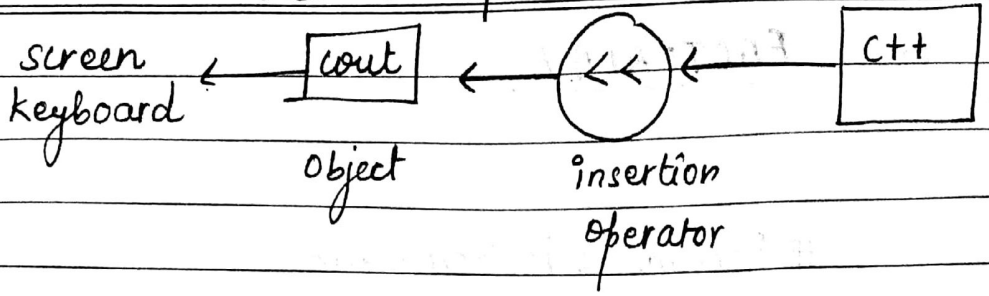
Ques1. Difference between C and C++

	C	C++
1.	C is invented by Denis Ritchi	C++ is invented by BJarne Stroustrup.
2.	In C top to bottom approach is followed.	C++ follows bottom to top approach.
3.	C is based on procedure oriented approach.	C++ is based on object oriented approach.
4.	C does not support Encapsulation	C++ supports Encapsulation.
5.	C does not support inheritance.	C++ supports inheritance.

	C	C++
6.	C does not support polymorphism	C++ supports polymorphism.
7.	C uses malloc and calloc for dynamic memory allocation.	C++ uses new and delete operator for dynamic memory allocation.
8.	C uses printf and scanf function for output and input.	C++ uses cin and cout operator for output and input.
9.	C does not have the property of friend function.	C++ has the property of friend function.

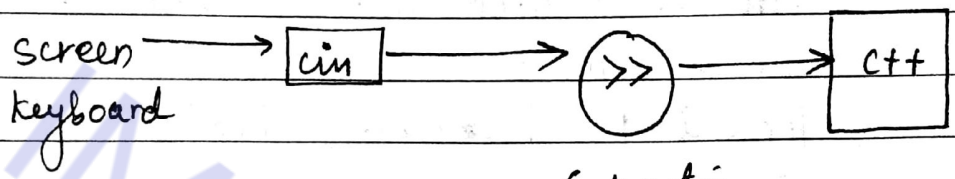
Output operator

cout << a → insertion operator



Input operator

cin (>>) a → Extraction operator.



1.
```
#include <iostream.h>
#include <conio.h>
void main()
{
  cout << "Hello";
}
```

AVERAGE

2.
```
#include <iostream.h>
#include <conio.h>
void main()
{
  int a, b, avg;
  cin >> a >> b;
  avg = (a+b)/2;
  cout << avg;
}
```

Page No.
Date

Features of C++

(1) **Encapsulation** : wrap up data into a single unit. → car

```
class      classname
Keyword    (user-defined)
{
  /variables/*;
  /functions/*;
};
```

(2.) **Abstraction** : hiding the complexity and showing the useful parts.

(3.) **Inheritance** : Public, Private, Protected } Access specifier

(4.) **Polymorphism** : many - forms.

Page No.
Date

ARMSTRONG NUMBER

```
main ()
{
  int orignum, num, rem, sum = 0
  cout << "Enter no. ";
  cin >>  orig num;
  num=  orig num;
  while (num != 0)
  {
   sum =  num % 10 ;
   sum =  sum + rem * rem* rem;
   num =  num /10;
  }
   If (sum == orignum)
  cout << orignum << is Armstrong no";
  else
  cout << orignum << "NO Armstrong no" << "\n";
}
```

Page No.
Date

Write a program to print "I am a student of BCA" ?

```
#include <iostream.h>
#include <conio.h>
void main
{
  cout << "I am a student of BCA";
  getch();
}
```

2 July

OOPS - Object Oriented Programming system.

* uses of Objects and classes.

Features of OOPs ——— Encapsulation ✓
Abstraction ✓
Inheritance ✓

Base (Parent) — Child

Polymorphism.

many forms

Eg: operators, functions.

Dynamic Binding = Dynamic binding means that the code which is associated with a given procedure call is not known until the time of call at runtime.

It is also call late binding

Date

CONCEPTS OF Object Oriented programming System

* It uses objects in programming and aims to implement real world entities like inheritance, polymorphism etc.

* **Heading object**

These are basic run time entities in OOPS. Where objects are instances of a class. and provides its reference. It is user defined.

* **Class**

It is also user defined and has a collection of data and function

* **Encapsulation**

It is the concept of wrapping the data and functions into a single unit. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it ...

Data Abstraction

It refers to providing only needed information

to the outside world and hiding implementation details.

The advantage of abstraction is we can change implementation at any point of time but the user wont get affected because the interface remains same.

Inheritance

It is the process by which objects of one class acquires the property of objects in another class.

It supports the concept of hierarchical classification.

It provides reusability to the data and functions.

Polymorphism

It means ability to take more than one form.

It supports operator overloading and (C++) function overloading which means every operator except a few in C++ can be overloading and functions can also be overloaded.

1. Define functional decomposition, data decomposition.

2. Define attributes of class and object.

3. What is default parameter value

4. Define Abstract data type with example.

5. Define Dynamic memory allocation.

Ans 1. **FUNCTIONAL DECOMPOSITION**

* Functional decomposition is the process of taking a complex process and breaking it down into its smaller simpler parts.
* Functional decomposition is the process of modularising the program into different modules.
* It makes our program more effective and efficient..
* Functional decomposition make our program short and decrease the complexity of the program.
* It provides reusability of code.

Data Decomposition

* Data decomposition is the process of breaking down of data into simpler parts or smaller simpler parts.
* Data decomposition is the process of removing or reducing the complexity of the program
* Data decomposition makes our program effective and efficient...

Page No.
Date

Ans2. ATTRIBUTES OF CLASS AND OBJECT

According to OOPs concept

* Class is a collection of object.
* A class is the logical entity.
* Class is a blueprint of the object.
* It is also the prototype.
* We can declare all the data members and member function inside a class.
* Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties of a "CLASS".

OBJECT

* An object is called the instance of a class. When a class is defined, no memory is allocated but when an object is created memory is also allocated.
* An object is a entity which has its own attributes and properties.

Ans 3. DEFAULT PARAMETER VALUE

* A default argument is a function argument that has a default value provided to it.
* If the user dont' supply a value for this argument, the default value will be used.
* If the user does supply a value for the default argument, the user-supplied value is used.

Eg: int add (int x, int y = 20, int z = 30);
int add (int x = 20, int y = 10, int z = 30);

Ans 4. ABSTRACT DATA TYPE AND EXAMPLE

Abstract data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and set of operations.

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.

Example of Abstract data type are:

LIST: (1) insert()
(2) get()
(3) remove()
(4) removeAt()

STACK: push()
pop()

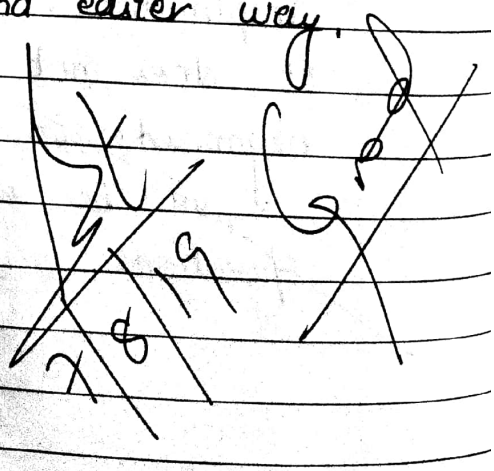
QUEUE: size()
peek()

Ans 5. DYNAMIC MEMORY ALLOCATION

* Dynamic memory allocation refers to performing memory allocation manually by programmer.
* Dynamically allocated memory is allocated on Heap and non-static and local variables get memory allocated on stack.

According to c dynamic memory allocation is done using malloc()
calloc()
free()

But c++ uses new and delete operators that perform the task of allocating and freeing the memory in a better and easier way.



