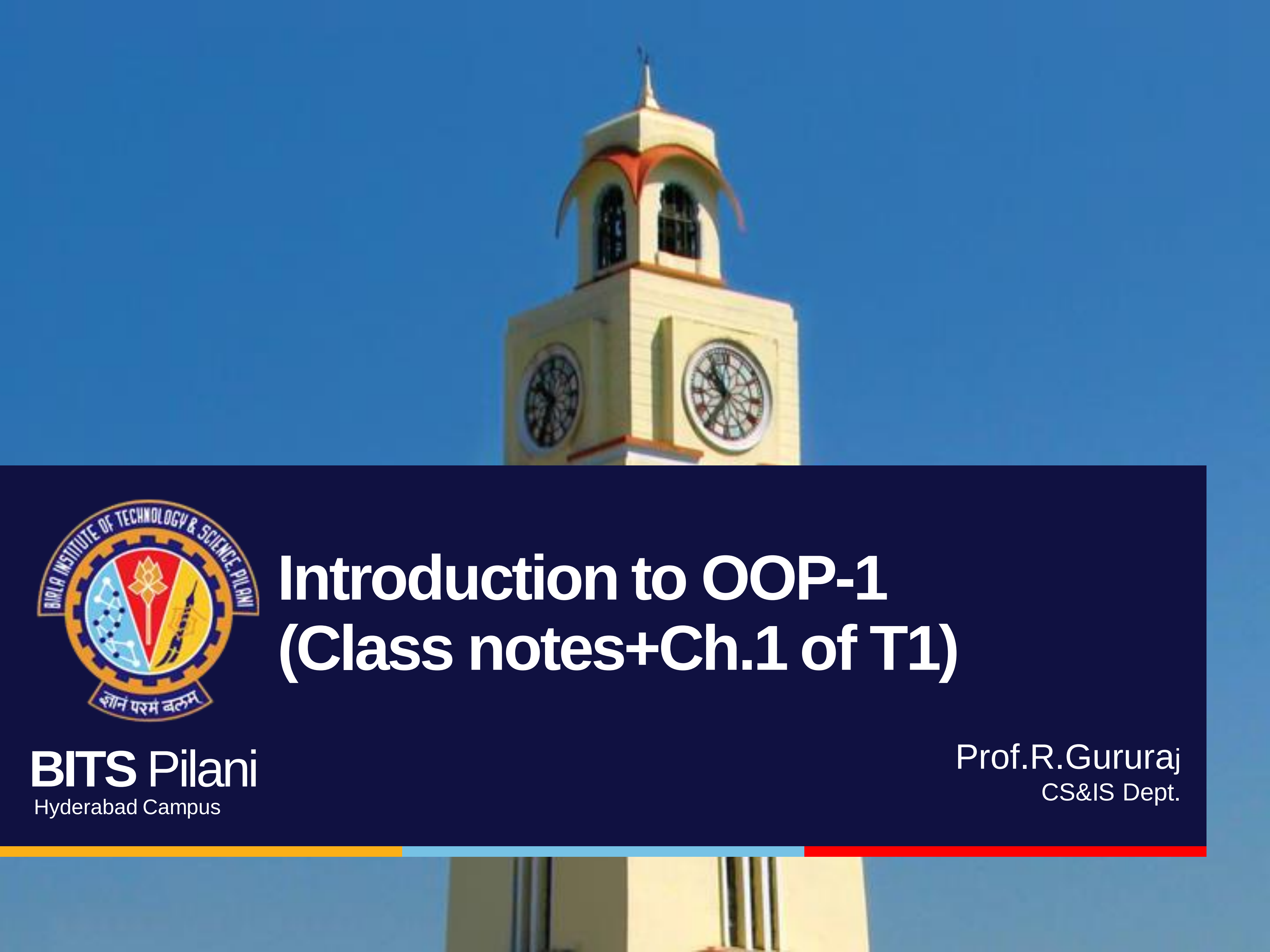
CS F213
Object Oriented Programming

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.



Introduction to OOP-1 (Class notes+Ch.1 of T1)

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Objectives of this course

- ➢ To gain an understanding of the need for Object Oriented Paradigm.
- ➢ Understand the foundations of Object Orientation
- ➢ To gain knowledge on important features of Object Orientation with the help of Java (through hands-on lab experience) including Multithreading, Exception Handling and Input/Output.
- ➢ To make the student understand how GUI applications can be developed with Java with sufficient hands-on.
- ➢ To gain basic knowledge on Object Oriented Design methodology, UML modeling and Design patterns.

Computers

We use computer to solve problems.

What are steps followed?

