2

Object Oriented Principles

Objectives

After completing this lesson, you should be able to do the following:

- Define objects and explain how they are used
- Associate objects so that they can communicate and interact via messages
- Define classes and explain how they are used
- Describe object-oriented principles: classes, objects, and methods
- Describe the value of reusable software components
- Examine the object-oriented model that is used in this course

Structured Programming

- Structured Programming, as name suggests, is a technique that is considered as precursor to OOP and usually consists of well-structured and separated modules.
- In this programming, user can create its own user-defined functions as well as this methodology tries to resolve issues that are associated with unconditional transfers to allow programmers follow logic of programs.
- It also requires more discipline at the design and logical structuring stage.
  - **Example :** Pascal, ALGOL, C, Modula-2, etc.

- Structured programming (sometimes known as modular programming) is a programming paradigm that facilitates the creation of programs with readable code and reusable components.
- All modern programming languages support structured programming, but the mechanisms of support, like the syntax of the programming languages, varies. Structured programming encourages dividing an application program into a hierarchy of modules or autonomous elements, which may, in turn, contain other such elements.
- Within each element, code may be further structured using blocks of related logic designed to improve readability and maintainability.

Modular Programming

- Modular programming, which is today seen as synonymous with structured programming, emerged a decade later as it became clear that reuse of common code could improve developer productivity.
- In modular programming, a program is divided into semi-independent modules, each of which is called when needed. C is called structured programming language because a program in c language can be divided into small logical functional modules or structures with the help of function procedure.

Disadvantages of structured programming

1. A high level language has to be translated into the machine language by translator and thus a price in computer time is paid.

2. The object code generated by a translator might be inefficient compared to an equivalent assembly language program.

3. Data types are proceeds in many functions in a structured program. When changes occur in those data types, the corresponding change must be made to every location that acts on those data types within the program. This is really a very time consuming task if the program is very large.

4. Let us consider the case of software development in which several programmers work as a team on an application. In a structured program, each programmer is assigned to build a specific set of functions and data types.
5. Since different programmers handle separate functions that have mutually shared data type. Other programmers in the team must reflect the changes in data types done by the programmer in data type handled. Otherwise, it requires rewriting several functions

Characteristics

1. It is a subset of procedural programming.
2. Programs are divided into small programs or functions.
3. It is all about facilitating creation of programs with readable code and reusable components.
4. Its main aim is to improve and increase quality, clarity, and development time of computer program.
5. It simply focuses on functions and processes that usually work on data.
6. It is a method of organizing, managing and coding programs that can give or provide much easier modification and understanding.
7. In this, methods are written globally and code lines are processed one by one i.e., Run sequentially.

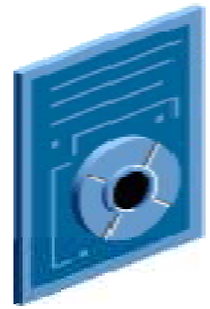
8. It generally follows “Top-Down Approach”.
9. It provides less flexibility and abstraction as compared to object-oriented programming.
10. It is more difficult to modify structured program and reuse code as compared to object-oriented programs.
11. It gives more importance of code.

Modeling

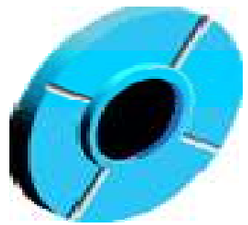
- Models perform the following functions:
  - Describe exactly what a situation needs
  - Facilitate discussion
  - Prevent mistakes
- Modeling and implementation are treated separately.
- Before coding can begin, the model must be correct.

Classes and Objects

- A class:
  - Models an abstraction of objects
  - Defines the attributes and behaviors of objects
  - Is the blueprint that defines an object



- An object:
  - Is stamped from the class mold
  - Is a single instance of a class
  - Retains the structure and behavior of a class



An Object's Attributes Maintain Its State

- Objects have knowledge about their current state.
- Each piece of knowledge is called an *attribute*.
  - The values of attributes dictate an object's state.

Object: My blue pen

Attribute: Ink amount

Object: Acme Bank ATM

Attribute: Cash available

Objects Have Behavior

- An object exists to provide behavior (functionality) to the system.
- Each distinct behavior is called an *operation*.