➢ UNIT-I

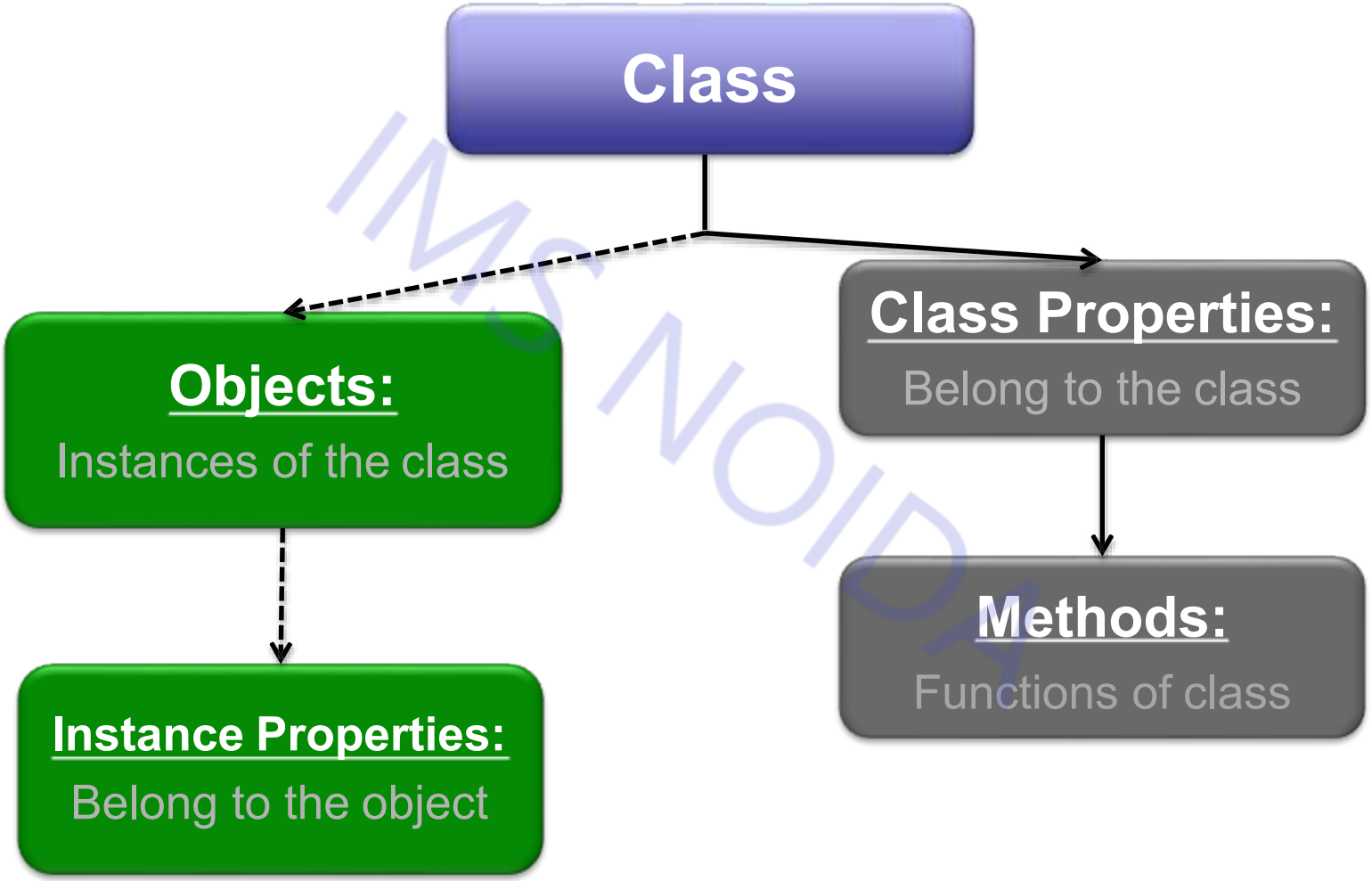
Introduction Introducing Object – Oriented Approach, Relating to other paradigms {Functional, Data decomposition}. Basic terms and ideas Abstraction, Encapsulation, Inheritance, Polymorphism, Review of C, Difference between C and C++ - cin, cout, new, delete, operators.

- Introduction
  - A Brief Programming History
  - OOP
  - Classes & Objects

Classes and Objects

- A *class* is a prototype, idea, and blueprint for creating objects.
- An object is an instance of a class.
- For example, in Java we define classes, which in turn are used to create objects
- A class has a *constructor* for creating objects
- *Class* is composed of three things: its name, attributes/properties, and methods.

Classes *(objects)*

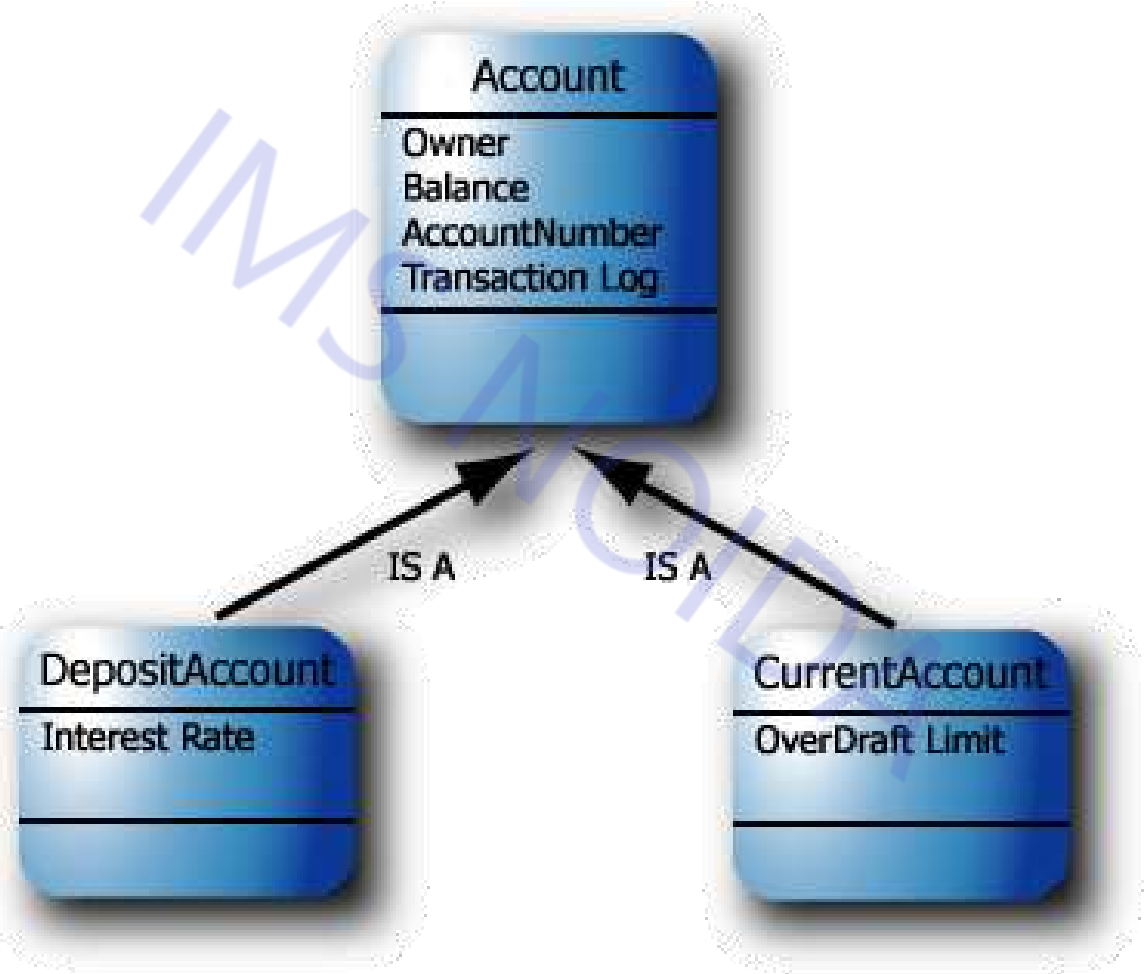


Classes & Objects

A ***class*** is a definition of objects with the same properties and the same methods.

```
class Bicycle {

       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
```

Classes Example



Almost everything in the world can be represented as an object

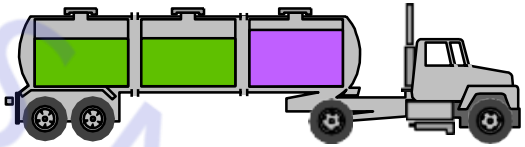
- A flower, a tree, an animal
- A student, a professor
- A desk, a chair, a classroom, a building
- A university, a city, a country
- The world, the universe
- A subject such as CS, IS, Math, History, …
- An information system, financial, legal, etc..

