Chapter 7

Object-Oriented Programming

Object-Oriented Programming

Objectives

After completing this unit you will be able to:

- **Describe the role of objects in modeling the real world.**
  - Explain how objects facilitate the development of reusable software components.
  - Explain the fundamental concepts of abstraction and encapsulation.
- **Describe the concept of a class and its relationship to an object.**
  - Explain the concept of inheritance, and describe other important relationships among classes.
- **Define the term polymorphism and explain how it can be used to make object oriented programs more flexible and easy to maintain.**
- **Discuss the process of object-oriented analysis and design.**

Objects

- **Objects have both a real-world and a software meaning**
- **An object model can describe a relationship between the two.**
- **This section summarizes the key terminology of objects.**

Objects in the Real World

- **The term *object* has an intuitive real world meaning.**
  - There are concrete, tangible objects such as a ball, an automobile and an airplane.
  - There are more abstract objects that have a definite intellectual meaning, such as a committee, a patent or an insurance contract.
- **Objects have both attributes or characteristics, and operations that can be performed upon them.**
  - A ball has a size, a weight, a color, etc.
  - Operations may be performed on the ball such as throw, catch, drop, etc.
- **There can be various relationships among classes of objects.**
  - A specialization relationship, such as an automobile is a special kind of vehicle.
  - A whole/part relationship, such as an automobile consists of an engine, a chassis, wheels, etc.

Object Models

- **Objects can also be used in programs.**
  - We've already seen a little about classes and objects in C# programs.
- **Objects are useful in programming because you can set up a software *model* of a real world system.**
  - Objects in software correspond to objects in the real world.
  - Explicitly describing the real world system in terms of objects helps you to understand the system more explicitly and precisely.
  - The model can then be implemented as actual software using a programming language.
  - A software system implemented in this way tends to be more faithful to the real system, and it can be changed more readily when the real system is changed.
- **There are formal languages for describing object models.**
  - The most popular language is UML (Unified Modeling Language, which is a synthesis of several earlier modeling languages).
  - Formal modeling languages are beyond the scope of this course, but we will find that informal models are useful.

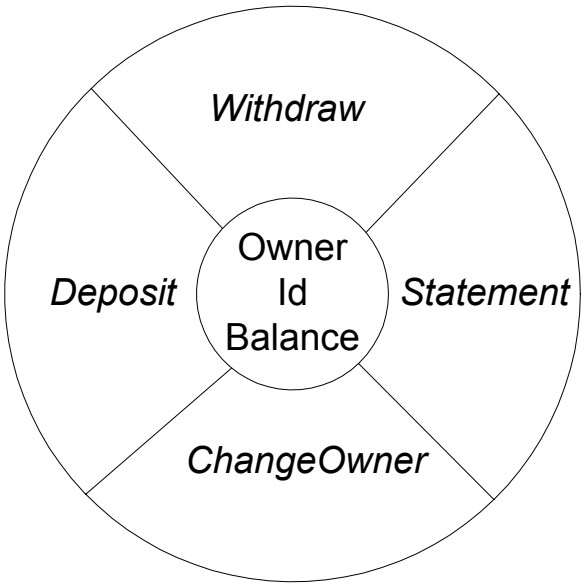
Reusable Software Components

- **Another advantage of objects in software is that they can facilitate reusable software components.**
- **Hardware has long enjoyed significant benefits from reusable hardware components.**
  - For example, computers can be created from power supplies, printed circuit boards, etc.
  - Printed circuit boards in turn can be created from chips.
  - The same chip can be reused in many different computers, and new hardware designs do not have to be done from scratch.
- **With appropriate software technology similar reuse is feasible in software systems.**
- **Objects provide the foundation for software reuse.**

Objects in Software

- **An object is a software entity containing data and related functions as a self contained module.**

Bank Account Object

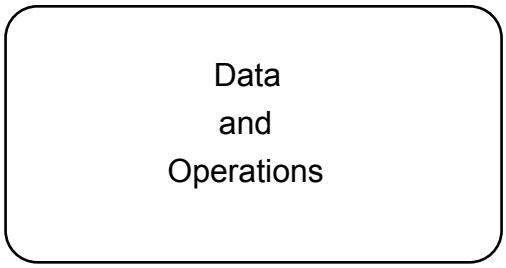


- **Objects hold *state* and specify *behavior*.**

- **Objects provide the means for *abstraction*, *encapsulation*, and *instantiation*.**

State and Behavior

- **An object has data, i.e. a set of properties or attributes, which are its essential characteristics.**
  - The state of an object is the value of these attributes at any point in time.
- **The behavior of an object is the set of operations or responsibilities it must fulfill for itself and for other objects.**
- **The data and operations are packaged together.**



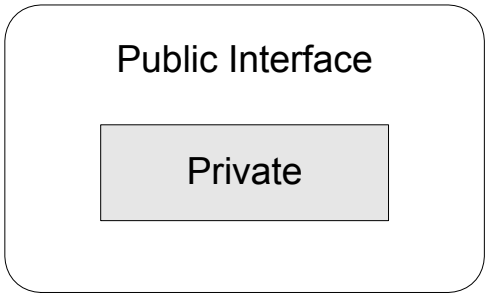
  - As part of software design, this packaging aids conceptualization and abstraction.
  - Disparate items are turned into a conceptual unit.

