Four Pillars of OOP

Four Pillars of OOP

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

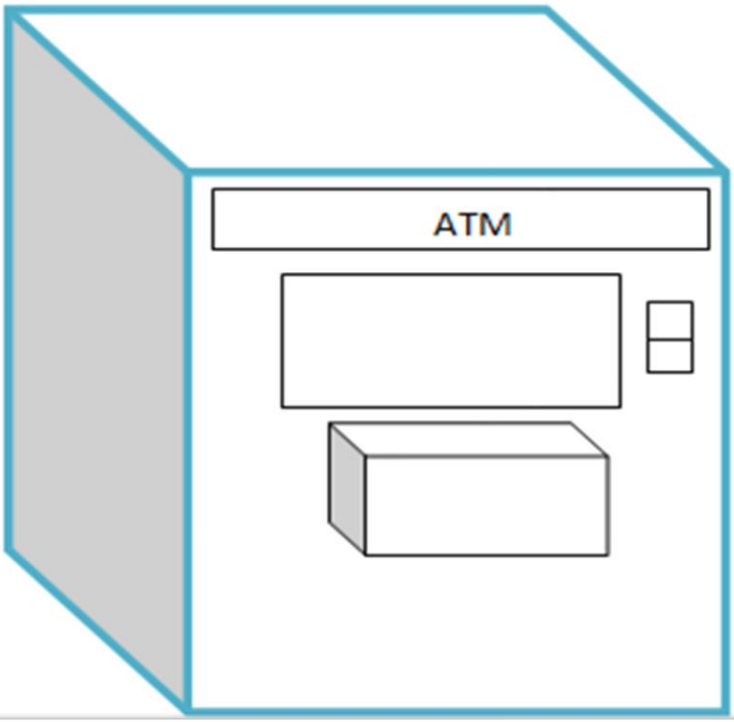
What is Abstraction

- Grady Booch defined abstraction as

"An abstraction indicates the important characteristics of an object which differentiates it from all other kinds of objects and to provide crisply defined conceptual boundaries, relative to the perspective of the viewer"

Let's take an example

Let's see how an ATM machine works



The user has to swipe his card, provide pin, enter amount and withdraw cash

All this is done by user by using all elements provided in ATM's user interface and other external devices like card swiping component, but a user will not know how exactly has this machine worked to withdraw amount from his account and provide him with cash

Hence, only those details that are required by the user are exposed and rest of the details are hidden

What is Encapsulation

- Encapsulation means bundling or hiding of data into a single structure, thus hiding it from external influence.
- In OOP encapsulation is done so that all non essential details are hidden from the user

Different Visibility levels for members of a class

- Public
- Package
- Protected
- Private

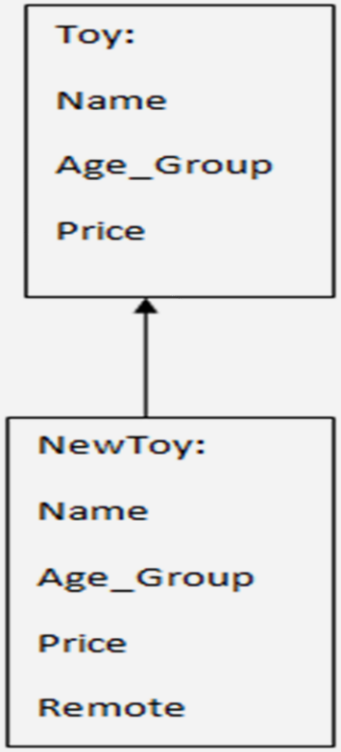
Visibility of members of class

	Public	Package	Protected	Private
Within class	Yes	Yes	Yes	Yes
Outside class inside package	Yes	Yes	Yes	No
Outside package	Yes	No	(through inheritance)	No

Inheritance

- Inheritance supports reusability. It allows us to create a new class or design from an already existing class or design.
- In this way the new class gets all the features (non private) of an existing class ,and it can also have its own features
- The class that is providing with all the features is **Base** class and the class that is absorbing all the features from parent class is called as **Derived** class

