**Defining Object Orientation**

* Object orientation is a software development methodology that relates to the real world.
* An object is the core concept involved in object orientation.
* An object is the representation of a real world entity or concept.
* For example, an employee, a window, a car, or a bird can be modeled as objects.
* You can think of an Object-oriented (OO) model as a collection of objects and their inter-relationships.

**Basic Concepts of Object Oriented Programming**

It is necessary to understand some of the concepts used extensively in object-oriented programming. These include:

* Objects
* Classes
* Data abstraction and encapsulation
* Inheritance
* Polymorphism
* Dynamic binding
* Message passing

**Foundation of Object Orientation**

* Object orientation involves splitting the software system into component parts and arranging the parts in a hierarchy.
* An object-oriented program consists of classes and objects.
* Objects communicate with each other by using messages**.**

**CCM01238.WMF**

What are classes, objects, and Messages

messages?

**Objects**

* An object means a material thing that can be presented physically.
* An object:
  1. has a state.
  2. may display some behavior.
  3. has a unique identity.
* Two objects may have the same behavior and state, but they can never have the same identity.

A car positioned at one place defines its state

Movement of the car defines its behavior

**MMY 88**

The car’s registration number,

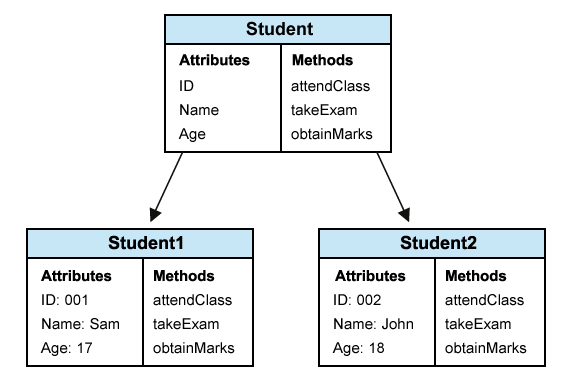
**MMY 88**

MMY 88, shows its identity.

**Classes:**

* A class consists of a blueprint of objects that share a common structure and behavior.
* The data stored within an object represents the state of the object. This data is called attributes.
* The behaviors are contained in methods.
* A method contains the programming code to implement a particular behavior.
* An object can be associated with multiple methods, each having a set of instructions included in it.
* Each method of an object has a unique purpose and helps to implement a particular behavior of the object.
* An object is an instance of a class.
* Objects communicate with each other through messages.

**Student class**



**Object of Student class Object of Student class**

**Data Abstraction and Encapsulation**

The wrapping up of data and function into a single unit (called class) is known as encapsulation. Data and encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

Abstraction refers to the act of representing essential features without including the background details or explanation.

**Inheritance**

Inheritance is the process by which objects of one class acquired the properties of objects of another classes. It supports the concept of hierarchical classification. For example, the bird, ‘robin’ is a part of class ‘flying bird’ which is again a part of the class ‘bird’. The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived.

In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined feature of both the classes.

**Polymorphism**

Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than on form. An operation may exhibit different behavior is different instances.

**Dynamic Binding**

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. It is associated with polymorphism and inheritance.

**Message Passing**

An object-oriented program consists of a set of objects that communicate with each other.

**Advantages of Object Orientation**

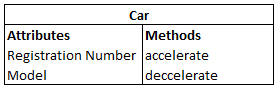
* Realistic modeling:
  + The object-oriented approach models the real world.

Object-oriented representation

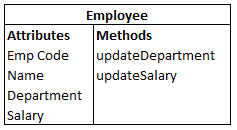
Real world object

**MMY 88**

of the real-world object



* Reusability:



Leave Accounting Application

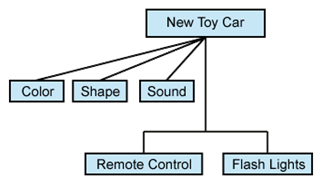
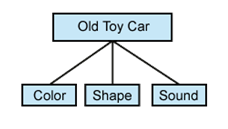
Payroll application

Employee Recruitment Application

* Modularity:
  1. The object-oriented approach allows objects to be maintained independently of other objects.

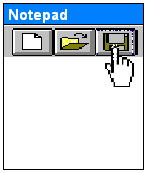
Independent objects in Training Management System

* Flexibility to change:
  1. The object-oriented approach allows you to create a new class by adding some new features to an existing class.



* Information hiding:
  1. The object-oriented approach allows limited access to information.

A person who is saving a file



on a computer does not know

how the computer saves the file

on the hard disk. The person just

knows that clicking the Save button

will save the file.

