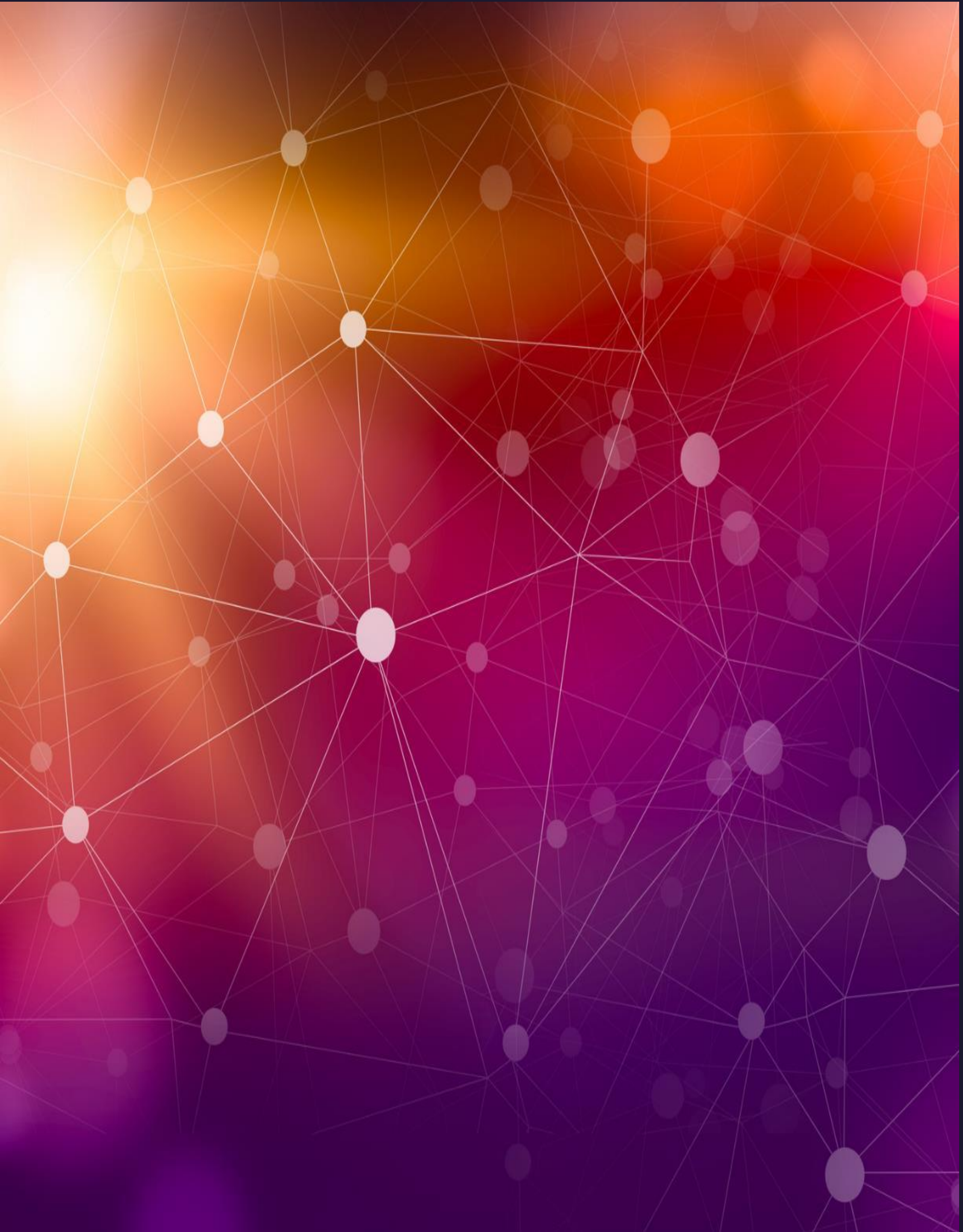


# Object-Oriented Programming

## Information Hiding

## Lecture 3

# Object-Oriented Programming
# Lecture 3
# Information Hiding

©2011, 2014, 2022, 2023,2024. David W. White, Tyrone A. Edwards, & Christopher Panther & Rorron A. Clarke

University of Technology, Jamaica
Faculty of Engineering and Computing
School of Computing and Information Technology

Email: dwwhite@utech.edu.jm, taedwards@utech.edu.jm; Christopher.Panther@utech.edu.jm; rclarke@utech.edu.jm

# Object-Oriented Programming

## Objectives/Expected Outcome

**At the end of this lecture students should be able to:**

- describe the role of access specifiers in hiding or exposing class attributes and operations.
- differentiate between constant and static values.
- discuss how the three types of class constructors (Default, Primary & Copy) are used to create class objects.
- differentiate memory allocation using the program stack versus the program heap.
- evaluate an object-oriented program in which each class in place into a separate file.