What Is an Object, again?

An object is an instance of a class

More about objects

- Informally, an object represents an entity, either physical, conceptual, or software.
  - Physical entity

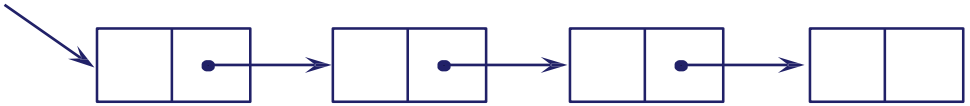


Truck

  - Conceptual entity

Chemical Process

  - Software entity



Linked List

More formal definition of an “Object”

An object is a computational entity that:

1. *Encapsulates* some state
2. Is able to perform actions, or *methods*, on this state
3. Communicates with other objects via *message passing*

Class/Object

Each copy of an object from a particular class is called an ***instance*** of the class.

Class/Object

The act of creating a new instance of an class is called **instantiation.**

In short…

- An Object is a Class when it comes alive!
- Homo Sapien is a class, John and Jack are objects
- Animal is a class “Snowball” the cat is an object
- Vehicle is a class My neighbor's BMW is an object
- Galaxy is a class, the MilkyWay is an object

Technical contrast between Objects & Classes

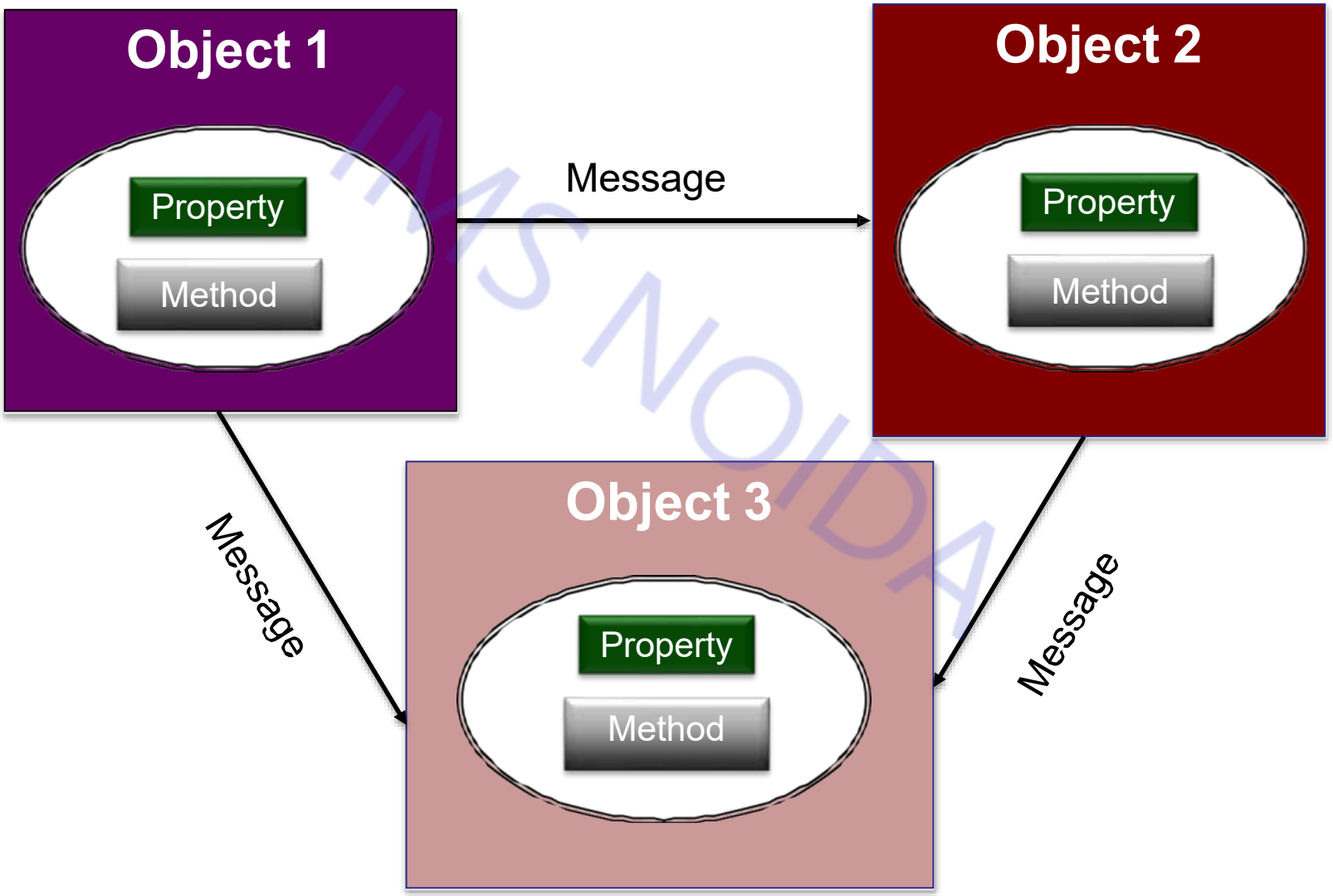
CLASS	OBJECT
Class is a data type	Object is an instance of Class.
It generates OBJECTS	It gives life to CLASS
Does not occupy memory location	It occupies memory location.
It cannot be manipulated because it is not available in memory *(except static class)*	It can be manipulated.

Object is a class in *"runtime"*

Objects Need to Collaborate!

- Objects are useless unless they can collaborate together to solve a problem.
  - Each object is responsible for its own behavior and status.
  - No one object can carry out every responsibility on its own.
- How do objects interact with each other?
  - They interact through messages.

Object Interaction



Introducing Object- Oriented Approach

By:- Ms. Priti Rani Rajvanshi
Assistant Professor-IT
IMS NOIDA

- Introduction
  - A Brief Programming History
  - OOP
  - Classes & Objects

Computer Programming

An ***algorithm*** is a step-by-step process.

A ***computer program*** is a step-by-step set of instructions for a computer.

Every computer program is an algorithm.

Computer Programming

The history of computer programming is a steady move away from ***machine-oriented views*** of programming towards concepts and metaphors that more closely reflect the way in which we ourselves see & understand the world

Programming Languages

- Programming languages allow programmers to develop software.
- The three major families of languages are:
  - Machine languages
  - Assembly languages
  - High-Level languages

Machine Languages

- Comprised of 1s and 0s
- The ***"native" language*** of a computer
- Difficult to program – one misplaced 1 or 0 will cause the program to fail.
- Example of code:

```
1110100010101        1110101011110
10111010110100       10100011110111
```

Assembly Languages

- Assembly languages are a step towards easier programming.
- Assembly languages are comprised of a ***set of elemental commands*** which are tied to a specific processor.
- Assembly language code needs to be translated to machine language before the computer processes it.
- Example:

```
ADD  1001010, 1011010
```

High-Level Languages

- High-level languages represent a giant leap towards easier programming.
- The syntax of HL languages is similar to English.
  - Example:

    ```
    grossPay = basePay + overTimePay
    ```

  - Interpreter – Executes high level language programs without compilation.
- Historically, we divide HL languages into two groups:
  - *Procedural languages*
  - *Object-Oriented languages (OOP)*

