University of Technology, Jamaica
School of Computing and Information Technology

# OOP Principles Lab Practical Three - Information Hiding

Objective:
The objectives of this tutorial are to allow students to be able to:

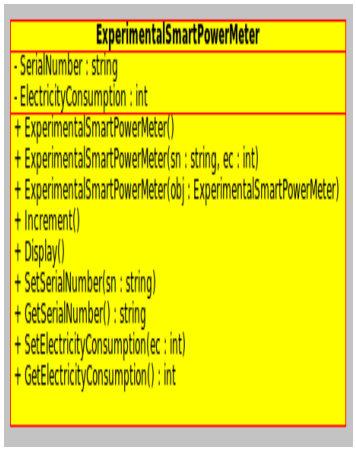
- show how to implement and call the three types of constructors
- give the student more practice in implementing composition
- demonstrate how to work with static values
- show how classes can be placed in separate files
- place main() in a driver file

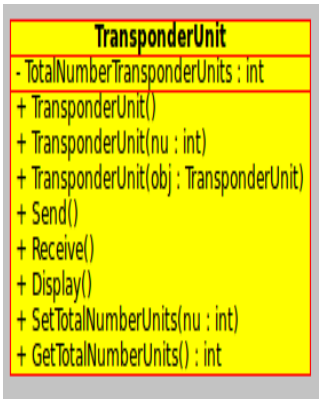
In the last lecture, you were introduced to the various types of constructors and other methods (accessors and mutators). You were also introduced to composition. This lab demonstrates the various constructors in action and gives you more practice with composition. It also introduces static values.

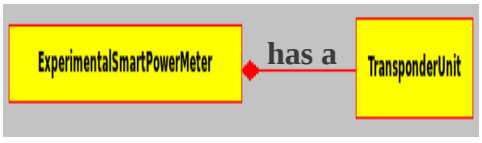
## Exercise One

Enter and run both implementations (in C++ and Java) of the UTech experimental power meter system below. Trace through the program and see if you can determine why the information displayed in the output appears as it does.

## Design using UML



C++ Implementation

***//TransponderUnit.h***

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

//TransponderUnit class
class TransponderUnit {
	//declare and initialize static attribute
	private:
		static int TotalNumberUnits;

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

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

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

		//send method
		void Send()
		{
		}

		//receive method
		void Receive()
		{
		}

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

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

```
            //get the total number of TotalNumberUnits
            int GetTotalNumberUnits()
            {
                  return TotalNumberUnits;
            }
};

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

```
//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;

	//declare the methods in class
	public:
		//default constructor
		ExperimentalSmartPowerMeter()
		{
			SerialNumber = "00000";
			ElectricityConsumption = 0;
			TU.SetTotalNumberUnits(1);
		}

		//primary constructor
		ExperimentalSmartPowerMeter(string sn, int ec, int ntu)
		{
			SerialNumber = sn;
			ElectricityConsumption = ec;
			TU.SetTotalNumberUnits(ntu);
		}

		//copy constructor
		ExperimentalSmartPowerMeter(ExperimentalSmartPowerMeter &obj)
		{
			SerialNumber = obj.SerialNumber;
			ElectricityConsumption = obj.ElectricityConsumption;
			TU.SetTotalNumberUnits(obj.TU.GetTotalNumberUnits());
		}

		//method to increment electricity consumption
		void Increment()
		{
			++ElectricityConsumption;
		}

		//method to display attributes
		void Display()
		{
			cout << "This meter's serial number is " << SerialNumber << endl;
			cout << "Electricity consumption is " << ElectricityConsumption
			 << endl;
			cout << "Total transpondors online is " <<
```

```
                TU.GetTotalNumberUnits() << endl;
        }

        //set the serial number
        void SetSerialNumber(string sn)
        {
                SerialNumber = sn;
        }

        //get the serial number
        string GetSetSerialNumber()
        {
                return SerialNumber;
        }

        //set the electricity consumption
        void SetElectricityConsumption(int ec)
        {
                ElectricityConsumption = ec;
        }

        //return the electricity consumption
        int SetElectricityConsumption()
        {
                return ElectricityConsumption;
        }
};
```

***//MetroDriver.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();

	//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;
}
```

Java Implementation

```
//TransponderUnit.java

package powermeterproject;

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

	//default constructor
	public TransponderUnit()
	{
		++TotalNumberUnits;
	}

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

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

	//send method
	public void Send()
	{
	}

	//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 "+
			TotalNumberUnits);
	}

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

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

```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

	//default constructor
	public ExperimentalSmartPowerMeter()
	{
		SerialNumber = "00000";
		ElectricityConsumption = 0;
		TU.SetTotalNumberUnits(1);
	}

	//primary constructor
	public ExperimentalSmartPowerMeter(String sn, int ec, int ntu)
	{
		SerialNumber = sn;
		ElectricityConsumption = ec;
		TU.SetTotalNumberUnits(ntu);
	}

	//copy constructor
	public ExperimentalSmartPowerMeter(ExperimentalSmartPowerMeter obj)
	{
		SerialNumber = obj.SerialNumber;
		ElectricityConsumption = obj.ElectricityConsumption;
		TU.SetTotalNumberUnits(obj.TU.GetTotalNumberUnits());
	}

	//method to increment electricity consumption
	public void Increment()
	{
		++ElectricityConsumption;
	}

	//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 transpondors online is " +
			TU.GetTotalNumberUnits());
```

```
	}

	//set the serial number
	public void SetSerialNumber(String sn)
	{
		SerialNumber = sn;
	}

	//get the serial number
	public String GetSetSerialNumber()
	{
		return SerialNumber;
	}

	//set the electricity consumption
	public void SetElectricityConsumption(int ec)
	{
		ElectricityConsumption = ec;
	}

	//return the electricity consumption
	public int SetElectricityConsumption()
	{
		return ElectricityConsumption;
	}

}
```

```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();

		//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();
	}

}
```

Exercise Two

a) Comment out all the code in the `MeterDriver.java` file.
b) Write a new driver with main().
c) Create three new objects (A, B, C) of the `TransponderUnit` class, calling only the default constructor.
d) Create a new object D of the `TransponderUnit` class, calling only the primary constructor.
e) Create a new object E of the `TransponderUnit` class, calling only the copy constructor.
f) Use the Display() method to display the value of the attributes after each of the above objects are created
g) Trace through the program. What do you notice?

Exercise Three – homework

Examine the individual and relationship UML diagrams on page one. Without looking at the sample code, write the entire system by yourself. Make the `TotalNumberUnits` attribute static. Run your program and compare it with the sample above. Practice writing this code until you can successfully write it in its entirety on your own.