# Object-Oriented Programming

**Topics to be covered in this lecture:**

- Encapsulation and the role of access specifiers
- Constant and static values
- Default Primary and Copy constructors
- Dynamic memory allocation
- Separating classes and driver files
- Complete program example

## Encapsulation: Role of Access Specifiers

- Encapsulation is a characteristic of object-oriented programs where the attributes and operations (i.e., class members) are wrapped inside the class.
- Encapsulation allows the object of a class to operate like a black box.
- Only the class members that need to be accessed by the outside world are exposed .
- Access specifiers are used to define the level of access/visibility for each class member.

## Encapsulation: Role of Access Specifiers Cont'd

In the object-oriented paradigm –common access specifiers are:

- ***private***
- ***public***
- ***protected***
- All of these are supported in C++ and Java using lowercase for the keywords
- Others exist such as package in Java

## Encapsulation: Role of Access Specifiers Cont'd

- **private**: member can only be accessed by objects of the class and friends o the class.
- **public**: member can be accessed by anyone using objects of the class.
- **protected**: member can only be accessed by objects of the class and its descendants (children, grandchildren, etc.)

## Encapsulation: Role of Access Specifiers Cont'd

- **private**: member can only be accessed by objects of the class and friends o the class.
- **public**: member can be accessed by anyone using objects of the class.
- **protected**: member can only be accessed by objects of the class and its descendants (children, grandchildren, etc.)

## Constant Values vs Static Values

- A constant is a value that cannot be modified.
- Attributes and Objects can be declared constant (called immutable values).
- Once declared constant the value is fixed for the life of the object.
- Implemented using the ***const*** keyword in C++ and the ***final*** keyword in Java.

## Constant Values vs Static Values

- A static attribute is one in which all objects containing that attribute share the same value.
- If the value of the attribute is changed in one object, the value of the attribute is immediately changed in all other objects of the same class.
- This implies that the attribute share the same space in memory.
- Static values remain even when the object goes out of scope .
- Keyword ***static*** is used in C++ and Java

# Default, Primary and Copy Constructor

- Constructors are methods which allow the attributes of an object to be properly initialized when the object is instantiated (created).
- Constructors have the same name as the class and are called automatically when each new object of the class is created.
- Three types of class constructors:
  - Default
  - Primary and
  - Copy

# Default, Primary and Copy Constructor

- ***Default*** Constructor

Takes no parameters and sets attributes to some predefine default values base on the data type of the attribute