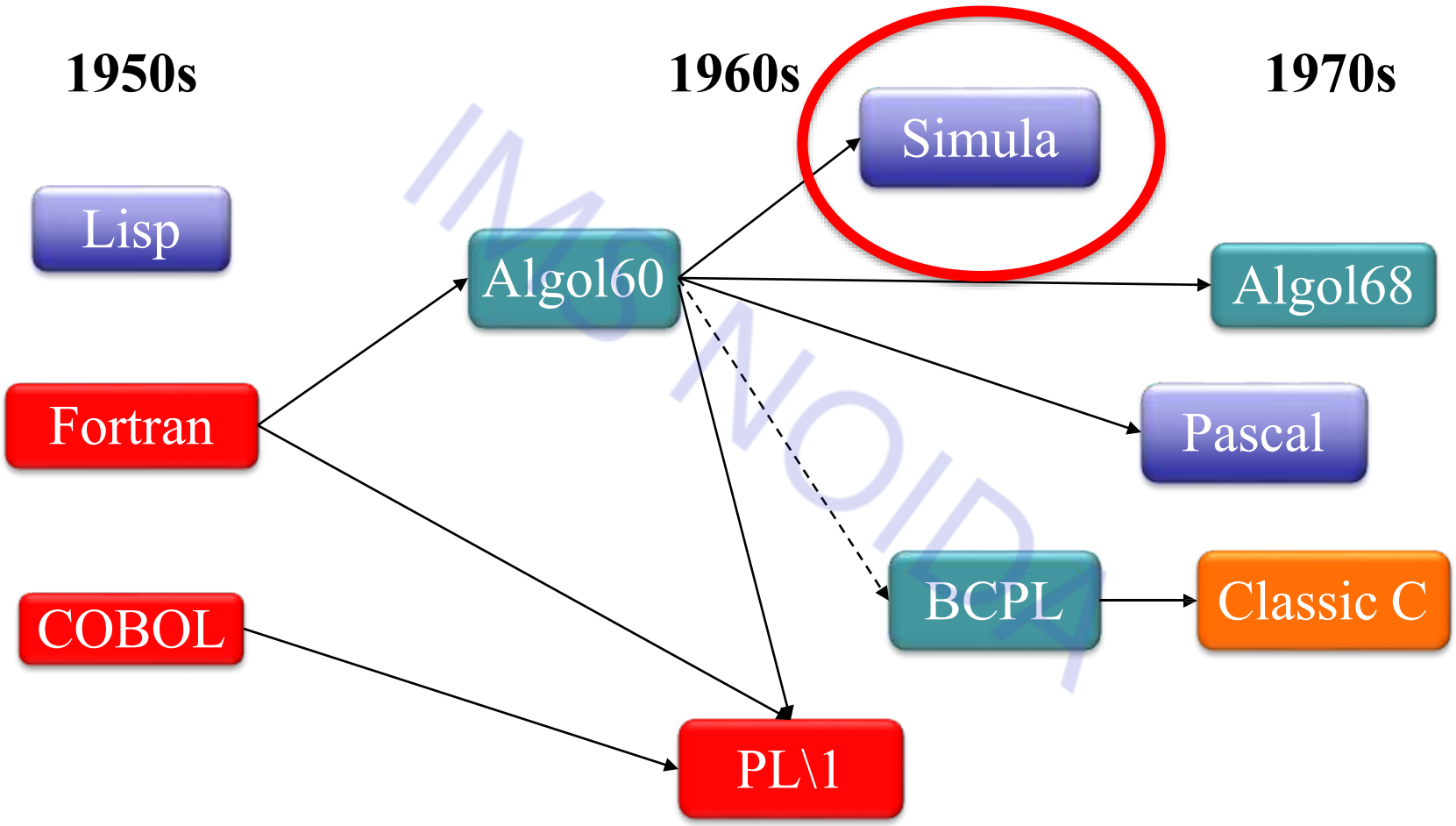
Procedural Languages

- Early high-level languages are typically called procedural languages.
- Procedural languages are characterized by sequential sets of linear commands. The focus of such languages is on *structure*.
- Examples include C, COBOL, Fortran, LISP, Perl, HTML, VBScript

Object-Oriented Languages

- The focus of OOP languages is not on structure, but on *modeling data*.
- Programmers code using "blueprints" of data models called *classes*.
- Examples of OOP languages include C++, Visual Basic.NET and Java.

Early programming languages



Red==major commercial use
Blue==will produce important "offspring"

- Simula is a name for two simulation programming languages, Simula I and Simula 67, developed in the 1960s at the Norwegian Computing Center in Oslo, by Ole-Johan Dahl and Kristen Nygaard.
- Simula is considered the first object-oriented programming language.
- Simula was designed for doing simulations, and the needs of that domain provided the framework for many of the features of object-oriented languages today.

– A Brief Programming History
– OOP
– Classes & Objects

OOP

- *OOP* is mainly a program design philosophy.
- OOP uses a different set of programming languages than old procedural programming languages (*C, Pascal*, etc.).
- Everything in *OOP* is grouped as self sustainable "*objects*". Hence, you gain re-usability by means of four main object-oriented programming concepts.

OOP

- In OOP programmers define not only the data type of a data structure, but also the types of operations/methods *(functions)* that can be applied to the data structure.
- In this way, the data structure becomes an object that includes both data and functions *(methods)* in one unit. In addition, programmers can create relationships between one object and another.
- For example, objects can inherit characteristics from other objects.

Object-Oriented Programming Languages

- Pure OO Languages

Eiffel, Actor, Emerald, JADE, Obix, Ruby, Python, Scala, Smalltalk, Self.

- Hybrid OO Languages

Delphi/Object Pascal, C++, Java, C#, VB.NET, Pascal, Visual Basic, MATLAB, Fortran, Perl, COBOL 2002, PHP, ABAP, Ada 95.

OOP

Key idea in object-oriented:

The real world can be "accurately" described as a collection of objects that interact.

OOP Basic Terminology

Object

- usually a person, place or thing (**a noun**)

Method

- an action performed by an object (**a verb)**

Property or attribute

- Characteristics of certain object.

Class

- a category of similar objects (such as *automobiles), does not hold any values of the object's attributes/properties*

Some Common Paradigms

You should know these:

Imperative:

Programming with an explicit sequence of commands that update state.

Declarative:

Programming by specifying the result you want, not how to get it.

- **Logic** (Rule-based):

  Programming by specifying a set of facts and rules. An engine infers the answers to questions.

- **Constraint**:

  Programming by specifying a set of constraints. An engine finds the values that meet the constraints.

- **Aspect-Oriented**:

  Programming cross-cutting concerns applied transparently

.

- **Structured**:

  Programming with clean, goto-free, nested control structures.

- **Procedural**:

Imperative programming with procedure calls.

- **Functional** (Applicative):

Programming with function calls that avoid any global state.

- **Function-Level** (Combinator):

  Programming with no variables at all.

- **Object-Oriented**:

Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces. Object orientation can be:

  - **Class-based**: Objects get state and behavior based on membership in a class.
  - **Prototype-based**: Objects get behavior from a prototype object.

- **Event-Driven**:

Programming with emitters and listeners of asynchronous actions.

- **Flow-Driven**:

Programming processes communicating with each other over predefined channels.

- **Array**:

  Programming with powerful array operators that usually make loops unnecessary.

Paradigms are **not meant to be mutually exclusive**; a single program can feature multiple paradigms!

- OOP Basic Concepts
  - Encapsulation
  - Inheritance
  - Abstraction
  - Polymorphism

Encapsulation

Encapsulation

- Is the inclusion of property & method within a class/object in which it needs to function properly.
- Also, enables reusability of an *instant* of an already implemented class within a new class while ***hiding & protecting*** the method and properties from the client classes.

Encapsulation

- The class is kind of a container or capsule or a cell, which encapsulate the set of methods, attributes and properties to provide its indented functionalities to other classes.
- In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system.
- That idea of encapsulation is to hide how a class does its operations while allowing requesting its operations.

Encapsulation in action

Example:

- Let's say we have a class called "Date" (day, month, year). And then you need to define another class called "Person" that has the following attributes (first name, last name, and birthdate). So in this case we can instantiate an object from class "Date" inside class "Person".