Object: My blue pen

Operation: Write

Object: Acme Bank ATM

Operation: Withdraw

Objects Are Modeled As Abstractions

- A Java object is modeled as an abstract representation of a real-world object.
- Model only those attributes and operations that are relevant to the context of the problem.

Problem domain: Product catalog
Real-world attributes and operations that you may want to model:

- **Attributes: Model, manufacturer, price**
- **Operations: Change price**

Real-world attributes and operations that you may *not* want to model:

- **Attributes: Ink color**
- **Operations: Refill, change color, point, write**

Defining Object Aggregation:

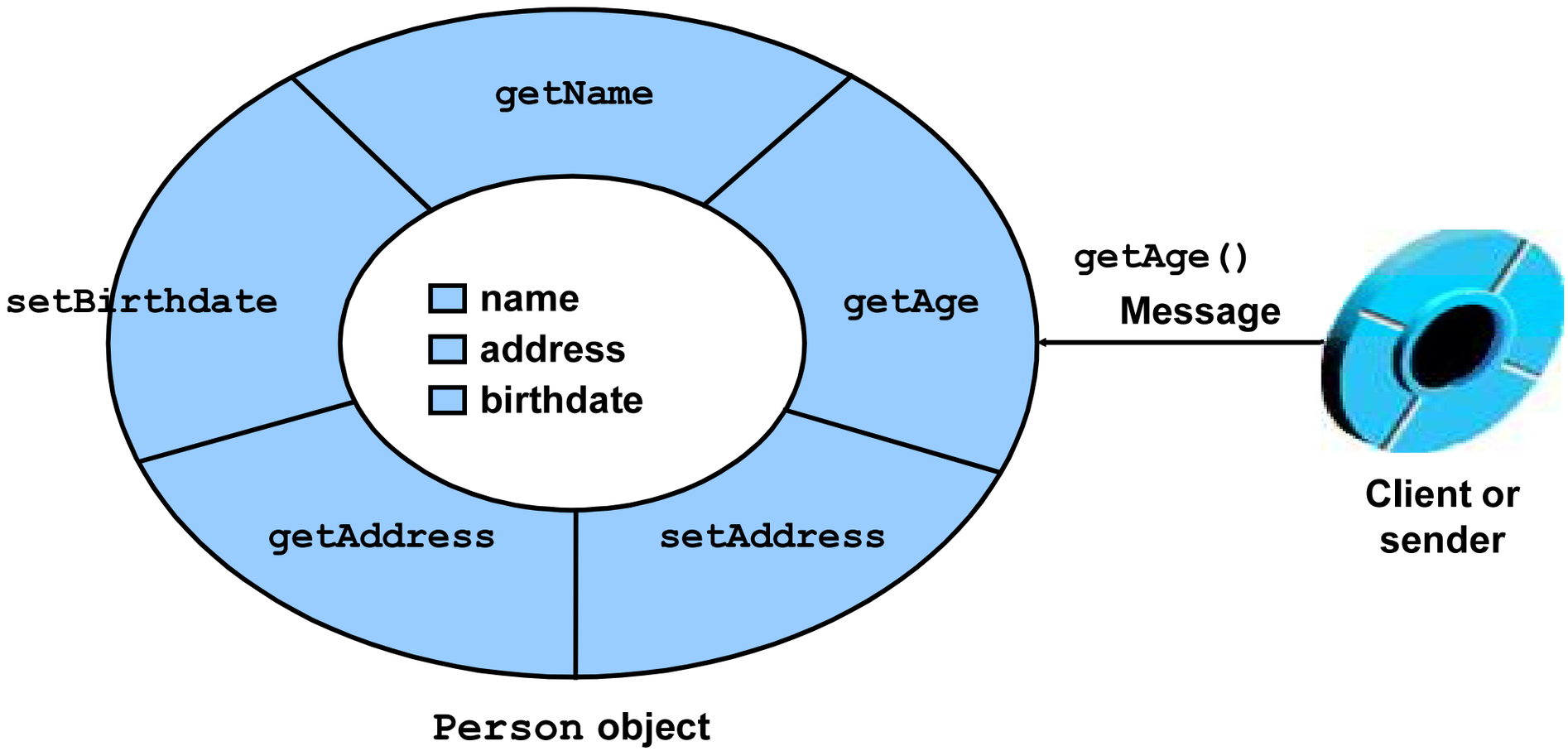
- Objects can be composed of other objects.
- Objects can be part of other objects.
- This relationship between objects is known as *aggregation*.
- Strong aggregation is called *composite aggregation*.