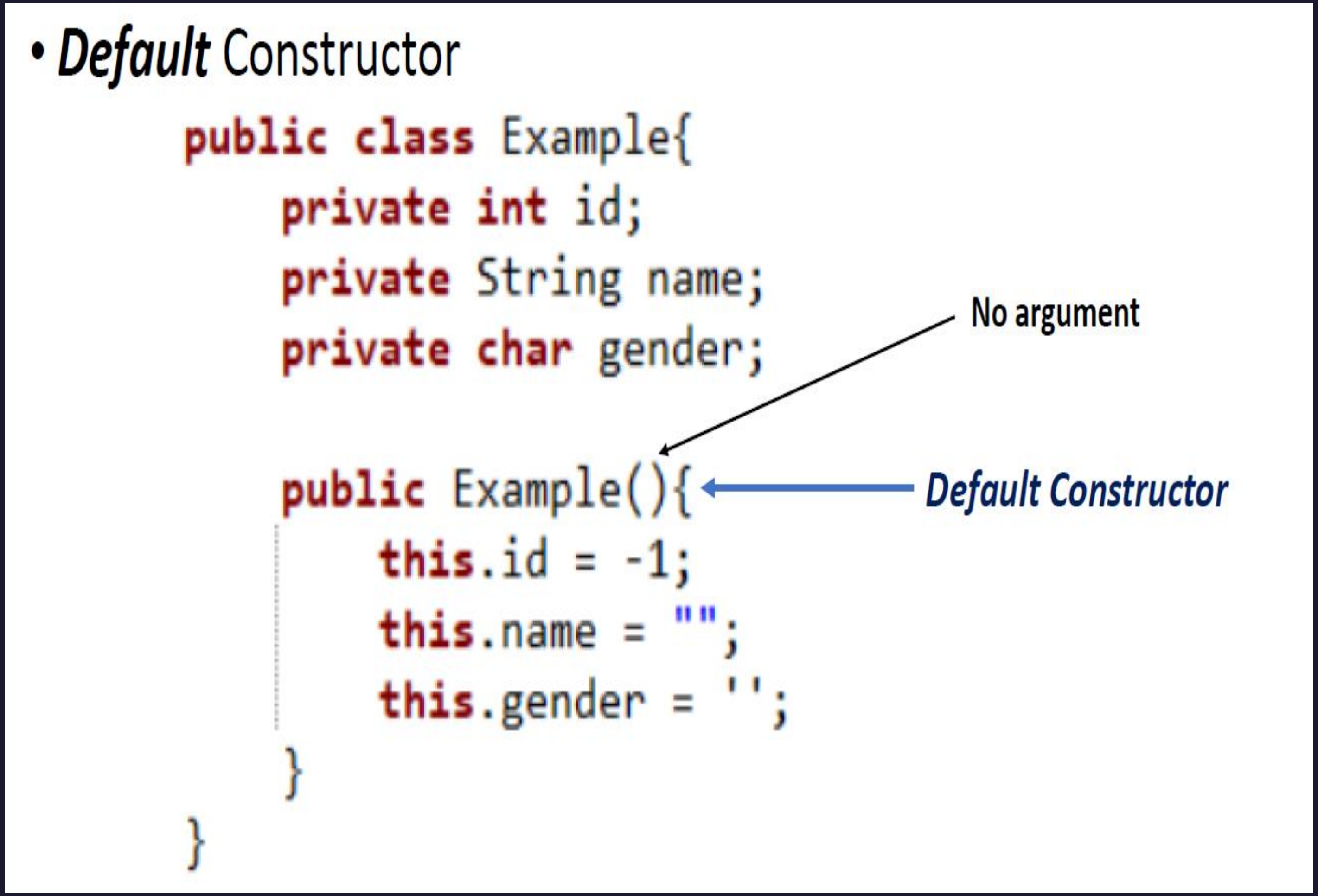
- ***Primary*** Constructor

Sets the attributes to values passed in as parameters. It takes one argument for each attribute

- ***Copy*** Constructor

Sets the attributes to the value of their counterparts in an entire object passed in as parameter

