# Object Oriented Programming
# Lecture

# **Information Hiding**



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

Email: dwwhite@utech.edu.jm, taedwards@utech.edu.jm

# Object Oriented Programming

**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.
- Discuss how the three types of class constructors (default, primary and copy) are used to create class objects.
- Differentiate memory allocation using the program stack versus the program heap.
- Differentiate between constant and static values.
- Write an object oriented program in which each class in placed 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
- A complete 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

- 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

- ***Private***

can only be accessed by objects of the class and friends of the class

- ***Public***

can be accessed by anyone using objects of the class

- ***Protected***

- can only be accessed by objects of the class and its descendants (children, grandchildren, etc)

# Constant and 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 and Static Values

- A static attribute is one in which all objects containing the 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 they go out of scope
- Keyword ***static*** is used in C++ and Java

# Default, primary and copy constructors

- 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 constructors

- ***Default***

Takes no parameters, sets attributes to some predefined default values based on the data type of the attribute

- ***Primary***

Sets the attributes to values passed in as parameters

- ***Copy***

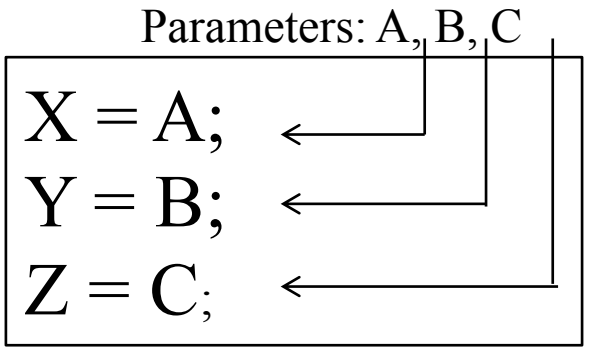
Sets the attributes to the value of their counterparts in an entire object passed in as a parameter

# Default, primary and copy constructors

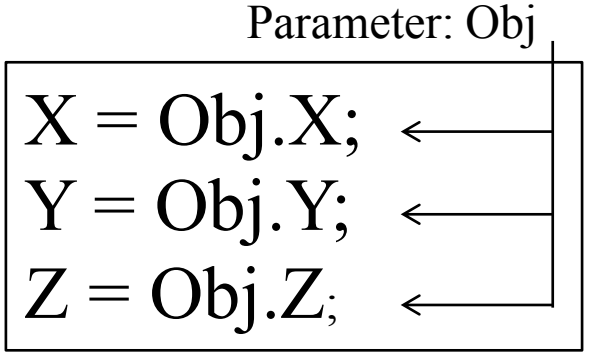
**Default Constructor**

```
X = 0;
Y = “”;
Z = 0.0;
```

**Primary Constructor**




**Copy Constructor**



# 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 heap, e.g.

- C++:

```
Student *s = new Student;
```

- Java:

```
Student s = new Student();
```

- Here, in C++ **s** is a pointer to the space in memory for the Student object created, which in Java **s** is a reference to the object

# 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 classes 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 classes and driver files

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

# A complete example

- The following is a complete example program
- Starts with Analysis, then Design, finally Implementation in both C++ and Java
- Demonstrates:
  - 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 call 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 a 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, and mutators and accessors among the methods.
2. Implement your model in an OOP programming language and write a small driver program to create objects of the classes and call the methods.

# Object-Oriented Analysis

**Underlining the nouns in noun-verb analysis**

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

**Underlining the nouns in noun-verb analysis**

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 a 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

**Nouns**

- UTech (rejected)
- experimental smart power meter (accepted)
- point (rejected)
- campus (rejected)
- old analog power meter (rejected)
- serial number (accepted)
- electricity consumption (accepted)

# Object-Oriented Analysis

**Nouns**

- kilo-watt-hours (KWH) (rejected)
- One (rejected)
- power grid (rejected)
- transponder unit (accepted)
- One (rejected)
- total number of transponder units (accepted)
- Moment (rejected)

# Object-Oriented Analysis

**Resulting Classes and Attributes**

- experimental smart power meter **(class)**
- serial number (*attribute*)
- electricity consumption (*attribute*)
- transponder (attribute)
- transponder unit **(class)**
- total number of transponder units (*attribute*)

# Object-Oriented Analysis

**Outlining the verbs in noun-verb analysis**

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

**Outlining the verbs in noun-verb analysis**

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 a 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

**Verbs**

- track (rejected)
- replace (rejected)
- Increment (accepted)
- display (accepted)
- send (accepted)
- receive (accepted)

