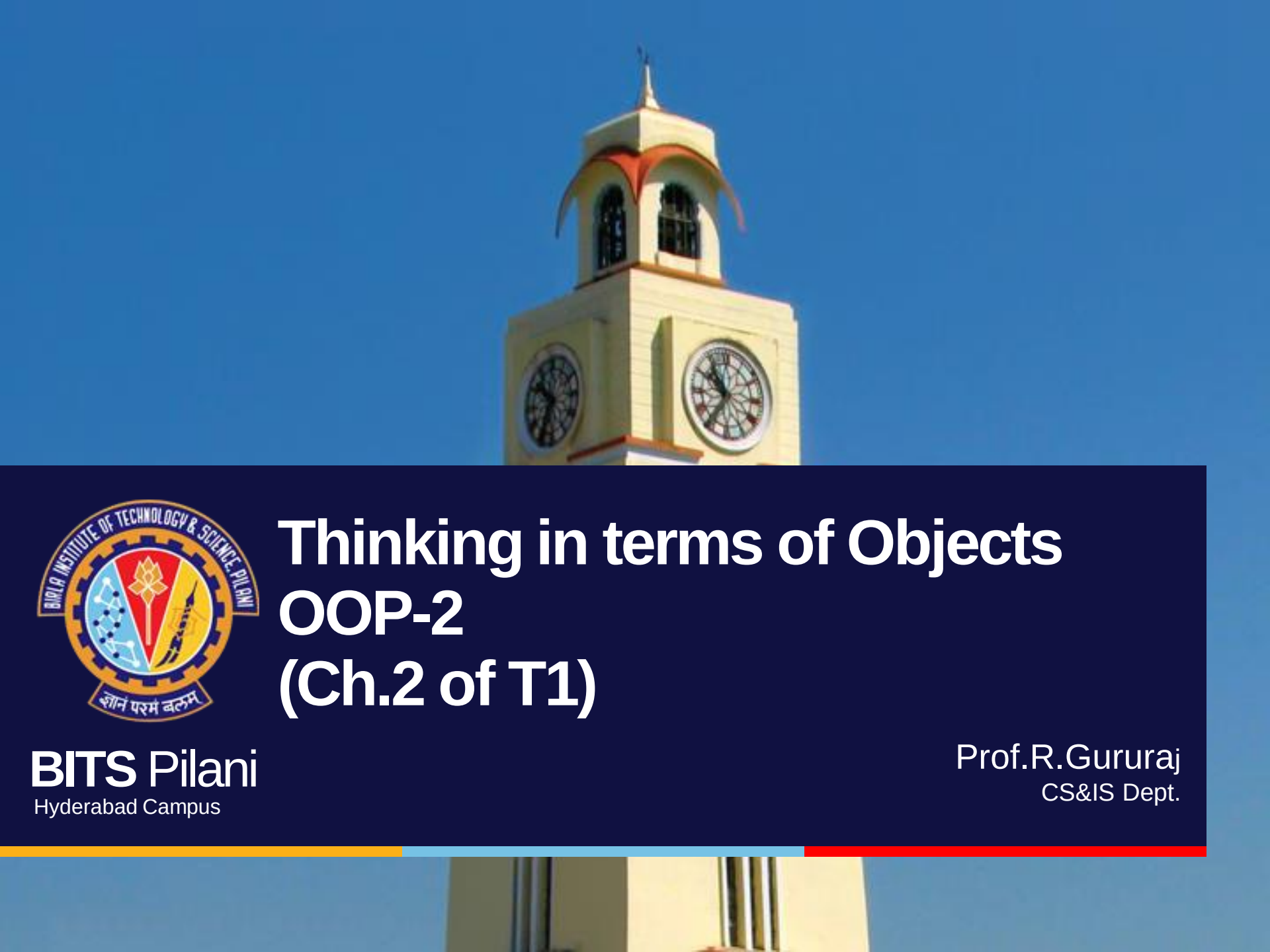
CS F213
Object Oriented Programming

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.



Thinking in terms of Objects
OOP-2
(Ch.2 of T1)

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Objects

The basic unit of OO design is class.

To have a good sense of OO thought process, we need to -

1. Understand the difference between Interface and Implementation
2. Think more abstractly
3. Give user a minimal interface as you can

Interface

We hide non-essential data from the user.

The service provided to the user comprise the interface.