# Default, Primary and Copy Constructor

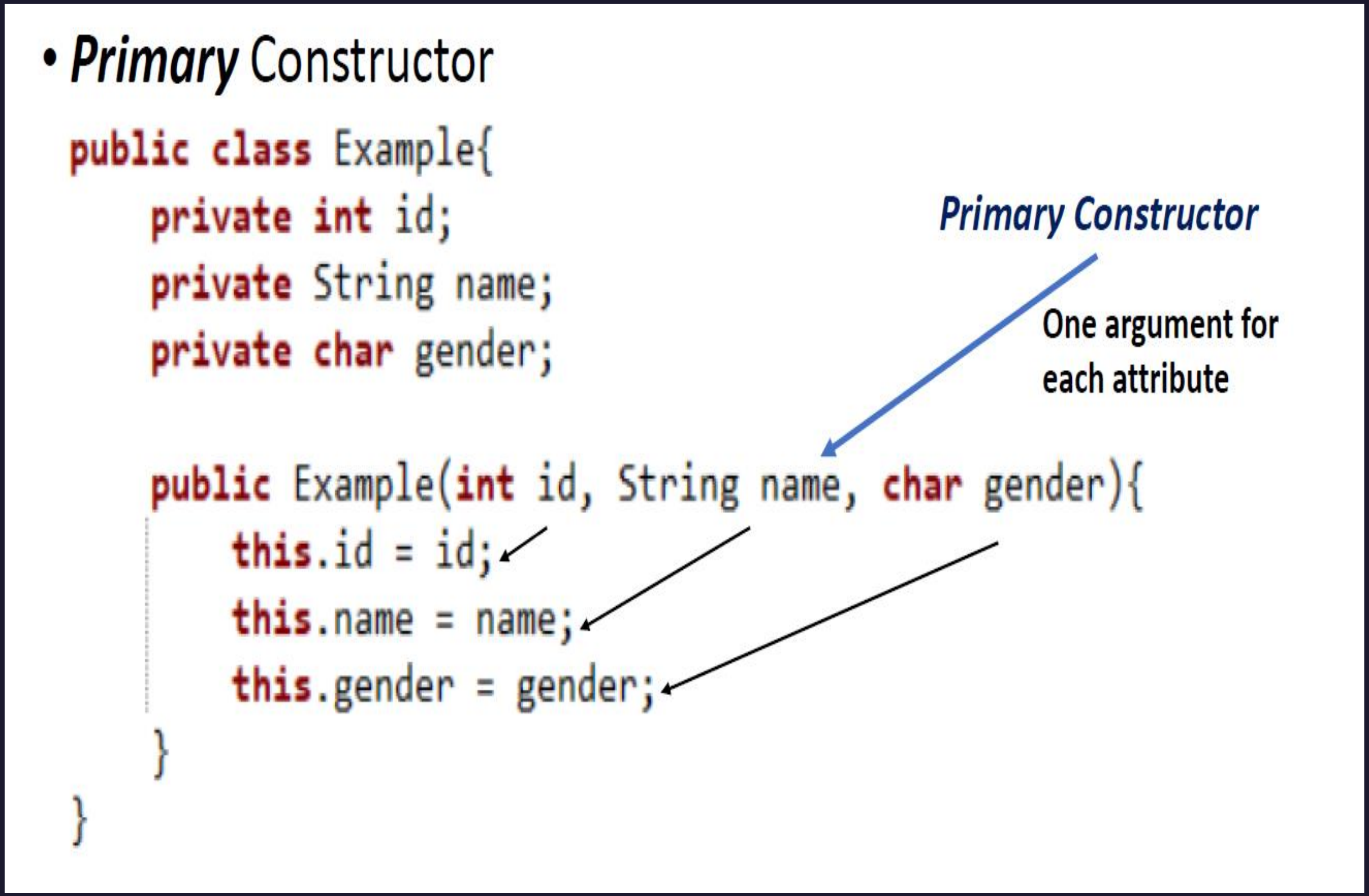


- ***Default*** Constructor

```
public class Example{
    private int id;
    private String name;
    private char gender;

    public Example(){
        this.id = -1;
        this.name = "";
        this.gender = '';
    }
}
```

