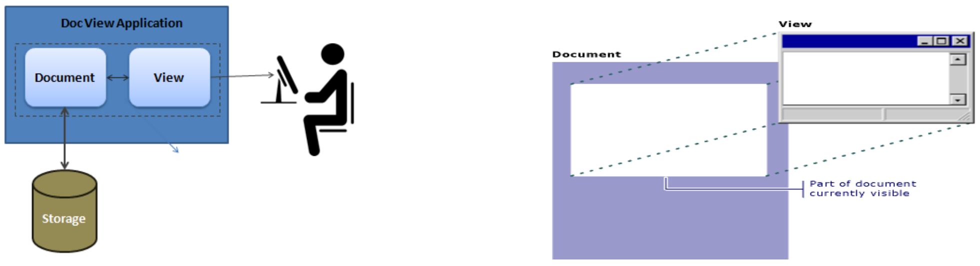


Figure: Document-view Architecture

The parts that compose the Document/View architecture are a **frame**, one or more **documents**, and the **view**. Putting together, these entities make up a usable application.

Document *(similar to a bucket that holds user's data)*

Document is the class to manage data part of the objects. Application often derives a class from CDocument. This deals with saving and retrieving data fields to and from the file stream. MFC often uses CArchive and serialization process to do the same. Users however have other options like direct CFile calls to synch the data to file system or query SQL server to do the same. Document module also deals with the logic to maintain the data and also responsible for managing different attributes of the data.

View

A view is the platform where the user works to do her job done. View is the class module to display data of the document. A derived class of CView is often used for this purpose. This display part can be the display of the document in graphical form or with UI elements or in the form of printable view which contains formatted text.

Frame

A Frame is a combination of the building blocks, the structure and the borders of an item. A Frame gives physical presence to a window. MFC framework uses a Frame window or class CFrame to display window in the screen and thus a CFrame class is always needed in any application which follows document view architecture.

Key classes of Document/View Architecture.

CDocument	It loads, stores, and manages the program's data. It also contains the functions that are used to access and work with the data.
CView	provides the basic framework for output to the view window and the printer, and communicates with the associated document. Also acts as intermediary between document and the user. the view renders an image of the document on the screen and interprets user input as operations upon the document.
CFrameView	Supports objects that provides the frame around one or more views of a document.
CDocTemplate	is the class that binds together the frame, view, document, and a set of application resources. uses two document template classes: **CSingleDocTemplate** for SDI applications and **CMultiDocTemplate** for MDI applications. An SDI application uses the main frame window to display its document. Only one document can be open at a time. An MDI application uses the main frame window as a workspace in which the user can open multiple document frame windows.

Reference: http://findbestvideos.blogspot.com/2009/11/documentview-fumdamentals.html
https://docs.microsoft.com/en-us/cpp/mfc/document-view-architecture?view=vs-2019

Advantages of Document/view Architecture:

This design may look complex for a small dialog based application. However this actually helps when dealing with a large application where there are multiple data objects and different view of the objects. This design decouples data from view.

Following are the advantages:

- Data managing part may go separate changes where display part can remain unchanged. Again there can be display part to change where data manage part can remain same. For both the cases this design fits well.
- When a user updates one of the views, that view object calls ***CDocument::UpdateAllViews***. That function notifies all of the document's views, and each view updates itself using the latest data from the document. The single call to UpdateAllViews synchronizes the different views. This scenario would be difficult to code without the separation of data from view, particularly if the views stored the data themselves.

Limitations of Doc/View Architecture (Alternatives to the Doc/View Architecture)

MFC applications normally use the document/view architecture to manage information, file formats, and the visual representation of data to users. For the majority of desktop applications, the document/view architecture is an appropriate and efficient application architecture. This architecture separates data from viewing and, in most cases, simplifies your application and reduces redundant code.

However, the document/view architecture is not appropriate for some situations. Consider these examples:

- If you are porting an application written in C for Windows, you might want to complete your port before adding document/view support to your application.
- If you are writing a lightweight utility, you might find that you can do without the document/view architecture.
- If your original code already mixes data management with data viewing, moving the code to the document/view model is not worth the effort because you must separate the two.

While the document/view model is a good default and useful in many applications, some applications need to bypass it. The point of the document/view architecture is to separate data from viewing. In most cases, this simplifies your application and reduces redundant code. As an example of when this is not the case, consider porting an application written in C for Windows. If your original code already mixes data management with data viewing, moving the code to the document/view model is harder because you must separate the two. You might prefer to leave the code as it is.