Let's look at an example



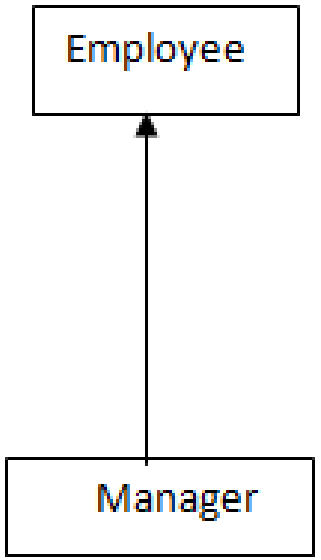
In above example there is a Toy class that contains attribute Name,Age-Group and Price .Now if a programmer wants to create a new class that will have attributes from Toy class as well as some of its own attributes, Inheritance can be applied in this case .So a new class is created as NewToy that has all the attributes of Toy class and also its own attribute Remote

Types of Inheritance

- Single
- Multiple
- Multilevel

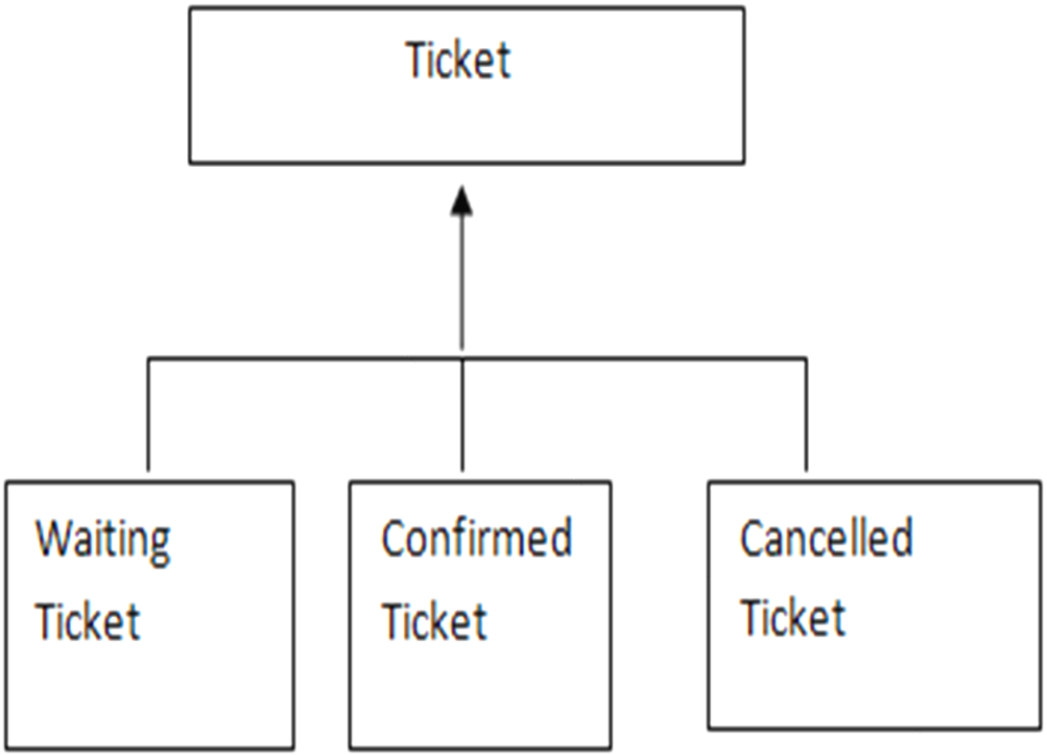
In single Inheritance, a class is derived from only one base class