Encapsulation –Benefits

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

Encapsulation allows adding some logic when accessing client's data

- E.g. validation on modifying a property value

Hiding implementation details reduces complexity → easier maintenance

44

- OOP Basic Concepts
  - Encapsulation
  - Inheritance
  - Abstraction
  - Polymorphism

Inheritance

- *Inheritance*—a way of organizing classes
- Term comes from inheritance of traits like eye color, hair color, and so on.
- Classes with properties in common can be grouped so that their common properties are only defined once in parent class.
- *Superclass* – inherit its attributes & methods to the subclass(es).
- *Subclass* – can inherit all its superclass attributes & methods besides having its own unique attributes & methods.

Inheritance

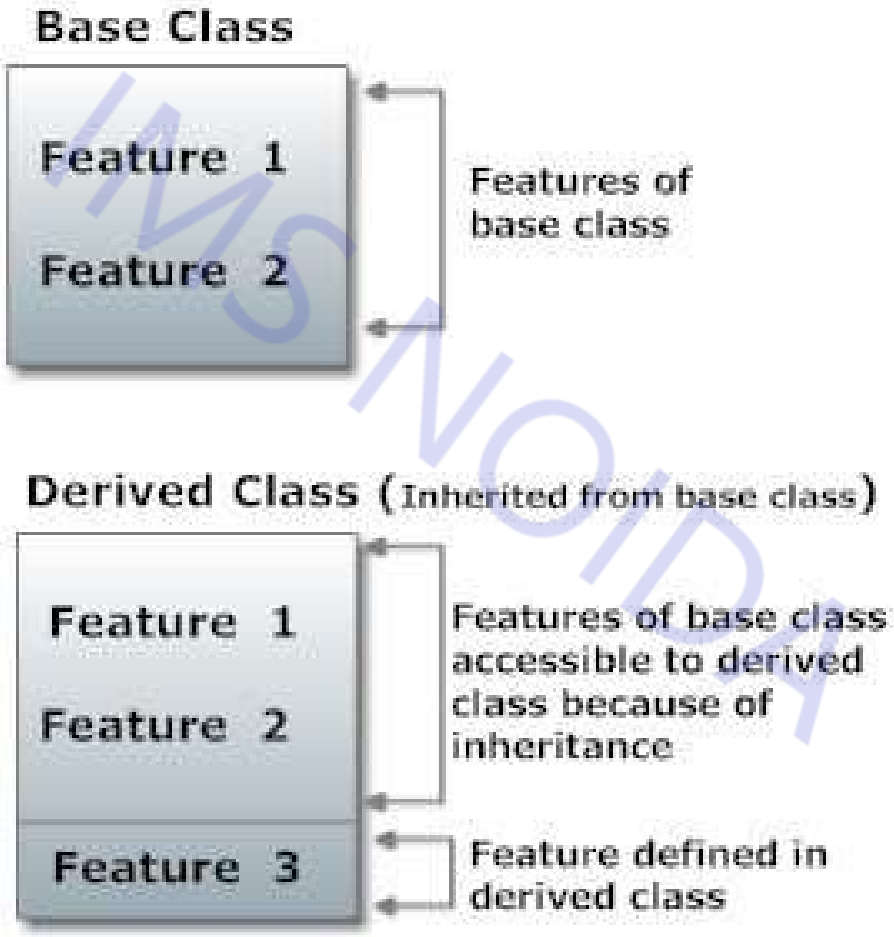
- Inheritance allows child classes to inherit the characteristics of existing parent class
  - Attributes (fields and properties)
  - Operations (methods)
- Child class can extend the parent class
  - Add new fields and methods
  - Redefine methods (modify existing behavior)
- A class can implement an interface by providing implementation for all its methods

47

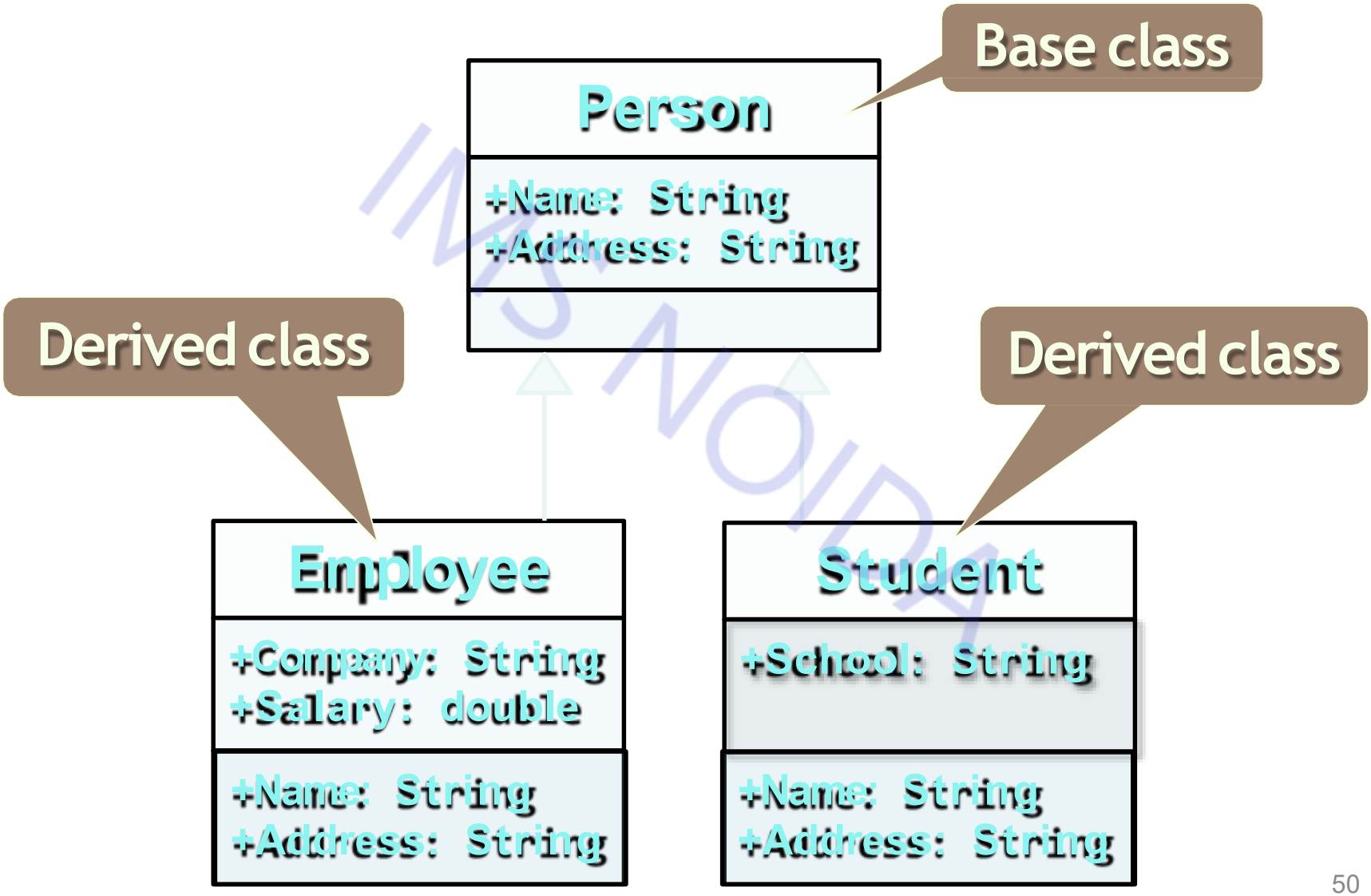
Inheritance

- Expresses commonality among classes/objects
- Allows code reusability
- Highlights relationships
- Helps in code organization