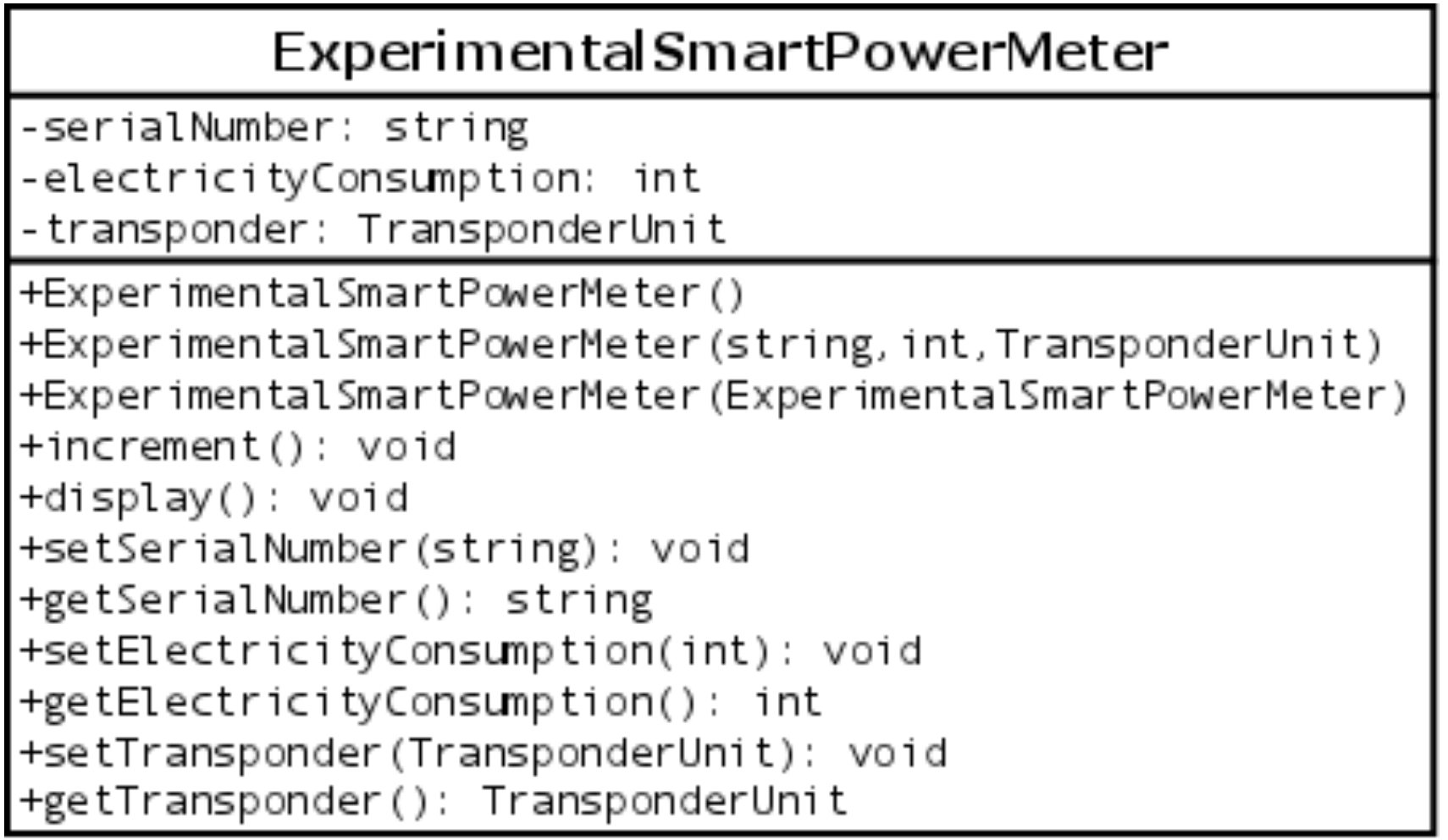
# Object-Oriented Analysis

**Resulting Methods**

- increment (accepted)
- display (accepted)
- send (accepted)
- receive (accepted)