Example:-



In the above example, Manager Class is derived from a single base class Employee

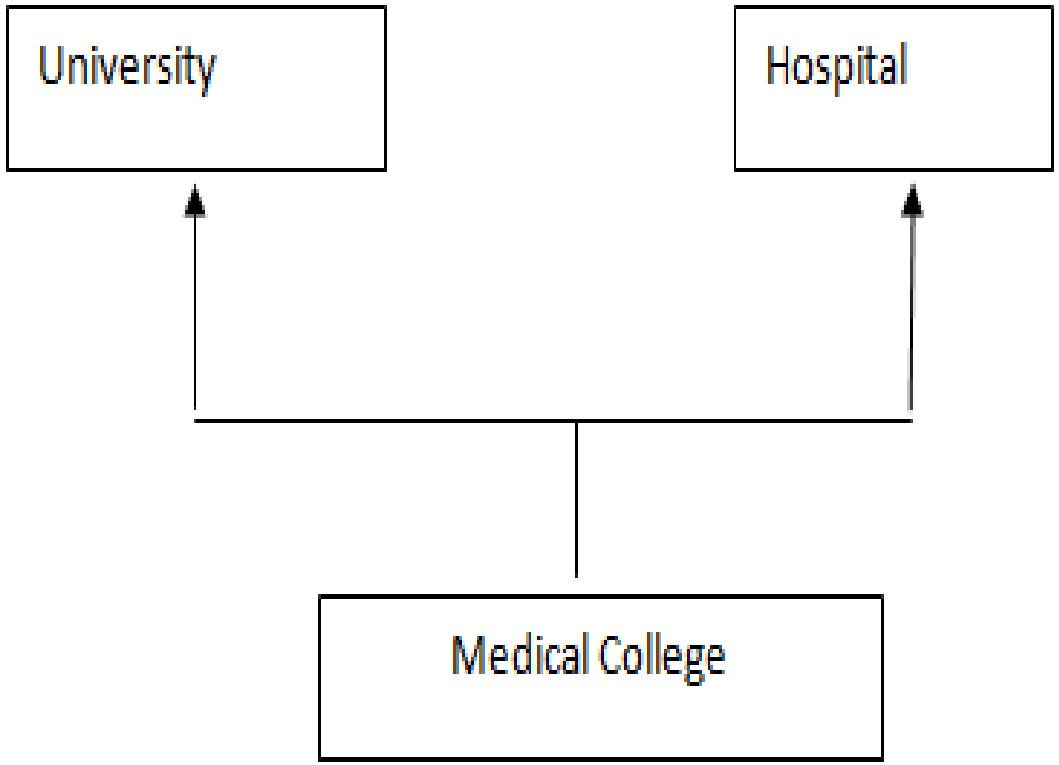
Let's look at another example



In the above example a base class can have more than one derived class , but here all derived classes will have only one base class

Multiple Inheritance:

This type of inheritance allows a class to be derived from more than one base class

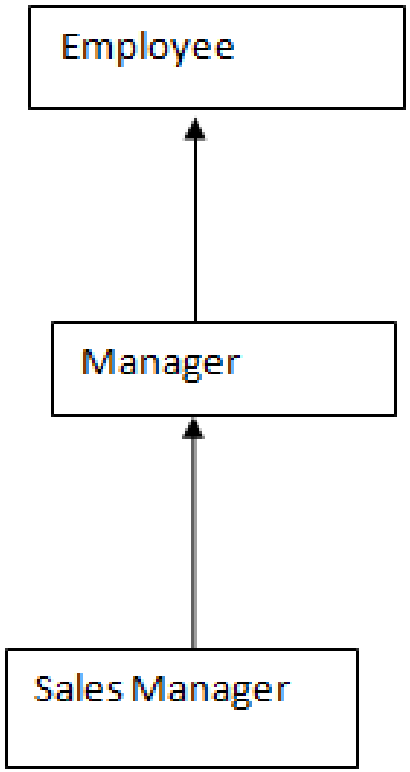


In the above example University and Hospital are base class and Medical College is derived class

Multilevel Inheritance:

In this type of Inheritance a class can be a derived class from a base class and can become a parent for another derived or child class

Example



In the above example Manager is parent for Sales Manager, but child for Employee class

Inheritance syntax

- public class Employee{ …}

  public class Manager extends Employee { …}

Exercise

- Create a class Ticket with function acceptDetails and displayDetails()
- Inherit Ticket into WaitingTicket and access above methods

Polymorphism

- Polymorphism states that an entity can exist in multiple form at the same time or as per requirement
- There are two types of polymorphism