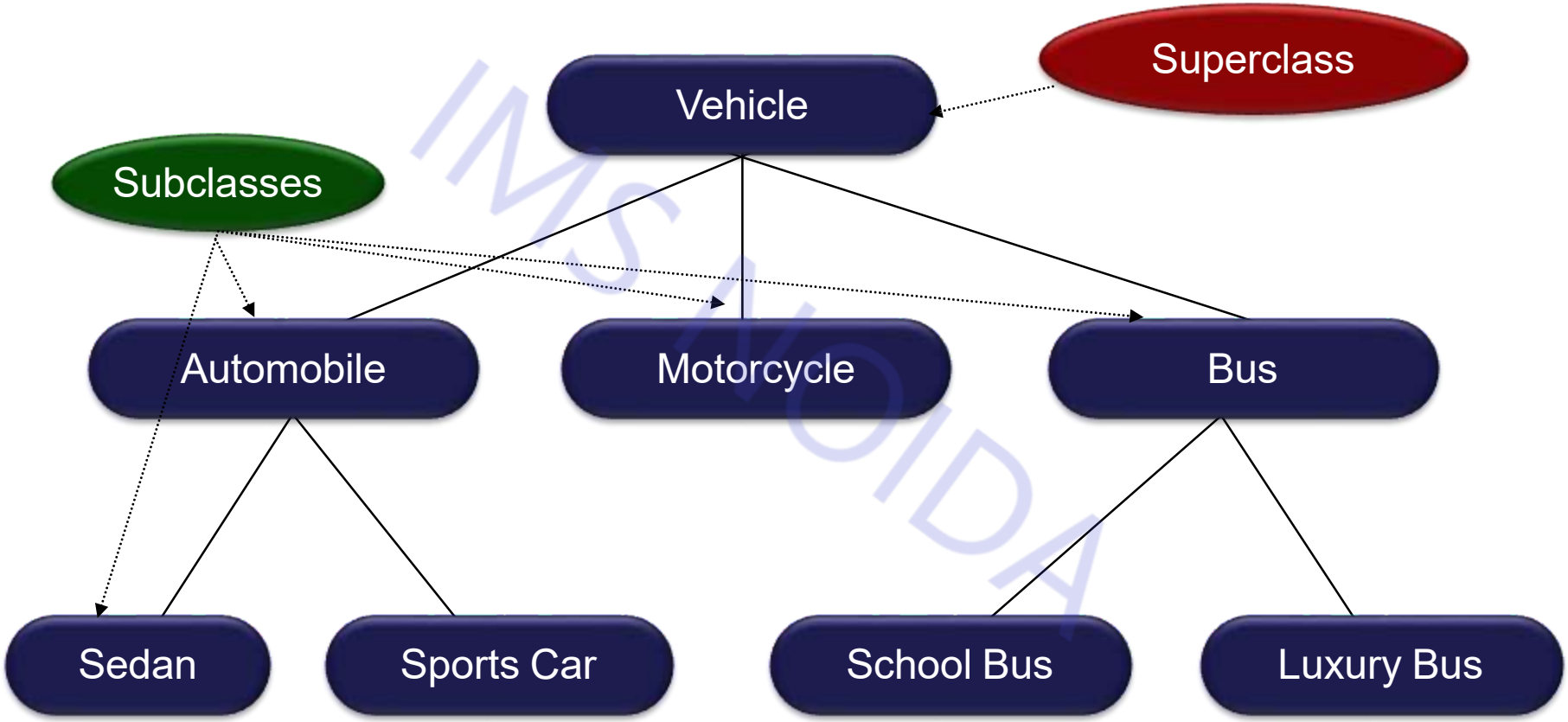
Inheritance



Inheritance - Example

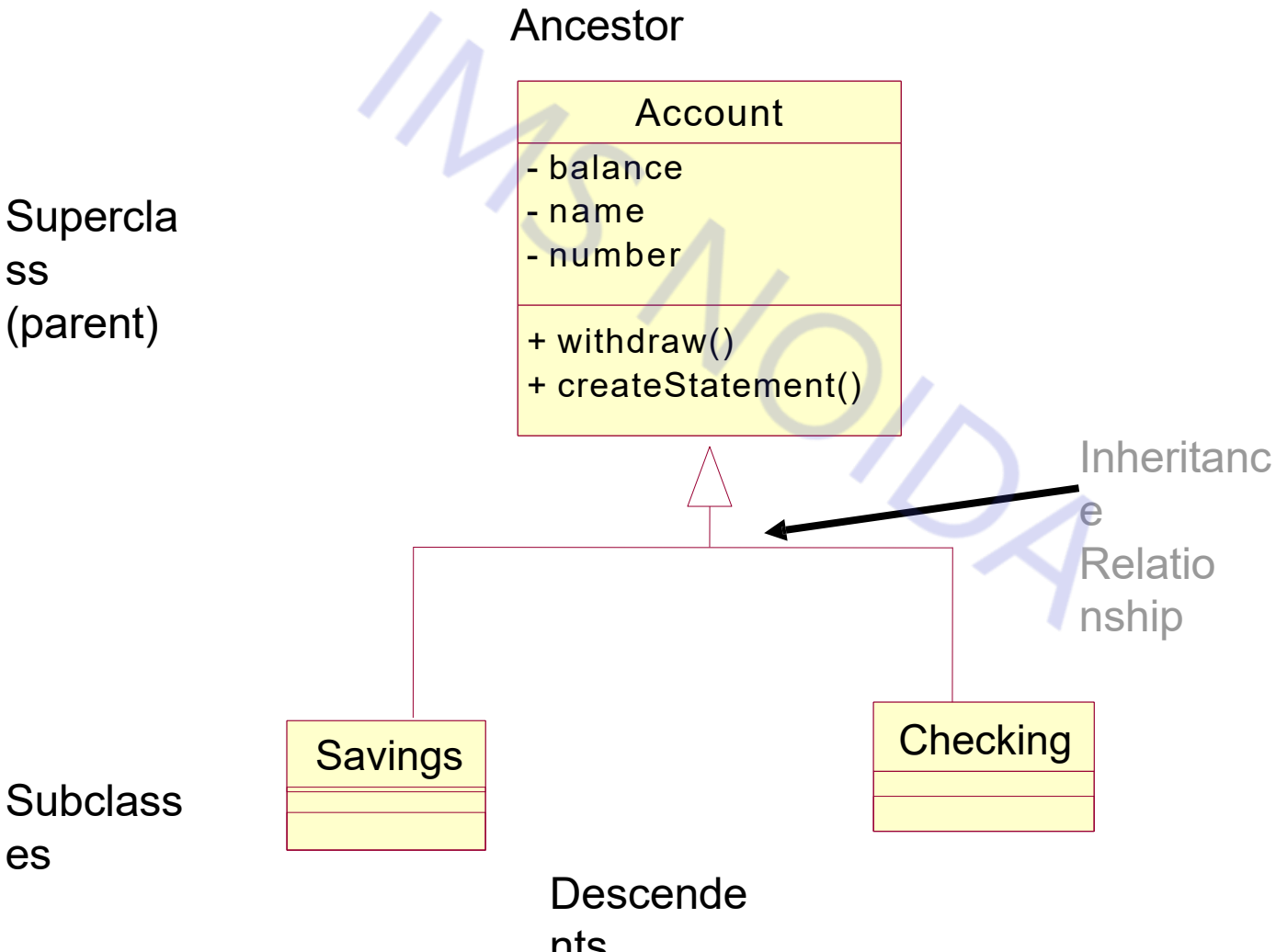


An Inheritance Hierarchy



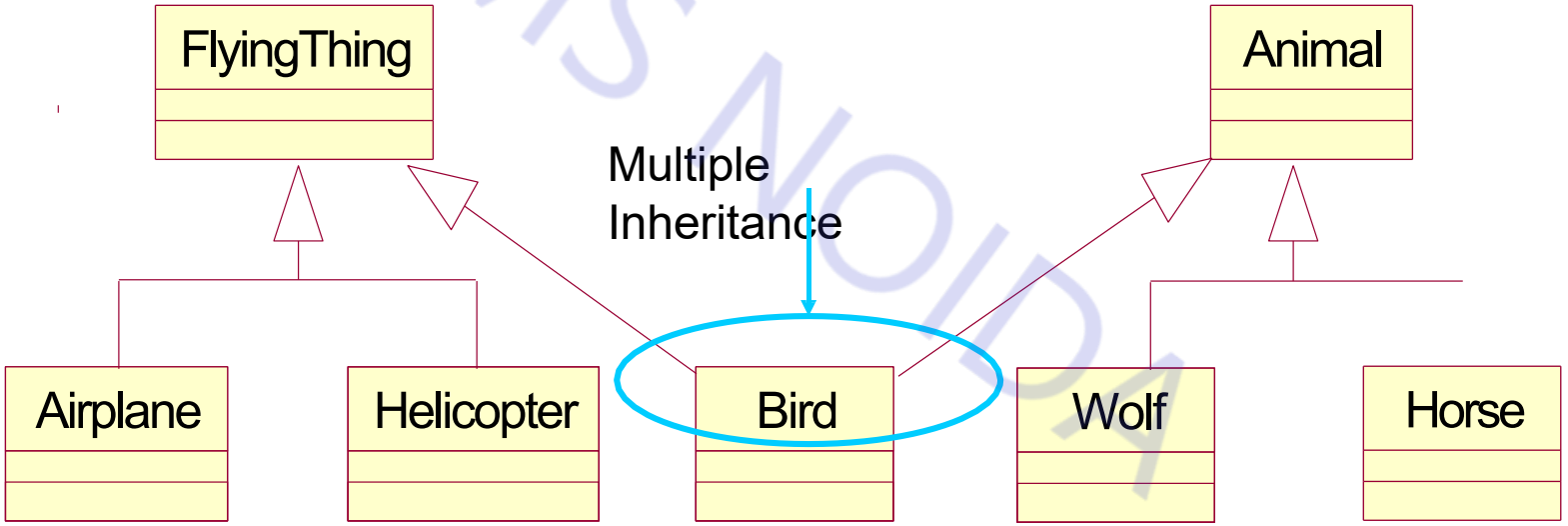
Example: Single Inheritance

One class inherits from another.



Example: Multiple Inheritance

- A class can inherit from several other classes.



Most modern languages don't support multiple inheritance!

- OOP Basic Concepts
  - –Encapsulation
  - – Inheritance
  - – Abstraction
  - – Polymorphism

Type of Classes

Can be instantiated directly

Can't be instantiated directly

Abstraction

- Abstraction is a design principle.
- Is the process of removing characteristics from something in order to reduce it to a set of essential characteristics.
- Through the process of abstraction, a programmer hides all but the relevant data about a class in order to reduce complexity and increase reusability.
