Object Oriented Programming Concepts

Lecture One

Introduction to Object Oriented Programming

(c) 2011. David W. White
School of Computing and Information Technology
Faculty of Engineering and Computing
University of Technology, Jamaica

Email: dwwhite@utech.edu.jm

Object Oriented Programming Concepts

Expected Outcome

At the end of this lecture the student should be able to:

- Explain in general terms the differences between the Object-Oriented programming paradigm and other programming language paradigms, particularly the procedural language paradigm.
- Students should be able to list and explain the main characteristics of the object oriented programming paradigm
- Be able to explain with the use of diagrams the process of object-oriented analysis, object oriented design and implementation of the design in an object oriented language.

Object Oriented Programming Concepts

Topics to be covered in this lecture:

- Programming Language Paradigms
- Characteristics of the paradigms
- Comparison between the Object-Oriented paradigm and others programming language paradigms
- Examples of languages belonging to each paradigm
- Developing Applications in Object-Oriented Languages
- Object-Oriented Analysis
- Object-Oriented Design
- Implementing the Design with an Object-Oriented Language

Programming Language Paradigms

- A programming language is a notational system used to describe computation in both machine-readable and human-readable form.
- Historically, there have been three types of programming languages:
  - Machine languages
  - Assembly Languages
  - High-level Languages

Programming Language Paradigms

Machine languages

- The only language the computer really understands directly

- Is hardware dependent

- Written in binary

Programming Language Paradigms

Assembly Languages

- Uses mnemonics to represent machine language instructions

- Symbolic labels used for memory addresses

- As a result, is also machine dependent

- Must be translated into machine language to be able to be executed on the target computer
- encouraged spaghetti code

Programming Language Paradigms

High-level Languages

- Source code is machine independent

- Programs written with instructions closely resembling natural languages and mathematics

- Must be translated to machine language to be executed

Programming Language Paradigms

A programming language paradigm is a particular style of programming in which the concepts and abstractions differ

- Imperative
  (also called Algorithmic and Procedural)
- Declarative (includes Functional and Logical)
- Object-Oriented
- Others

Characteristics of the paradigms

Imperative Programming Paradigm

- Describes computation as a series of steps, that is, as an algorithm, hence also called algorithmic programming paradigm
- Sometimes Procedural Programming is used as a synonym for Imperative Programming
- Formally, Procedural Programming is a type imperative programming in which the computational steps are placed in procedures

Characteristics of the paradigms

Imperative Programming Paradigm

- Modifies state e.g. through variable assignment
- Early machine languages were the first imperative programming languages
- By extension assembly language was imperative programming
- Early high-level languages were also based on the imperative programming language paradigm

Characteristics of the paradigms

Declarative Programming Paradigm

- Involves stating ***what*** computation is to be done without specifying ***how*** it is to be done
- Lacks side-effects

  i.e. state is not modified
- Includes functional and logical programing

Characteristics of the paradigms

Object-Oriented Programming Paradigm

- Focuses on on “objects” - which are collections of data items and the operations that can be performed on them
- Data elements of objects are called attributes
- Operations belonging to an object are called methods
- Messages can be sent to an object to tell it to perform a particular operation or method

Characteristics of the paradigms

Some of the Main Features of Object-Oriented Programming

- Inheritance
- Polymorphism
- Dynamic Dispatch (or Dynamic Binding)
- Encapsulation
- Abstraction
- Modularity

Comparison between the object-oriented paradigm and other paradigms

Imperative	**Declarative**	**Object-Oriented**
Focus is on commands to accomplish the task (imperatives)	Focus is what not how to accomplish the task, using mathematical functions or logical relations	Focus is on the data items (objects)
Has side-effects (procedures can modify values in memory called variables)	Does not have side-effects (functions or relations do not modify values in memory but may transform the values into new ones)	Has-side effects (methods can modify values in objects called attributes)
Operations are specified in one of more procedures		Operations are placed in methods which belong to a class
Computation is performed by executing commands that modify variables in memory	Computation is performed by evaluating mathematic functions (functional programming) or predicate logic (logic programming)	Computation is performed by passing messages to objects

Examples of languages belonging to each paradigm

- **Imperative/Algorithmic/Procedural**

  C, Pascal, BASIC, FORTRAN, COBOL, Algol, PL/1, Modula-3, etc.