# Default, Primary and Copy Constructor

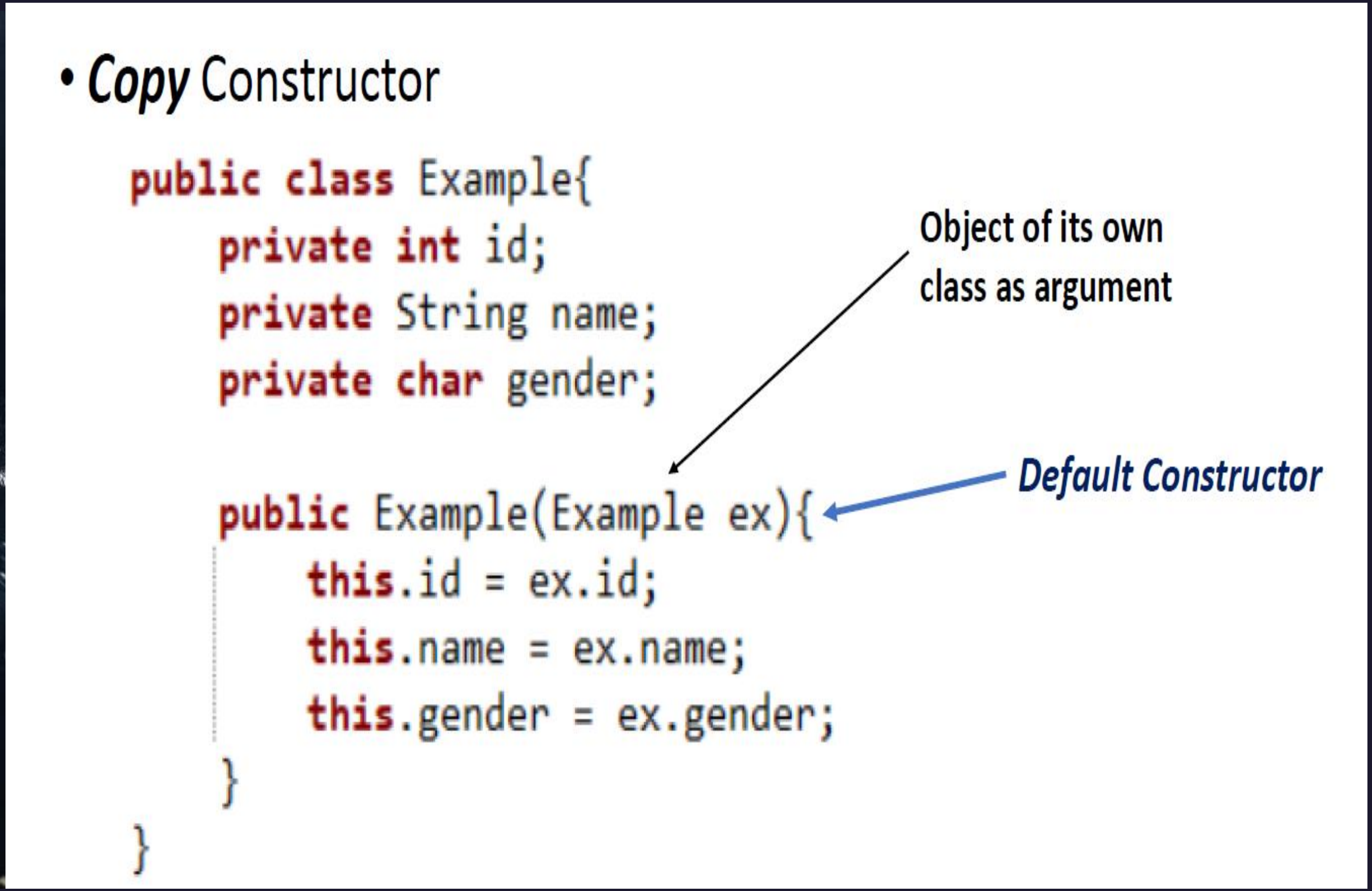


- ***Primary*** Constructor

```
public class Example{
    private int id;
    private String name;
    private char gender;

    public Example(int id, String name, char gender){
        this.id = id;
        this.name = name;
        this.gender = gender;
    }
}
```

# Default, Primary and Copy Constructor



- *Copy* Constructor

```java
public class Example{
    private int id;
    private String name;
    private char gender;

    public Example(Example ex){
        this.id = ex.id;
        this.name = ex.name;
        this.gender = ex.gender;
    }
}
```

## Dynamic Memory Allocation

- Normally memory is allocated from the **program stack** e.g., **int** X = 0;
- The above means set aside enough room in memory to store an integer which will be referred to as X and assign it an initial value of 0.

## Dynamic Memory Allocation

- Memory can instead be allocated as needed (i.e., dynamically) from the **program heap** e.g.
- C++: Student ***s** = new Student;
- Java: Student **s**= new Student();
- Here ‘**s**’ in C++ is a pointer to the space in memory for the Student object created. In Java ‘**s**’ is a reference to the object in the computer’s memory.

## Dynamic Memory Allocation

- Dynamically allocated memory can be deallocated, that is it can be reclaimed back from the program and given back to the heap.
- Also called freeing memory
- In C++ must be done explicitly to prevent memory leaks by using the ***delete*** keyword:
- ***Delete s;***
- In Java this is done automatically by the Garbage Collector which detects that an object is no longer in use and frees its memory.

## Separating Class and Driver Files

- Instead of writing an OOP program as one monolithic file it is possible and desirable to place each class in a separate file .
- The file has the same name as the class.
- Each class can be accessed when needed .
- Header files in C++ packages and archives in Java.
- #include in C and import in Java

# Separating Class and Driver Files

- Program can be run from Driver file containing the main() method.
- Sometimes it is desirable to separate the class interface from its implementation.
- The interface is exposed to the user.
- The implementation can be kept hidden.
- It is possible to change the implementation and leave the interface intact

## Example Program

- The following is a complete example program.
- Starts with Analysis, then Design, finally implementation.
  - –Classes in separate files
  - –Use of driver file
  - –Implementing composition (has-a) relationship in code .
  - –Default, Primary and Copy constructors
  - –Using a static attribute
  - –Creating objects and calling methods from driver file

# Object-Oriented Analysis

- **Sample Requirements**

The UTech experimental smart power meter is designed to track electricity consumption at various points on the campus. It replaces the old analog power meter. Each experimental smart power meter has a serial number and tracks electricity consumption in kilo-watt-hours (KWH). It can increment the electricity consumption by one each time and can also display its serial number and the electricity consumption.

# Object-Oriented Analysis

- **Sample Requirements**

Each Experimental Smart Power Meter also has a single Transponder unit. For now, each Transponder unit can send and receive, and tracks the total number of Transponder units including itself that are on the power grid. Each time an Experimental Smart Power Meter comes on the power grid the number of transponder units is increased by one. Each Transponder unit can display the total number of Transponder units on the power grid at any given moment.

# Object-Oriented Analysis

- **Sample Requirements**

1. Create a model for the system described showing clearly the class attributes and methods. Also, show the relationship among the classes identified. For each class include a Default, Primary and Copy constructor, also mutators and accessors among the methods.

2. Implement your model in an OOP language and write a small driver program to create objects of the classes and call the methods.

# Thank You

End of Presentation.