Hence identifying users and their needs plays a major role in deciding the interface part.

Implementation

Implementation details are hidden from the user.

Change in implementation should not require change in the user application code.

Interface includes the syntax to call a public method and return value.

Ex: Cellphone, Car brake system

If a class has no public method, it is useless. First make all of them private.

Then based on the user needs negotiate and add interfaces.

Example

```
class Emp
{
    private int id, basic, sal, hra, da, rent, ta, bonus;  String city;
    Emp(int i, int bas, int h, int d, String ct)
    { id=i; basic=bas; hra=h; da=d; city=ct;}
     private int computeSal(int id)
     {// code to compute Salary}
     private int computeRentAllowance(int id)
     {// code to compute rent}
     private  int computeTA(int id)
     {// code to compute TA}
 }
```

Example

```
class Emp
{
    private int id, basic, sal, hra, da, rent, ta, bonus;  String city;
    Emp(int i, int bas, int h, int d, String ct)
    { id=i; basic=bas; hra=h; da=d; city=ct;}
     private int computeSal(int id){// code to compute Salary}
      private int computeRentAllowance(int id){// code to compute rent}
      private  int computeTA(int id){// code to compute TA}
      public  int getSalary(int id){ return computeSal(id);}   // first interface added
 }
```

Example

```
class Emp
{
    private int id, basic, sal, hra, da, rent, ta, bonus;  String city;
    Emp(int i, int bas, int h, int d, String ct)
    { id=i; basic=bas; hra=h; da=d; city=ct;}
     private int computeSal(int id){// code to compute Salary}
      private int computeRentAllowance(int id){// code to compute rent}
      private  int computeTA(int id){// code to compute TA}
      public  int getSalary(int id){ return computeSal(id);}   // first interface added
      public int computeBonus(int id)   // second interface added
     { sal=computeSal(id);       rent= computeRentAllowance(id);
       ta=computeTA(id);
       return  sal+rent+ta;
      }
 }
```

Abstract thinking

innovate achieve lead

One advantage of OO is that classes can be reused. Reusable classes tend to have more abstract behavior than concrete.

Concrete- more specific

Abstract- more generic

Ex: DataConnectionManager

To make a class more reusable, make ir more abstract.

Abstract classes

Superclass declares the structure of the class, without complete implementation to all the methods(abstract class).

Here superclass is unable to define the implementation.

It is the responsibility of the sub class to provide implementation required.

```
Class DataConnectionManager
{
    abstract void getConnection();
    abstract void closeConnection();
}
Class OracleDBConnectionManager extends DataConnectionManager
{
        void getConnection(){// code to connect to Oracle DB}
        void closeConnection() {// code to diconnectconnect OracleDB}
}
Class MongoDBConnectionManager extends DataConnectionManager
{
        void getConnection()  {// code to connect to Mongo DB}
        void closeConnection() {// code to diconnectconnect MongoDB}
}
```

Give user minimal interface

Always provide the user with little knowledge about the internals of the class.

1. Give user only what is obsoletely needed
2. For class, make as few interfaces as possible.

Ex: Emp salary

Always design a class from the user's perspective, not from technology perspective.

Keep end-user in mind, not the developer.

Identify the object's behavior from the user's perspective.

Consider the environmental constraints as well.

Identify the public interfaces- check if the interface contributes to the behavior of the object, if not it need not be the part of the interface.

Each interface must handle only one behavior.

Example

```
class Emp
{
    private int id, basic, sal, hra, da, rent, ta, bonus;  String city;
    Emp(int i, int bas, int h, int d, String ct)
    { id=i; basic=bas; hra=h; da=d; city=ct;}
     private int computeSal(int id){// code to compute Salary}
      private int computeRentAllowance(int id){// code to compute rent}
      private  int computeTA(int id){// code to compute TA}
      public  int getSalary(int id){ return computeSal(id);}
      public int computeBonus(int id)
     { sal=computeSal(id);       rent= computeRentAllowance(id);
       ta=computeTA(id);
       return  sal+rent+ta;
      }
 }
```

Hence,

Interface- what a user can see

Ex: int getSalary(int id) // first interface

int computeBonus(int id) // second interface

Implementation- nuts and bolts of an object which can't be seen by the user.

Excepting above, remaining part of the object is treated as the implementation.

Summary

- Interface
- Implementation
- Identifying interface
- Applying abstract thinking