1. Problem statement
2. Algorithm
3. Program
4. Machine instructions
5. Execution

Computer Languages

Why languages?

What are the types?

- Low-level Language (LLL)

  Low-level Language is tied to the computer hardware (machine dependent).

- High-level Language (HLL)

  Each computer (HW) will have one such low-level language we call such language as Assembly Language (in addition to its Machine Language).

HLL vs LLL

High-level Languages	Low-level Languages
Machine independent	Machine dependent
Closer to programmer	Closer to the machine
Easy to write code	Difficult
Needs great effort for translation	Needs minimal effort
Programmer need not know the internals of the HW	Programmer should understand the internals
Less chance for errors	Error prone
Programmer can focus more on solution	Programmer needs to spend more time on understanding LLL specifics
More readable	Less readability

High-level statement

C = A + B to add A, and B and assign the sum to C, is obviously much more readable, quicker to write and less error-prone. But needs translation into machine code. Compilers will do that.

Summary

- High-level languages allow us to use symbolic names for values.
- High-level languages provide expressiveness.
- High-level languages enhance readability.
- High-level language provide abstraction of the underlying HW.
- High-level languages safeguard against bugs.

BASIC, PROLOG, LISP

C, C++, Cobol,

FORTRAN, Java,

Pascal, Perl, PHP, Python,

Ruby, and Visual Basic and many more.

How many PLs we have?

Types of Languages

1. Procedural Programming Language. (C)
2. Functional Programming Language. (LSP)
3. Scripting Programming Language. (PERL)
4. Logic Programming Language. (PROLOG)
5. Object-Oriented Programming Language. (Java)

Setting Context

We write programs in some language to give instructions to the computer to perform a specific task (functionality).

Now we consider High-level programming languages only.

All of us have good acquaintance with C-Language.

C-language is a high level language.

Sample C Program

```
/* INVESTMENT PROBLEM */
/* Written by Dr. R. Gururaj*/
#include <stdio.h>
#define PERIOD  10                    /*Symbolic constant */
#define PRINCIPAL  5000.00      /*Symbolic constant */
void main( )
{     intyear;float amount, value, inrate;
      number=100; amount=PRINCIPAL;
      inrate=0.11;  year=0;
      while(year <= PERIOD)
      {printf("%d     %f\n",  year, amount);
      value=amount  +   inrate*amount;        year=year+1;
      amount=value;
      }
}
```

```
#include <stdio.h>
intmul(intx, inty);/*Function declaration*/
void main( )
{
     inta, b, c;
     a=4;         b=7;
     c= mul(a,b);
     printf("Multiplication of  %d  and %d  is: %d",  a,b,c);
}
intmul(intx, inty)
{
     intp;
     p=x*y;
     return p;
}
```

Features of C

A. Robust-It has a rich set of built-in functions & operators helping in writing complex programs

B. Suitable for writing both System & Application SW

C. Efficient & fast

D. Portable-Any C program written on one computer can be run on another computer with minimal or no modifications.

E. Well suited for structured programming, requiring the user to think of a problem in terms of function modules or blocks. A proper collection of this modules would make a complete program.

F. Extendable-new functions can be added to C libraries

A. Library functions
B. Different data types
C. Operators
D. Declaration of variables, constants
E. Storage classes (auto, register, static, extern)
F. IF-THEN-ELSE, FOR loop, DO WHILE, SWITCH-CASE
G. Functions to perform specific tasks
H. STRUCTURE and UNION
I. Memory allocation (malloc and calloc)
J. Pointers

Observation

All computer programs have two important elements: Code and Data.

Observation

Prof.R.Gururaj   Object Oriented Programming

```
void main( )
{
      intlength, breadth, age, rank, area;
      length=10; breadth=5;  age=15; rank=8;
      area= compute_area(length, breadth);
      printf(“Area of the shape is: %d”,  area);
}
intcompute_area(intl, intb)
{
      inta;   p=l*b;
      return a;
}
```

This is way of programming is known as Process-Oriented approach.

This can be characterized as code acting data.

If by mistake if we pass on some invalid data values to the function (which is semantically wrong), still the code acts on the data without discrimination.

```
void main( )
    int length, breadth, age, rank, area;
    length=10; breadth=5;  age=15; rank=8;
    area= compute_area(age, rank);
    printf("Area of the shape is: %d",  area);
}
int compute_area(int l, int b)
{
    int a;  p=l*b;
    return a;
}
```

As long as the size (number of lines) of the code is within some limits it is OK.

But when it increases in size i.e., more that 5000Lines, it becomes difficult to keep track of what data is passed on to what code(function).

Because the complexity increases drastically. Programmers can make mistakes.