There are many approaches to bypassing the document/view architecture, of which the following are only a few:

- *Treat the document as an unused appendage and implement your data management code in the view class. Overhead for the document is relatively low, as described below.*
- *Treat both document and view as unused appendages. Put your data management and drawing code in the frame window rather than the view. This is close to the C language programming model.*
- *Override the parts of the MFC framework that create the document and view to eliminate creating them at all.*

1.6. Virtual machines and runtime environments

Virtual machines:

A virtual machine (VM) is a software program or operating system that not only exhibits the behavior of a separate computer, but is also capable of performing tasks such as running applications and programs like a separate computer. A virtual machine, usually known as a guest is created within another computing environment referred as a "host." Multiple virtual machines can exist within a single host at one time.

A virtual machine is also known as a guest.

Depending on their use and level of correspondence to any physical computer, virtual machines can be divided into two categories:

1. System Virtual Machines: A system platform that supports the sharing of the host computer's physical resources between multiple virtual machines, each running with its own copy of the operating system. The virtualization technique is provided by a software layer known as a hypervisor, which can run either on bare hardware or on top of an operating system. Eg: VirtualBox
2. Process Virtual Machine: Designed to provide a platform-independent programming environment that masks the information of the underlying hardware or operating system and allows program execution to take place in the same way on any given platform.

Some of the advantages of a virtual machine include:

- Allows multiple operating system environments on a single physical computer without any intervention
- Virtual machines are widely available and are easy to manage and maintain.
- Offers application provisioning and disaster recovery options

Some of the drawbacks of virtual machines include:

- They are not as efficient as a physical computer because the hardware resources are distributed in an indirect way.
- Multiple VMs running on a single physical machine can deliver unstable performance

Reference: *https://www.desktop-virtualization.com/glossary/virtual-machine/*

Runtime environments

A runtime environment is the execution environment provided to an application or software by the operating system. In a runtime environment, the application can send instructions or commands to the processor and access other system resources such as RAM, which otherwise is not possible as most programming languages used are high level languages.

The runtime environment provides a state for the target machine to have access to resources such as software libraries, system variables and environment variables, and provide all necessary services and support to the processes involved in the execution of the application or program. In certain software or applications such as Adobe Flash Player or Microsoft PowerPoint Viewer the runtime environment is available to end users as well.

Software developers need a runtime environment to test their software's functioning. As a result, all software development applications include a runtime environment component which allows the testing of the application during execution. Tracking bugs or debugging for any errors are done in most applications with the help of runtime environments. Runtime execution continues even if the application or program crashes. Most runtime environments are capable of reporting of why an application or program crashed. One of the more popular runtime environments is Java, which helps Java applets and applications to be executed in any machine which has a Java runtime environment installed.

Exam Questions:

1. What is Document-View architecture? Explain event-oriented programming, subject-oriented and aspect-oriented programming. [2018 Spring]
2. Define AOP. What are the differences between POP and OOP? [2018 Fall]
3. What is IDE? What are the components of Visual programming? [2017 Fall]
4. What is an IDE and how does it work? Discuss the key features in Visual Studio .NET [2017 spring]
5. Describe the features of four different language paradigms and give one example language for each category. [2017 spring]
6. Describe the features of Procedure, Event and Object Programming? [2016 Fall]
7. Define EOP and AOP. Explain virtual machine and run time environments. [2016 Spring]
8. Describe the features of AOP and SOP. [2015 spring]
9. Explain Document/view architecture? What are the four key classes used in this architecture. [2015 spring]
10. What do you mean by IDE? What are the differences between EOP and OOP? [2015 Fall]
11. Differentiate between OOP and EOP. [2014 spring]
12. What are the advantages of DVA? Explain in brief about the components of Visual programming. [2014 Spring]
13. Explain about the significance of AOP and SOP with example. [2014 fall]
14. Explain DV architecture with its advantages. [2013 spring]
15. Write short notes:
    a) Virtual machine Vs Runtime Machine [2018 Fall]
    b) Document view Architecture and its advantages [2017 Fall]
    c) Doc-View Architecture [2016 Fall, Spring]
    d) Window object. [2015 fall]
    e) Components of Visual programming [2014 fall]
    f) Event oriented language [2013 spring]

*****End of Chapter*****