  Static

  Dynamic

Static Polymorphism

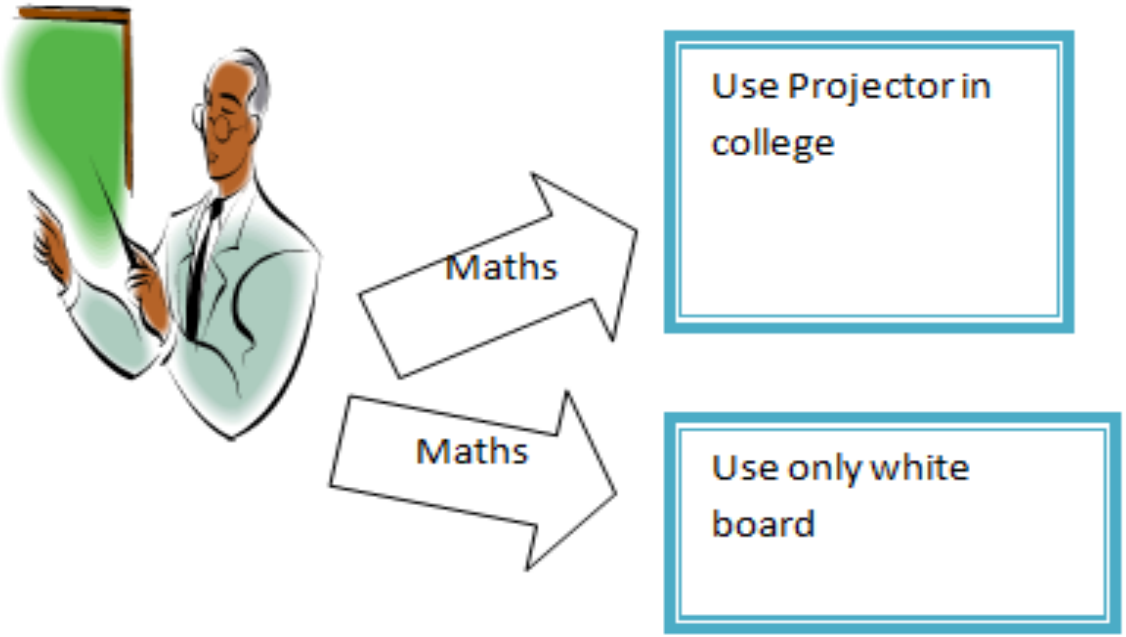
- Static polymorphism is achieved by creating methods that have same name ,but have different parameters

Dynamic Polymorphism

- Dynamic Polymorphism states that an entity can change it's form as per requirement
- Dynamic polymorphism requires that a method that is available through inheritance from parent class has to be overridden in child class. In such a scenario, the parameters for the method will not change (Subject) In this case, but logic for the method can be rewritten (Use of white board instead of projector)

Let's take the above example again,

A teacher may teach same subject in a different way in college ,and in a different way in his home coaching classes



Dynamic polymorphism requires that a method that is available through inheritance from parent class has to be overridden in child class. In such a scenario, the parameters for the method will not change (Subject) In this case, but logic for the method can be rewritten (Use of white board instead of projector)

Exercise

- Jason wants to create a class through which he can send messages using parameters of different types .Help Jason by implementing function overloading

Exercise

Observe below class

```
public class Ticket {
public void setTicketDetails()
{
 System.out.println("Ticket details");
}
}
```

Create three class WaitingTicket ,CancelledTicket ,ConfirmedTicket

Override the method in Ticket class with each class having it's own function definition Create another class with below definition

```
 public class TicketDetails {
    public void callTicket(Ticket ticket)
 {
 ticket.setTicketDetails()
 }
}
```

Call the above function with instances of WaitingTicket ,CancelledTicket and ConfirmedTicket

Lets Summarize

- Encapsulation means bundling or hiding of data into a single structure, thus hiding it from external influence
- Inheritance supports reusability. It allows us to create a new class or design from an already existing class or design.
- Polymorphism states that an entity can exist in multiple form at the same time or as per requirement