Debugging and Maintenance teams face a lot of hardship.

All declarations happen in the beginning of the code.

For what data we are invoking the function?

Process Oriented approach

1. Data and behavior (code/function/procedure) are separated.
2. Mostly data is global.
3. Any code can access any data. Uncontrolled access to data.
4. Due to this, testing and debugging is difficult.
5. Leads to low data integrity.
6. In the Process-Oriented approach, the focus is on *what is happening*? But not *who is being affected*.
7. For what entity/object, am I executing the code, not clear.
8. Humans understand the world in terms of entities or objects.
9. This problem can be solved by use of Structures to some extent but not completely.

Object-oriented approach

Object-Oriented approach

- Humans understand the world in terms of entities or objects. We experience objects in our daily life.

  Ex: Car, Table, Faculty, Student
- Objects contain data and behavior.
- It is about wrapping code and data in objects.
- If properly designed, we will not have any global data.
- Hence we have strict control over access to data and code.
- Hence less possibility of misuse.
- Easy to test and debug.
- High integrity.

Objects are much more than structures in C and C++.

In objects methods are used to perform operations on data as well as perform other actions.

We can hide certain elements from outside world.

In object orientation,

Data is referred to as → attributes.

Behavior is referred to as → methods

Restriction on access to data gives rise to ***data hiding***.

One object should not be allowed to access and change the values of attributes of other object directly.

Rather it should be through designated *interfaces* provided by the objects to outsiders.

Hence Data and operations are encapsulated in objects.

What is an object

Objects are the building blocks of object-oriented programming.

A program actually uses a collection of objects.

Each object has data and behavior.

Object Data- Data represent the state of the object at given point of time.

In OO data is called attributes.

Every attribute contains some information that differentiate between various objects.

Object behavior

Behavior of an object is what an object can do.

In OO behavior is contained in *methods*.

Methods of an object can be invoked by sending a message to that object.

Usually method can be a *setter* or a *getter* method.

Setters change/update the values of attributes.

Getters get the values associated with attributes.

Encapsulation is a powerful feature of OO.

Note:

All objects of same class share the implementation copy.

What is a class?

Class is a blueprint of an object.

It defines the structure of similar kind of objects.

Every object must belong to a class and must belong the only one class.

Class can be seen as templates for objects.

Classes can be seen as high-level data types.

Attributes reflect the state of the object.

Methods implement the behavior of the object.

One object calls methods of other objects to invoke the behavior.

Messaging defines the communication mechanism between objects.

An object can call any public method of another object.

Encapsulation & Data hiding

Primary advantage of object is that it need not reveal all its attributes and behavior to outsiders.

In good OO design, an object must reveal only the interface that other objects must have to interact with it.

The details that are not necessary must be hidden.

Encapsulation: Putting data and behavior together.

Data hiding: is major advantage of encapsulation.

Interfaces

The *interface* define the fundamental means of communication between objects.

Message passing happens trough one of the interfaces.

Interface should describe how others can interact.

The methods that are part of the interface are public methods.

Private data

Helps to hide data.

Hence attributes are not part of the interface.

Should not declare attributes public.

Usually interfaces need not change while we change the internals of a class.

Users need not change their application code.

Inheritance

Inheritance is one of the most powerful features of OO.

This results in code reuse.

We can define relationships between classes.

It allows a class inherit the attributes and behaviour of other classes.

Superclass and Subclass

Is-a relationship

```
class Shape   // super class
{  String color;
    Shape(String c) {color=c;}
     void compteArea(){}
}
Class Rect extends Shape // subclass
{
     Int l,b;
     void computePerimeter(){}
}
```

Abstraction

Control Abstraction

Abstraction of behavior. Provides an easier, higher level API to hide **client** from unnecessary execution **details**.

Implementing abstraction

To implement abstraction in object oriented programming, you should be able to define just the behavior. With very limited or no implementation logic.

Hierarchical abstraction

We manage the complexity of systems through hierarchical abstraction. Ex. Car- music, transmission, fuel, brake etc.

Data abstraction

Abstraction of data structures. Data abstraction refers to defining the behavior of the data structure. Data may be internally represented in different ways in concrete implementations.

Polymorphism

- ❑ Method overloading
- ❑ Method overriding.

Composition

Defines has-a   or is-part-of relationship.

Object may contain other objects .

We see aggregation and composition.

Summary

- Importance of high-level languages
- Procedural vs. OO
- Issues with Procedural approach
- What is an object
- Class
- Attributes and behaviour
- Encapsulation and data hiding
- Inheritance
- Polymorphism
- Aaggregation and Composition