- **Object-Oriented**

  C++, Java, C#, SmallTalk, Simula, Eiffel, Ruby, etc.

- **Functional**

  LISP, Haskel, ML, Miranda, APL

- **Logical**

  Prolog, Godel

Examples of languages belonging to each programming paradigm

- Sometimes a programming language may support multiple programming paradigms

  - e.g. C++

- Both procedural and object-oriented programs can be written in C++ and are supported by the C++ standard
- This contrasts with C and Java, which only support procedural and object-oriented programming respectively

Developing Applications in Object-Oriented Languages

- Object-oriented Analysis
- Object-oriented Design
- Implementation in an Object-Oriented Programming Language (OOP)

Object-Oriented Analysis

- In object-oriented analysis (OOA), the analyst examines the written requirements and extracts the class objects and their elements and interactions.
- Formal methods such as those developed by Coad and Yourdon can be used
- OOA can be performed by conducting a textual parse of the requirements. This textual parse is also called a noun-verb analysis
- Nouns become classes, objects or attributes, while verbs become methods

Object-Oriented Analysis

Coad and Yourdon's Six Characteristics for Selecting an Item as an Object during OOA

- Retained Information
- Needed Services
- Multiple Attributes
- Common Attributes
- Common Operations
- Essential Requirements

Object-Oriented Analysis

OOA/Textual Parse/Noun-Verb Analysis Example

- A digital clock keeps track of the hour and the minute. A person can set the hour, set the minute, and view both the hour and the minute.
- Required: Perform an object-oriented analysis on the above paragraph, showing what are the likely classes, attributes and methods

Object-Oriented Analysis

A *digital clock* keeps track of the *hour* and the *minute*. A *person* can set the *hour*, set the *minute*, and view both the hour and the *minute*.

- All the nouns in the text above are underlined and italicized However, person is not a component of the system to be built so it is rejected.
- The class identified is digital clock, and the attributes identified are hour and minute

Object-Oriented Analysis

A digital clock keeps track of the hour and the minute. A person can set the hour, set the minute, and view both the hour and the minute.

- All the verbs in the text above are outlined. However, “keeps track” does not describe an essential part of the system so it is rejected.
- The methods identified are therefore “set hour”, “set minute” and “view”