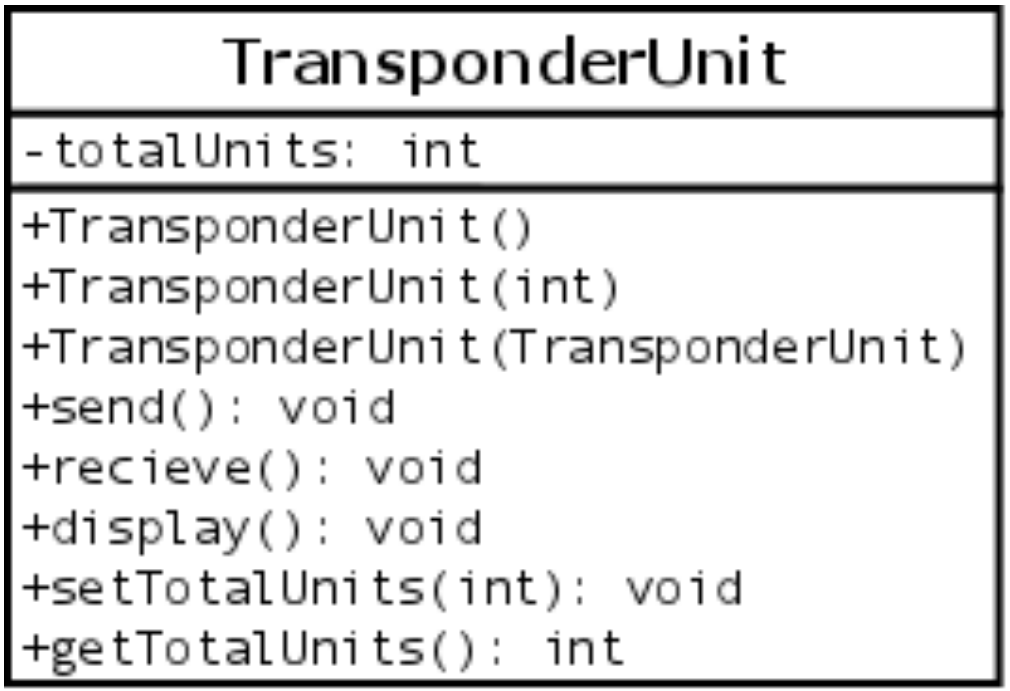
# Modeling classes derived from OOA

**Detailed Individual UML diagram**



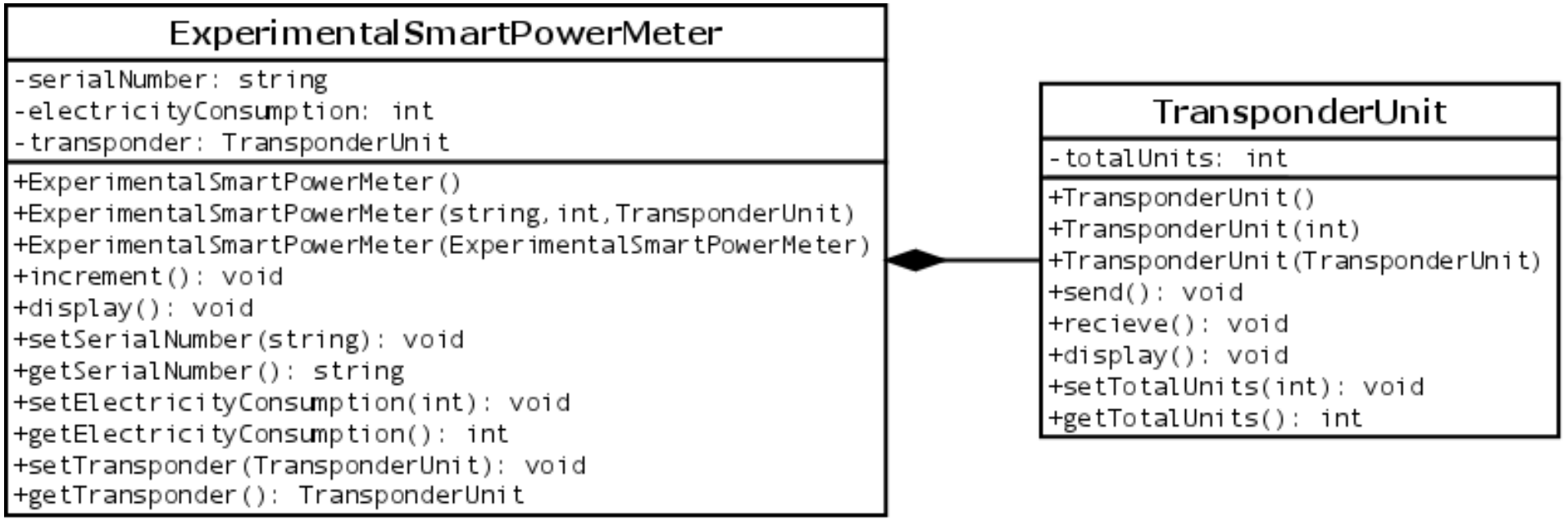
# Modeling classes derived from OOA

**Detailed Individual UML diagram**



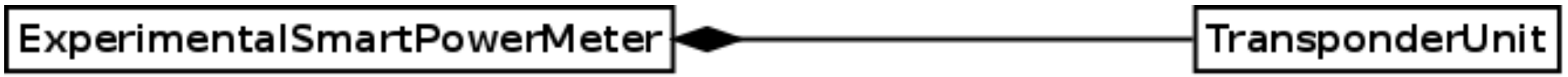
# Modeling classes derived from OOA

**Detailed UML Relationship diagram**



# Modeling classes derived from OOA

**Relationship UML diagram**



# C++ Implementation

TransponderUnit.h

```
#include <iostream>
using namespace std;

//TransponderUnit class
class TransponderUnit {
	//declare static attribute
	private:
		static int totalUnits;

	public:
		//default constructor
		TransponderUnit()
		{
			++totalUnits;
		}
```

# C++ Implementation

TransponderUnit.h

```
//primary constructor
TransponderUnit(int nu)
{
	totalUnits = nu;
}

//copy constructor
TransponderUnit(const TransponderUnit &tu)
{
	totalUnits = tu.totalUnits;
}

//send method
void send()
{
}
```

# C++ Implementation

TransponderUnit.h

```
//receive method
void receive()
{
}

//display's total number of units on grid
void display() const
{
	cout  << "The number of units on the grid is "
	      << totalUnits << endl;
}
```

# C++ Implementation

TransponderUnit.h

```
                //set the total number of units on the grid
                void setTotalUnits(int nu)
                {
                        totalUnits = nu;
                }

                //get the total number of units on the grid
                int getTotalUnits() const
                {
                        return totalUnits;
                }
};

//initialize static attribute
 int TransponderUnit::totalUnits = 0;
```

# Implementation - C++

ExperimentalSmartPowerMeter.h

```
//include the TransponderUnit class
#include "TransponderUnit.h"
#include <string>
using namespace std;

//declare the ExperimentalSmartPowerMeter class
class ExperimentalSmartPowerMeter {

	//declare attributes in class
	private:
		string serialNumber;
		int electricityConsumption;
		TransponderUnit tu;
```

# Implementation - C++

ExperimentalSmartPowerMeter.h

```
//declare the methods in class
public:
	//default constructor
	ExperimentalSmartPowerMeter()
	{
		serialNumber = "00000";
		electricityConsumption = 0;
		tu.setTotalUnits(1);
	}

	//primary constructor
	ExperimentalSmartPowerMeter(string sn, int ec, int ntu)
	{
		serialNumber = sn;