A PC may be an object.

A PC may have a keyboard, mouse, and network card, all of which may be objects.

A PC may have a CD drive, which may be an object.

Donut Diagram



Guided Practice:

Collaborating Objects

Collaborating objects work together to complete a task and form the basis of an application system.

- All methods are defined in a class and are not defined globally as in traditional languages.
- All objects are created from classes and contain all the attributes and methods of their class.
- Objects must associate with each other to collaborate on common tasks.
- Associated objects communicate by sending messages.

Objects Interact Through Messages

- Objects communicate by sending messages.
- A sending object must be associated with or linked to the receiving object.
- The message sender requests the receiver to perform the operation that is named in the message.
- This communication is similar to calling a procedure:
  - The sender calls a method of the receiver.
  - The receiver executes the called method.
- Calling a method is always in the context of a particular object:
  - `myPen.write( )`: Object-oriented programming
  - `write (myPen)`: Traditional structured programming

Classes

- A class is a template for objects.
- A class definition specifies the operations and attributes for all instances of that class.
- A class is used to manage complexity.

When you create *my blue pen,* you do not have to specify its operations or attributes. You simply say what class it belongs to.

Identifying a Class

- Identify the common behavior and structure for a group of objects.
- Recognize a single coherent concept.
- Caution: A common misconception is the use of the words *classes* and *objects* interchangeably. Classes *define* objects.

My blue pen	**ops:**	**write, refill**
	attribs:	**ink amount, color of ink**
Your blue pen	**ops:**	**write, refill**
	attribs:	**ink amount**

Comparing Classes and Objects

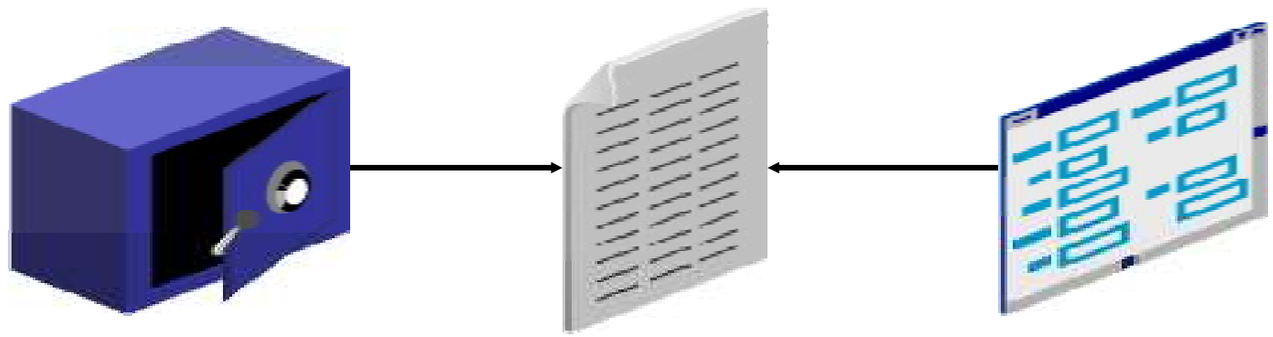
- A class is a static definition that you can use to understand all the objects of that class.
- Objects are the dynamic entities that exist in the real world and your simulation of it.
- Caution: In object-oriented programming, people almost always use the words *classes* and *objects* interchangeably. You must understand the context to differentiate between the two terms.

Data Encapsulation / Data Hiding / Data Abstraction

Encapsulation hides the internal structure and operations of an object behind an interface.

- A bank ATM is an object that gives its users cash.
  - The ATM hides (encapsulates) the actual operation of withdrawal from the user.
  - The interface (way to operate the ATM) is provided by the keyboard functions, screen, cash dispenser, and so on.
  - Bypassing encapsulation is bank robbery.
- Bypassing encapsulation in object-oriented programming is impossib

- There may be a commonality between different classes.
- Define the common properties in a superclass.