Packages

- A package in java is a group of related classes and interfaces
- Packages help in categorizing and maintaining classes and interfaces
- Packages provide access protection and help in avoiding naming collision
- Package is created using package keyword at the top of the file where class is created
- Packages can be imported using import statement
- Java provides with many in-built packages like java.util,java.io,

Static members of class

- A member variable or method declared as static is shared among all objects
- Only one copy of such a member is created in memory
- A static method cannot access non static variables of class

Static members of class

- Static variables
- Static Methods
- Static class

Static Block

- A block can be declared as static for initializing static variables of class
- Such a block is called before main method

Logging API

- In Java, **logging** is an important feature that helps developers to trace out the errors. Java is the programming language that comes with the **logging** approach. It provides a **Logging API** that was introduced in Java 1.4 version. It provides the ability to capture the log file

Logging API

- **Logging** is an API that provides the ability to trace out the errors of the applications. When an application generates the logging call, the Logger records the event in the LogRecord. After that, it sends to the corresponding handlers or appenders. Before sending it to the console or file, the appenders format that log record by using the formatter or layouts.

Components of Logger

- The Java Logging components used by the developers to create logs and passes these logs to the corresponding destination in the proper format. There are the following three core components of the Java logging API:
- Loggers
- Logging Handlers or Appender
- Logging Formatters or Layouts

Loggers

- The code used by the client sends the log request to the Logger objects. These logger objects keep track of a log level that is interested in, also rejects the log requests that are below this level.

Logging Handlers

Java Logging API allows us to use multiple handlers in a Java logger and the handlers process the logs accordingly. There are the five logging handlers in Java:

- **StreamHandler:** It writes the formatted log message to an OutputStream.
- **ConsoleHandler:** It writes all the formatted log messages to the console.
- **FileHandler:** It writes the log message either to a single file or a group of rotating log files in the XML format.
- **SocketHandler:** It writes the log message to the remote TCP ports.
- **MemoryHandler:** It handles the buffer log records resides in the memory.

Logging Formatters or Layouts

- The logging formatters or layouts are used to format the log messages and convert data into log events. Java SE provides the following two standard formatters class:
- SimpleFormatter
- XMLFormatter

Java Logging Levels

FINEST	Specialized Developer Information
FINER	Detailed Developer Information
FINE	General Developer Information
CONFIG	Configuration Information
INFO	General Information
WARNING	Potential Problem
SEVERE	Represents serious failure
Special Log Levels	
OFF	Turns off the logging
ALL	Captures everything

Using Logging libraries

- Simple Logging Facade for Java (abbreviated SLF4J) acts as a facade for different logging frameworks (e.g., java.util.logging, logback, Log4j). It offers a generic API, making the logging independent of the actual implementation.
- This allows for different logging frameworks to coexist. And it helps migrate from one framework to another. Finally, apart from standardized API, it also offers some "syntactic sugar."

Using SLF4j with Maven

- Maven, a Yiddish word meaning *accumulator of knowledge*, Maven is a tool that can be used for building and managing any Java-based project. It has been created to make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

SlF4J

```
public class App
{
    public static void main( String[] args )
    {
     Logger logger = LoggerFactory.getLogger(App.class);

         logger.info("This is an information message");
         logger.error("this is a error message");
         logger.warn("this is a warning message");
    }
}
```

Cohesion

- Cohesion in Java is the Object-Oriented principle most closely associated with making sure that a class is designed with a single, well-focused purpose. In object-oriented design, cohesion refers to how a single class is designed

Cohesion

- The advantage of high cohesion is that such classes are much easier to maintain and less frequently changed than classes with low cohesion. Another benefit of high cohesion is that classes with a well-focused purpose tend to be more reusable than other classes.

Loose Coupling

- When two classes, modules, or components have low dependencies on each other, it is called loose coupling in Java. Loose coupling in Java means that the classes are independent of each other. The only knowledge one class has about the other class is what the other class has exposed through its interfaces in loose coupling. If a situation requires objects to be used from outside, it is termed as a loose coupling situation.