		electricityConsumption = ec;
		tu.setTotalUnits(ntu);
	}
```

# Implementation - C++

ExperimentalSmartPowerMeter.h

```
//copy constructor
ExperimentalSmartPowerMeter(
                const ExperimentalSmartPowerMeter  &obj)
{
        serialNumber = obj.serialNumber;
        electricityConsumption =
obj.electricityConsumption;
        tu.setTotalUnits(
                                obj.tu.getTotalUnits()
+ 1);
}

//method to increment electricity consumption
void increment()
{
        ++electricityConsumption;
}
```

# Implementation - C++

ExperimentalSmartPowerMeter.h

```
//method to display attributes
void display()
{
	cout << "This meter's serial number is " <<
		serialNumber << endl;
	cout << "Electricity consumption is " <<
		electricityConsumption << endl;
	cout << "Total transponders on-line is " <<
		tu.getTotalUnits() << endl;
}

//set the serial number
void setSerialNumber(string sn)
{
	serialNumber = sn;
}
```

# Implementation - C++

ExperimentalSmartPowerMeter.h

```
        //get the serial number
        string getSerialNumber()const
        {
                return serialNumber;
        }

        //set the electricity consumption
        void setElectricityConsumption(int ec)
        {
                electricityConsumption = ec;
        }

        //return the electricity consumption
        int getElectricityConsumption()const
        {
                return electricityConsumption;
        }
};
```

# Implementation - C++

Driver.cpp

```
//This code is to demonstrate how one of the three different types
//of constructors can be called when an object of the
//ExperimentalSmartPowerMeter class is created

#include "ExperimentalSmartPowerMeter.h"
#include <iostream>
using namespace std;

int main()
{
	//create an object A calling the default constructor
	ExperimentalSmartPowerMeter A;
	//Display the attributes values in object A
	A.display();
```

# Implementation - C++

Driver.cpp

```
        //create an object B calling the primary constructor

        ExperimentalSmartPowerMeter B("12345",500, 2);
        //Display the attributes values in object B
        B.display();

        //create an object C calling the copy constructor with B
        //as a parameter
        ExperimentalSmartPowerMeter C(B);
        //Display the attributes values in object B
        C.display();

        return 0;
}
```