Abstraction

- **An abstraction captures the essential features of an entity, suppressing unnecessary details.**
- **All instances of an abstraction share these common features.**
- **Abstraction helps us deal with complexity.**

Encapsulation

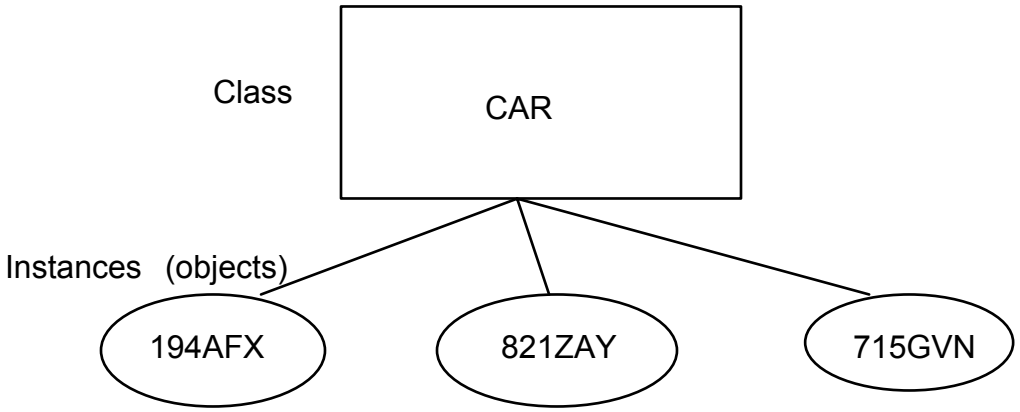
- **The implementation of an abstraction should be hidden from the rest of the system, or encapsulated.**
- **Objects have a public and a private side.**
- **Public side is what the rest of the system knows, while private side implements the public side.**



- **Data itself is private, walled off from the rest of the program.**
- **Data can only be accessed through functions with a public interface.**
- **There are two kinds of protection:**
  - Internal data is protected from corruption.
  - Users of the object are protected from changes in the representation.

Classes

- **A *class* groups all objects with common behavior and common structure.**
- **A class allows production of new objects of the same type. An *object* is an instance of some class.**

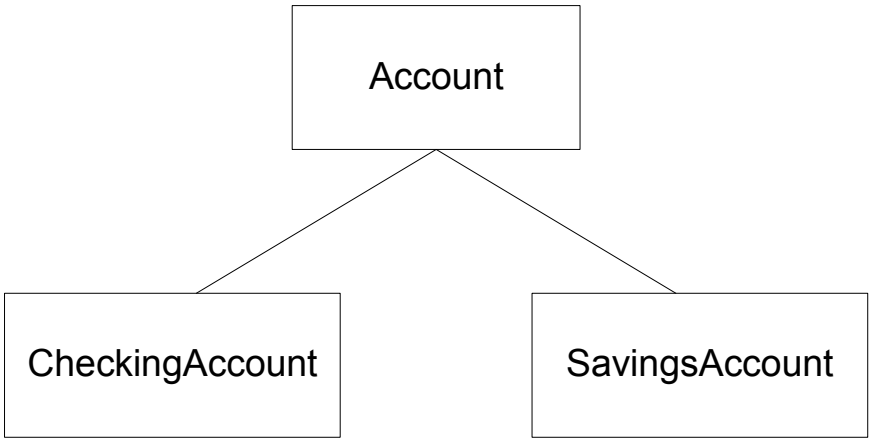


- **We refer to the process of creating an individual object as *instantiation*.**

Inheritance Concept

- **Inheritance is a key feature of the object oriented programming paradigm.**
  - You abstract out common features of your classes and put them in a high level base class.
  - You can add or change features in more specialized derived classes, which "inherit" the standard behavior from the base class.
  - Inheritance facilitates code reuse and extensibility.
- **Consider *Account* as a base class, with derived classes *CheckingAccount* and *SavingsAccount***
  - All accounts share some characteristics, such as balance.
  - Different kinds of accounts differ in other respects. For example, a checking account has a monthly fee, while a savings account pays interest at a certain rate.