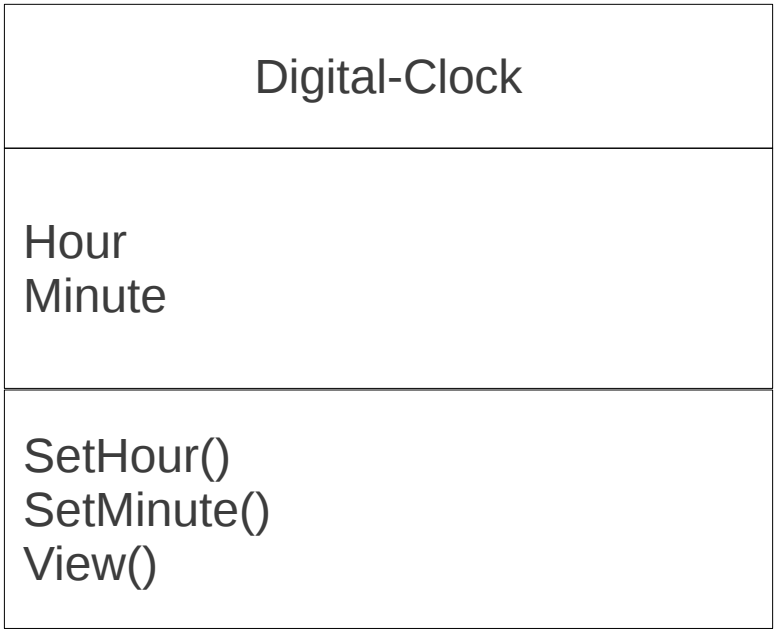
Object-Oriented Design

- In object-oriented design, the system to be built is modeled
- The classes and their relationships are usually modeled using the Unified Modeling Language (UML)
- UML is a diagrammatic language
- Sometimes the analysis and design phases are combined into object-oriented analysis and design (OOAD)

Object-Oriented Design

- Model each class in UML
- The class identified is digital clock, and the attributes identified are hour and minute
- The methods identified are therefore “set hour”, “set minute” and “view”

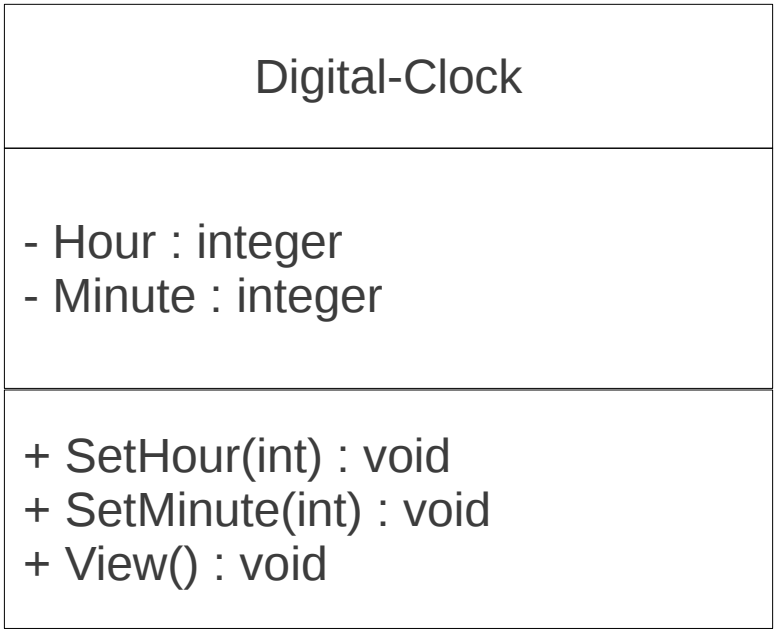
UML Diagram for digital clock example done in OOA phase



Object-Oriented Design

- As the design is refined, the data type and access specifier for the members of the class are defined
- The “-” indicates “private” while the “+” indicates public
- Hour and Minute are stated to be integers

UML Diagram for digital clock example done in OOA phase



Implementing the Design with an Object-Oriented Language

- The model of the system in UML is implemented in the object-oriented programming language of choice by converting the objects and their relationships into corresponding appropriate programming language constructs
- The code is then compiled into machine language or some other internal representation and executed on the target computer

Implementing the Design with an Object-Oriented Language

C++ code

```
//Digital clock class
//By D. White 2011 Aug 26

#include <iostream>
using namespace std;

class DigitalClock
{
	private:
		int Hour;
		int Minute;
	public:
		void SetHour(int h)
		{
			Hour  = h;
		}
		void SetMinute(int m)
		{
			Minute = m;
		}
		void Show()
		{
			cout << Hour << ":" << Minute << endl;
		}
};
```

Java code

```
//Digital clock class
//By D. White 2011 Aug 26

import java.lang.*;

public class DigitalClock
{

	int Hour;
	int Minute;
	public void SetHour(int h)
	{
		Hour  = h;
	}
	public void SetMinute(int m)
	{
		Minute = m;
	}
	public void Show()
	{
		System.out.println(Hour + ":" + Minute);
	}
}
```