# Implementation - Java

TransponderUnit.java

```
package powermeterproject;

//TransponderUnit class
public class TransponderUnit {
        //declare and initialize static attribute
        private static int totalUnits = 0;

        //default constructor
        public TransponderUnit()
        {
                ++totalUnits;
        }
```

# Implementation - Java

TransponderUnit.java

```
//primary constructor
public TransponderUnit(int nu)
{
        totalUnits = nu;
}

//copy constructor
public TransponderUnit(TransponderUnit tu)
{
        totalUnits = tu.totalUnits;
}

//send method
public void send()
{
}
```

# Implementation - Java

TransponderUnit.java

```
//receive method
public void receive()
{
}

//display's total number of units on grid
public void display()
{
        System.out.println("The number of units on the grid is "
                +totalUnits);
}
```

# Implementation - Java

TransponderUnit.java

```
        //set the total number of TotalNumberUnits
        public void setTotalUnits(int nu)
        {
                totalUnits = nu;
        }

        //get the total number of TotalNumberUnits
        public int getTotalUnits()
        {
                return totalUnits;
        }
}
```

# Implementation - Java

ExperimentalSmartPowerMeter.java

```
package powermeterproject;

//import the TransponderUnit class
import powermeterproject.TransponderUnit;

//declare the ExperimentalSmartPowerMeter class
public class ExperimentalSmartPowerMeter {

        //declare attributes in class
        private String serialNumber;
        private int electricityConsumption;
        private TransponderUnit tu = new TransponderUnit();

        //declare the methods in class
```

# Implementation - Java

ExperimentalSmartPowerMeter.java

```
//default constructor
public ExperimentalSmartPowerMeter()
{
        serialNumber = "00000";
        electricityConsumption = 0;
        tu.setTotalUnits(1);
}

//primary constructor
public ExperimentalSmartPowerMeter(String sn, int ec, int ntu)
{
        serialNumber = sn;
        electricityConsumption = ec;
        tu.setTotalUnits(ntu);
}
```

# Implementation - Java

ExperimentalSmartPowerMeter.java

```
//copy constructor
public ExperimentalSmartPowerMeter(
        ExperimentalSmartPowerMeter obj)
{
        serialNumber = obj.serialNumber;
        electricityConsumption = obj.electricityConsumption;
        tu.setTotalUnits(obj.tu.getTotalUnits() + 1);
}

//method to increment electricity consumption
public void increment()
{
        ++electricityConsumption;
}
```

# Implementation - Java

ExperimentalSmartPowerMeter.java

```
        //method to display attributes
        public void display()
        {
                System.out.println("This meter's serial number is " +
                        serialNumber );
                System.out.println("Electricity consumption is " +
                        electricityConsumption);
                System.out.println("Total transponders on-line is " +
                        tu.getTotalUnits());
        }

        //set the serial number
        public void setSerialNumber(String sn)
        {
                serialNumber = sn;
        }
```

# Implementation - Java

ExperimentalSmartPowerMeter.java

```
        //get the serial number
        public String getSerialNumber()
        {
                return serialNumber;
        }

        //set the electricity consumption
        public void setElectricityConsumption(int ec)
        {
                electricityConsumption = ec;
        }

        //return the electricity consumption
        public int getElectricityConsumption()
        {
                return electricityConsumption;
        }
}
```

# Implementation - Java

MeterDriver.java

```
//This class is to demonstrate how one of the three different types
//of constructors can be called when an object of the
//ExperimentalSmartPowerMeter class is created

import powermeterproject.ExperimentalSmartPowerMeter;

public class MeterDriver {

        public static void main(String[] args) {
                //create an object A calling the default constructor
                ExperimentalSmartPowerMeter A = new
                        ExperimentalSmartPowerMeter();
                //Display the attributes values in object A
                A.display();
```

# Implementation - Java

MeterDriver.java

```
        //create an object B calling the primary constructor
        ExperimentalSmartPowerMeter B = new
                ExperimentalSmartPowerMeter("12345",500, 2);
        //Display the attributes values in object B
        B.display();

        //create an object C calling the copy constructor with B
        // as a parameter
        ExperimentalSmartPowerMeter C = new
                ExperimentalSmartPowerMeter(B);
        //Display the attributes values in object B
        C.display();
    }

}
```

# Output from Sample Run (C++ and Java)

```
This meter's serial number is 00000
Electricity consumption is 0
Total transponders on-line is 1

This meter's serial number is 12345
Electricity consumption is 500
Total transponders on-line is 2

This meter's serial number is 12345
Electricity consumption is 500
Total transponders on-line is 3
```