**Object-Oriented Analysis (OOA)**

**Analysis:**

* Is a phase where users and developers get together and arrive at a common understanding of the system
* Requires the developer to concentrate on obtaining maximum possible information about the problem domain
* Results in one of the end products as specification of the function of the system

**Design:**

* Generates the blueprint of the system that has to be implemented
* Involves identifying classes using Abbott’s technique
* Abbott’s technique follows the listed steps:
* Write English description of the problem
* Underline nouns (nouns represent candidate classes)

**Object-Oriented Programming (OOP)**

* Is a way of writing programs
* Some applications built using OOP techniques are:

Computer-Aided Design (CAD)

Computer-Aided Manufacturing (CAM)

Artificial Intelligence (AI) and Expert Systems

Object-Oriented Databases

**Enumeration (ENUM)**

* An enumeration is a user-defined type consisting of a set of named constants called enumerators
* It serves to create data types, that is not limited to either numerical or character constants or Boolean values
* The syntax to declare an enum is as follows:

enum model\_name {

value1,

value2,

value3,

. .};

* By default, the first enumerator has a value of 0
* Each successive enumerator is one larger than the value of the previous one, unless you explicitly specify a value for a particular enumerator
* An enumerator can be promoted to an integer value.
* Converting an integer to an enumerator requires an explicit cast
* The following statement will initialize the enum object mycolor to blue:

mycolor = blue;

**CONSTRUCTORS AND DESTRUCTORS**

**Constructors**

* Are used to initialize the member variables of the class when the objects of the class are created
* Must have the same name as that of class name
* Cannot return any value, not even a void type
* Class can have more than one constructors defined in it (known as overloaded constructors)
* Default constructors accept no parameters and are automatically invoked by the compiler

**Need for Constructors**

* To initialize a member variable at the time of declaration

**Declaration of Constructors**

* Example:

class Calculator

{

private:

int number1, number2, tot;

public:

...

Calculator()

{

number1 = number2 = tot = 0;

}

};

**Destructors**

* Are used to de-initialize the objects when they are destroyed
* Are used to clear memory space occupied by a data member when an object goes out of scope
* Must have the same name as that of the class, preceded by a ~ (For example: ~Calculator())
* Are automatically invoked
* Can also be explicitly invoked when required
* Cannot be overloaded

**Need for Destructors**

* To de-initialize the objects when they are destroyed
* To clear memory space occupied by a data member when an object goes out of scope

**Declaration of Destructors**

* Example:

**/\* This Code Shows The Use Of Destructor In The Calculator Class\* /**

class Calculator

{

private:

int number1, number2, tot;

public:

...

**~**Calculator() **//Body Of The //Destructor**

{

number1 = number2 = tot = 0;

}

};

**POLYMORPHISM**

**Static Polymorphism**

* Refers to an entity existing in different physical forms simultaneously

**Function Overloading**

* Is the process of using the same name for two or more functions
* Requires each redefinition of a function to use a different function signature that is:

different types of parameters,

or sequence of parameters,

or number of parameters

Is used so that a programmer does not have to remember multiple function names

**Constructor Overloading**

* Is commonly used in OOPS

Example:

class Calculator

{

int number1, number2, tot;

public:

Calculator()**//Default Constructor**

{

number1=number2=tot=0;

}

Calculator(int,int)**//Two-Argument Constructor**

{

number1=num1;

number2=num2;

tot=0;

}

void input(int,int){};

void add(){};

void disp(){};

};

**INHERITANCE**

**Relationships**

* Are specified among classes based on the behavior of each class
* Exist between classes because of two reasons:

To indicate some sort of sharing

To indicate some kind of a connection

* Are of four types:

Inheritance

Composition

Utilization

Instantiation

**Inheritance Relationship**

* Between classes or objects means an object or a class inherits a set of attributes from another class
* Example:



* Superclass

Is a class from which another class inherits a set of attributes

* Subclass

Is a class that inherits a set of attributes from another class

The types of inheritance are:

* Single inheritance

Is displayed when a class inherits attributes from a single class

* Multiple inheritance

Is displayed when a class inherits attributes from two or more classes

**Need for Generalization**

* To create programs that are more extensible
* To provide source reusability

**Generalization**

* Is done by clubbing the structure and behavior that is common to all the classes
* Is represented by the superclass
* Is implemented by using the Abstract class, which

Is not an existent entity

Is a base from which other classes inherit attributes

**Composition Relationship**

* Occurs when one class is made up of another
* Example:

Relationship between a car and its parts like engine, doors, steering wheel, gear box, seats, and so on

**Utilization Relationship**

* Exists between two or more classes, which make use of other classes
* Example:

Relationship between a driver and a car

**Instantiation Relationship**

* Is a relationship between a class and an instance of that class
* Example:

Relationship between a book and ‘Gone with the Wind’

**Uses of Inheritance**

* Reduces redundancy in code
* Enables easy maintenance of code
* Extends the functionality of an existing class by adding functions to the subclass

**Sequence of Invoking Constructors and Destructors**

* Constructors are called in the order of Base - to - Derived
* Destructors are called in the order of Derived - to - Base
* **DYNAMIC POLYMORPHISM**
* *Dynamic polymorphism*refers to an entity changing its form depending on the circumstances. A function is said to exhibit dynamic polymorphism when it exists in more than one form, and calls to its various forms are resolved dynamically when the program is executed.
* *Binding* is the process of associating a function with a class by identifying the type of the object or pointer that is used to invoke the function.
* The term *late* *binding* refers to the resolution of the functions at run-time instead of compile time. This feature increases the flexibility of the program by allowing the appropriate method to be invoked, depending on the context.
* *Virtual Function* is a function in base class, which is overrided in the derived class, and which tells the compiler to perform Late Binding on this function.

Implementation of Virtual Function in Derived class

**Static vs Dynamic Polymorphism**

* Static polymorphism

Is considered more efficient

Is function overloading, which is bound by scopes - local and class scope

* Dynamic polymorphism

Is considered more flexible

Is based on overriding principles, which, therefore, is purely class scope and is based on inheritance

**Multiple Inheritance**

* Is the phenomenon where a class may inherit from two or more classes