- Abstraction is a basic representation of a concept.

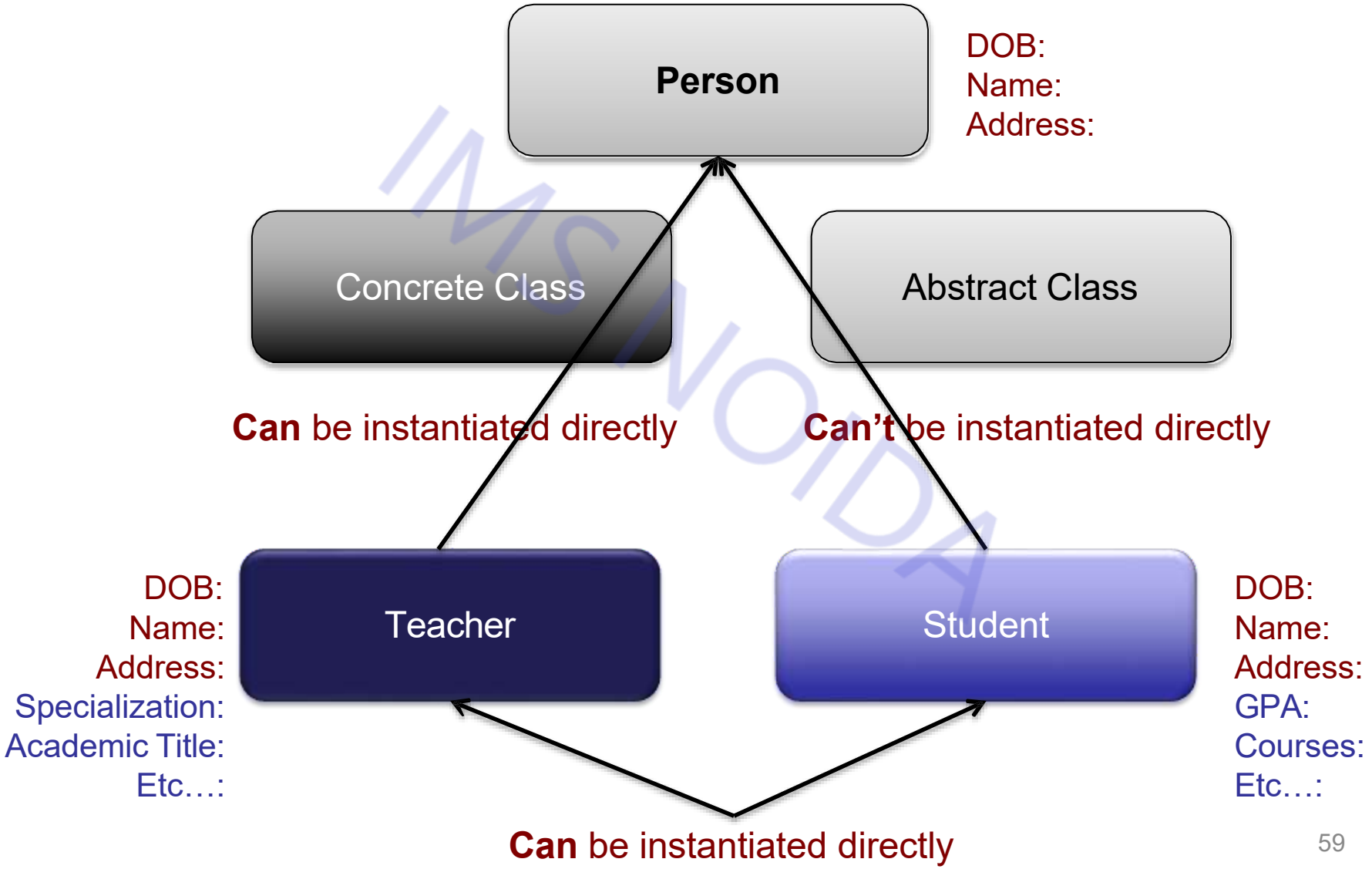
Abstraction

- Abstraction allows programmers to represent complex real world in the simplest manner.
- It is a process of identifying the relevant qualities and behaviors an object should possess, in other word represent the necessary features without representing the back ground details
- You should always use abstraction to ease reusability, and understanding for the design and enable extension.
- When we design the abstract classes, we define the *framework* for later extensions.

Abstraction

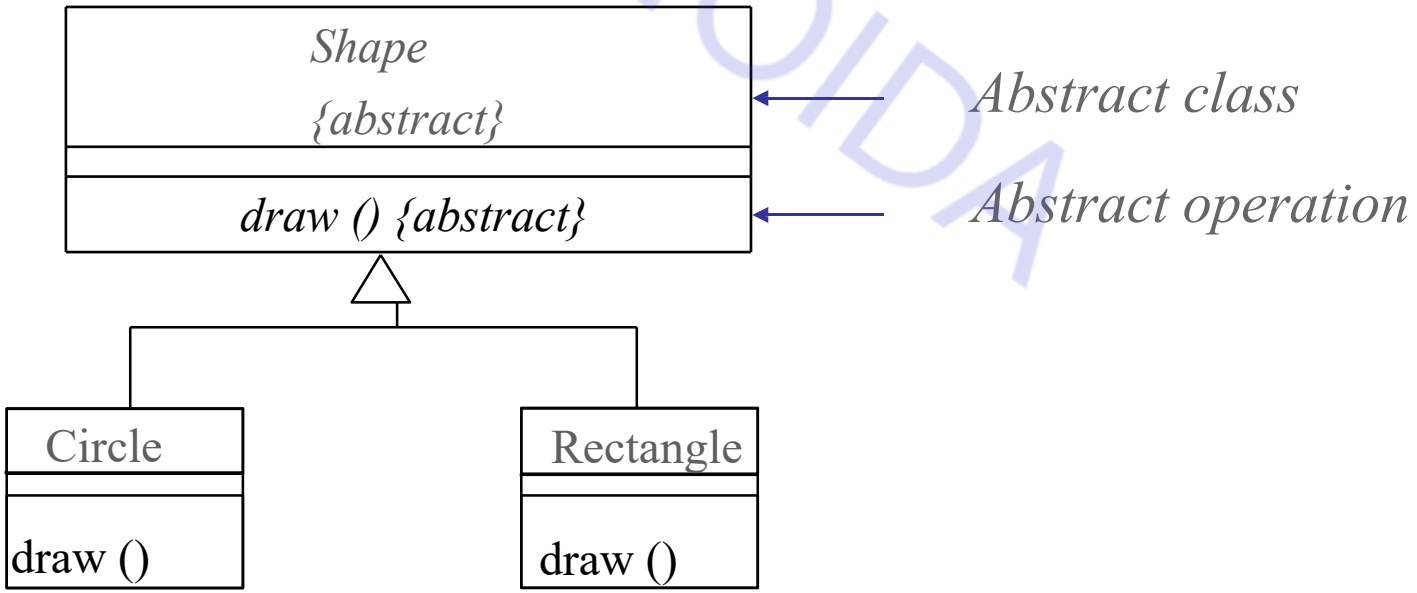
- An abstract class, which declared with the *"abstract"* keyword, cannot be instantiated.
- It can only be used as a super-class for other classes that extend the abstract class. Abstract class is a design concept and implementation gets completed when it is being realized by a subclass.