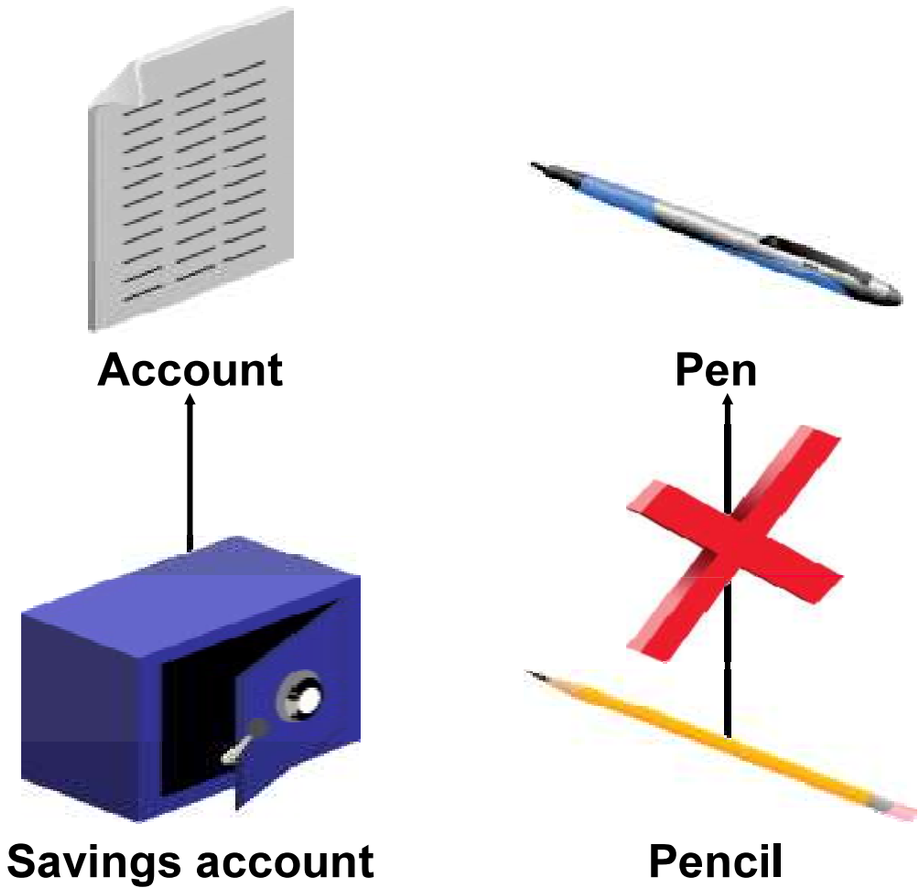


Savings account **Account** **Checking account**

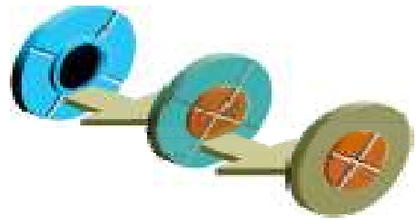
- The subclasses use inheritance to include those properties.

Using the "Is-a-Kind-of" Relationship

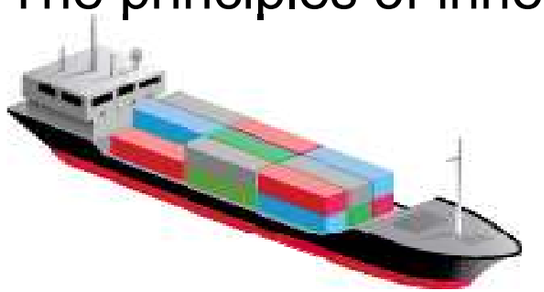
- A subclass object "is-a-kind-of" superclass object.
- All the attributes and behaviors of the superclass must also apply to the subclass.