Inheritance Example



Balance	Account

Balance	CheckingAccount
Fee	

Balance	SavingsAccount
Rate	

Relationships Among Classes

- **Classes may be related to each other in various ways**
- **The inheritance (IS-A) relationship specifies how one class is a special case of another class**
  - A **CheckingAccount** (subclass or derived class) is a special kind of **Account** (superclass or base class)
- **The composition (HAS-A) relationship specifies how one class (the whole) is made up of other classes (the parts)**
  - A **Bank** (whole) has a list of **Account** objects.
- **A weaker kind of relationship (USES-A) can be identified when one class merely makes use of some other class when carrying out its responsibilities.**

Polymorphism

- **Consider the problem of generating monthly statements for different kinds of accounts.**
  - Checking and savings accounts differ, with one possibly resulting in a fee and the other in a posting of interest.
- **A traditional approach is to maintain a type field in an account structure and to perform processing in a switch statement, with cases for each type.**
  - Such use of switch statements is error prone and requires much maintenance when adding a new account type.
- **An alternative is to localize the intelligence to generate a statement in each account class, which will support its own *GetStatement* method.**
  - Generic monthly statement code can then be written that will handle different types of accounts and will not have to be modified to support an additional account type.
  - Provide a **GetStatement** method in the base class and an override of this method in each derived class.
  - Call **GetStatement** through an object reference to a general **Account** object.
  - Depending on the actual account class referred to, the appropriate **GetStatement** method will be called.

Polymorphism (Cont'd)

- **The ability for the same method call to result in different behavior depending on the object through which the method is invoked is referred to as *polymorphism*.**
- **Although somewhat advanced, polymorphism can greatly simplify complex systems and is an important part of the object-oriented paradigm.**

Object Oriented Analysis and Design

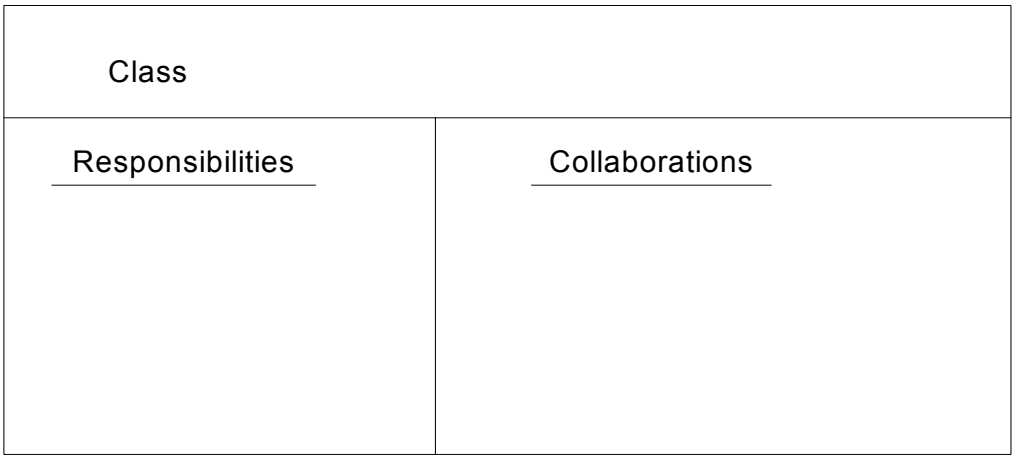
- **The process of creating an object model of a system that can be used as the foundation of implementing a corresponding software system is referred to as *object oriented analysis and design (OOAD)***
  - Analysis refers to the first part of the process in which the essential features of the system to be created are captured, without regard to details of implementation
  - Design refers to a refinement in which more details are provided, in preparation for actually coding the software
  - In practice the dividing line between the two is fuzzy, and in modern terminology the two are often grouped together under the phrase "object oriented analysis and design"
- **Key parts of OOAD include**
  - Identify "use cases" capturing the different ways in which the system will be used
  - Identify the most important **classes** in the system
  - Identify **responsibilities** of the classes
  - Identify classes that "**collaborate**" in carrying out the responsibilities of each class
  - Identify relationships among the classes in the system

Use Cases

- **The *use cases* describe how the system we are trying to model will be used**
- **A use case represents an interaction of a person or another element with the system**
  - The persons or entities doing the interactions are called **actors**
- **Use cases are used early in the analysis process to ensure that your system will do all the different things it is supposed to do**
- **Use cases can be used late in the implementation to help specify *tests* of the completed software**

CRC Cards and UML

- **A very popular "low tech" approach to the early stage of object oriented analysis and design is the use of *CRC Cards***
  - **CRC** stands for class – responsibilities – collaborations
- **A CRC card is typically 4" x 6" and has the following information**
  - Class name written at top (and possibly superclass)
  - Two columns, with responsibilities written in the first column and collaborating classes written in the second column



- **A more full-blown notation for object oriented analysis and design is Unified Modeling Language or UML.**
- **Details of OOAD are beyond the scope of this course.**

Summary

- **Objects can model entities in the real world.**
- **A software system based on objects tends to be more faithful to the real system, and it can be changed more readily.**
- **Objects also facilitate the development of reusable software components.**
- **An object is an instance of a class.**
- **Fundamental concepts of object-oriented programming include abstraction, encapsulation and inheritance.**
- **The process of object oriented analysis and design helps you find useful objects.**
- **Use cases describe how the system we are trying to model will be used.**
- **CRC cards document classes, responsibilities and collaborations.**