Abstraction - type of classes

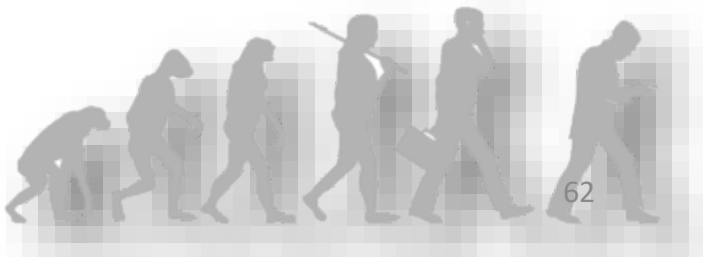


Abstraction

- An ***abstract class*** is a class that may not have any direct instances.
- An ***abstract operation*** is an operation that it is incomplete and requires a child to supply an implementation of the operation.



- OOP Basic Concepts
  - Encapsulation
  - Inheritance
  - Abstraction
  - Polymorphism



Polymorphism

- Encapsulation, Inheritance, and Abstraction concepts are very related to Polymorphism.

Polymorphism

- Polymorphisms is a generic term that means 'many shapes'. More precisely *Polymorphisms* means the ability to request that the same *methods* be performed by a wide range of different types of things.
- In *OOP*, *polymorphisms* is a technical issue and principle.
- It is achieved by using many different techniques named method overloading, operator overloading, and method overriding.

Polymorphism

- An object has “multiple identities”, based on its class inheritance tree
- It can be used in different ways

Polymorphism

- In Java, two or more classes could each have a method called **`output`**
- Each **`output`** method would do the right thing for the class that it was in.
- One **`output`** might display a number (output.number) in one class, whereas it might display a name (output.text) in another class.

Polymorphism

- It is the ability to look at a class in its parent image.
- Lets see the robot example throughout the following few slides

Polymorphism- Abstract class, again!!!!

- It is a class that you cannot instantiate from, however, you use it to dominate and specify how the minimum requirements in an inherited classes should be.

```
public abstract class Robot
{
  public virtual abstract void Move()
// abstract move needs
}
```

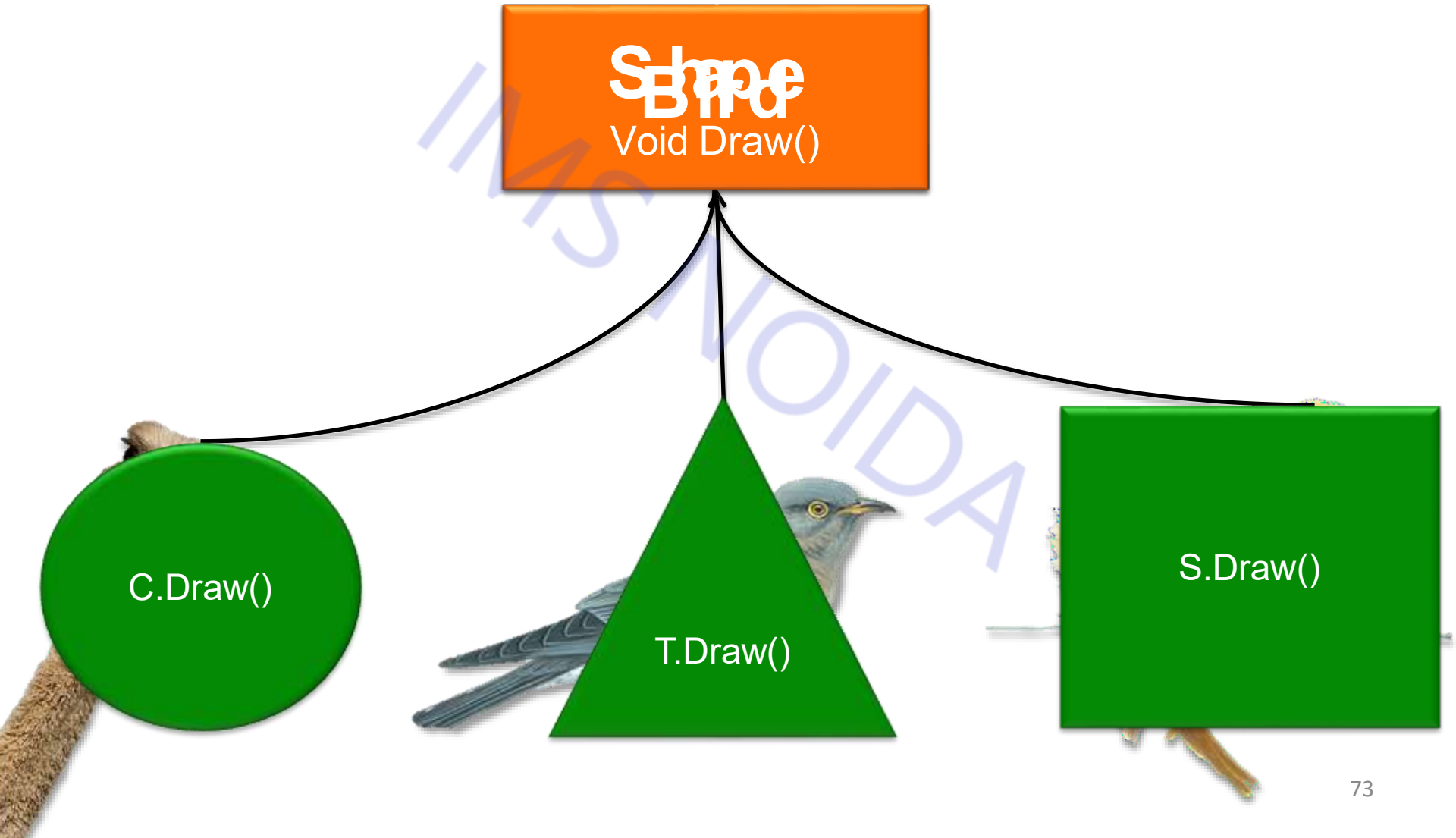
```
public class LeggedRobot:Robot
{
  public override void Move()
   {
     // actions of legged robot to move
   }
}
```

```
public class WheeledRobot:Robot
{
  public override void Move()
   {
     // actions of Wheeled robot to move
   }
}
```

```
public void moveRobot(Robot A)
{
  A.Move();
}
```

```
public moveAllRobots()
{
  LeggedRobot lr = new LeggedRobot();
  WheeledRobot wr = new WheeledRobot();

  moveRobot(lr);  moveRobot(wr);
}
```

Polymorphism



Advantages of OOP

- Code reuse & recycling
- Improved software-development productivity
- Improved software maintainability
- Faster development
- Lower cost of development
- Higher-quality software
- Encapsulation

Disadvantages of OOP

- Steep learning curve
- Could lead to larger program sizes
- Could produce slower programs

OOP Suitability

- Object oriented programming is good in complex projects or modular type of systems. It allows simultaneous system development teams and also could aid in agile system development environments like Xtreme Programming.