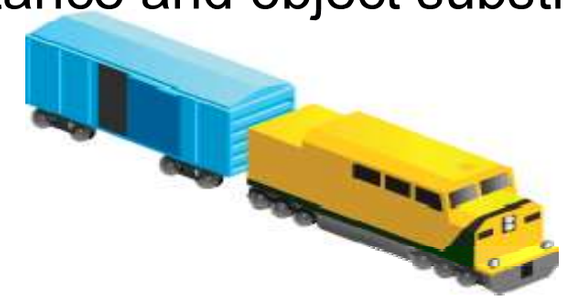


Polymorphism



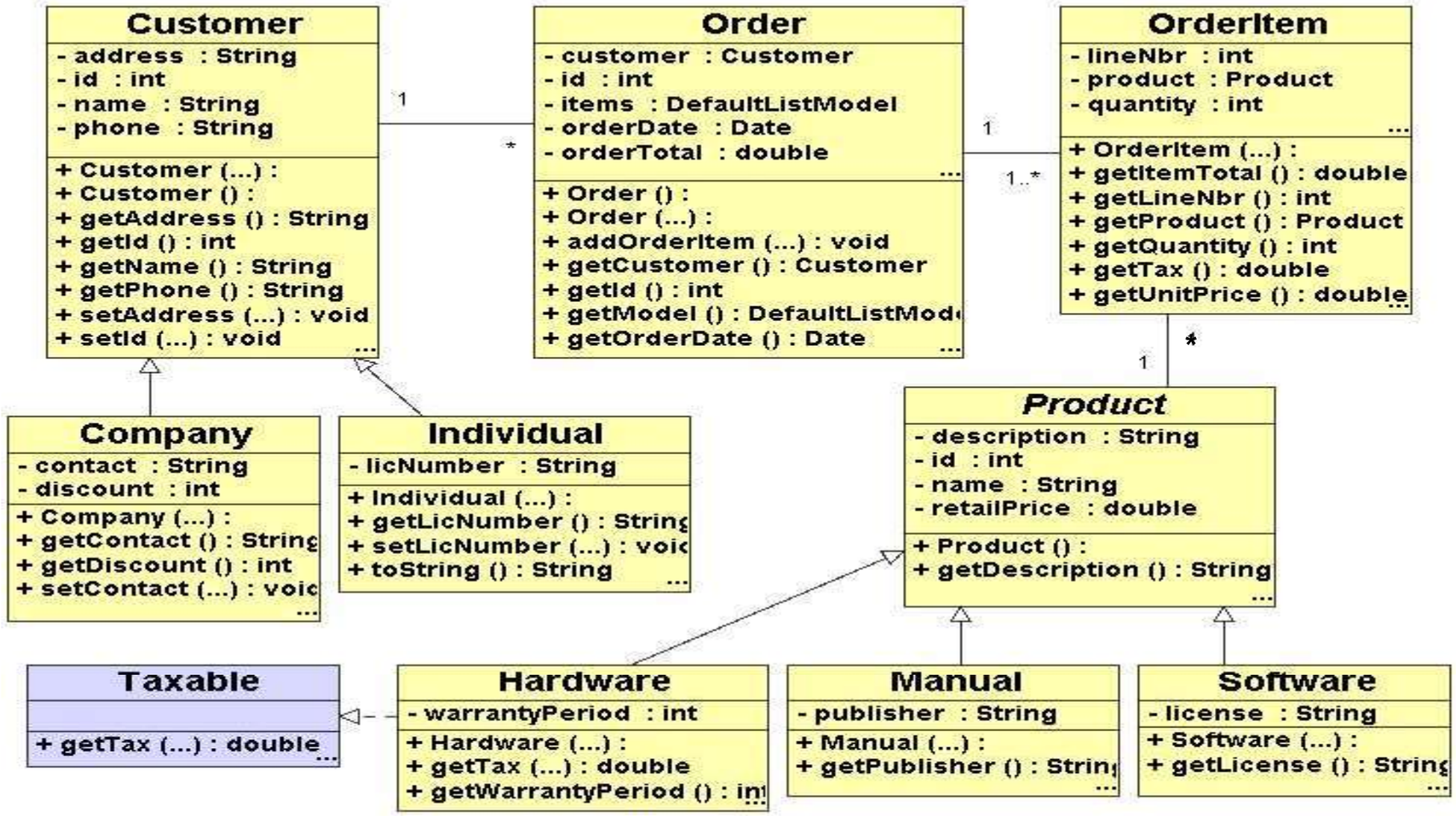
- Polymorphism refers to:
  - Many forms of the same operation
  - The ability to request an operation with the same meaning to different objects. (However, each object implements the operation in a unique way.)
  - The principles of inheritance and object substitution





Load cargo

UML Diagram for `OrderEntry`



Summary

In this lesson, you should have learned the following:

- An object is an abstraction of a real-world object.
- A class is a template or blueprint for objects.
- Classes form inheritance trees: Operations that are defined in one class are inherited by all subclasses.
- Polymorphism frees the caller from knowing the class of the receiving object.

Practice : Overview

This practice covers the following topics:

- Identifying business objects for the Order Entry system
- Identifying methods for the classes
- Identifying attributes for the classes
- Searching for inheritance in the classes
- Examining the UML class model for the course application