



Topic 1

Introduction to Object-Oriented Programming and Software Development

Animated Version

Objectives

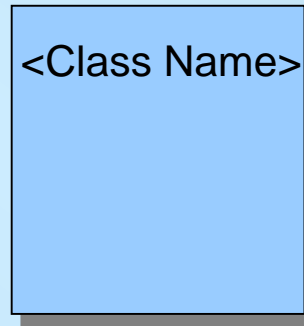
After you have read and studied this Topic, you should be able to

- Name the basic components of object-oriented programming
- Differentiate classes and objects.
- Differentiate class and instance methods.
- Differentiate class and instance data values.
- Draw program diagrams using icons for classes and objects
- Describe significance of inheritance in object-oriented programs
- Name and explain the stages of the software lifecycle

Classes and Objects

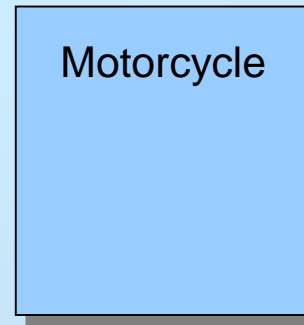
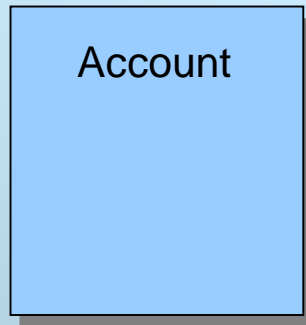
- Object-oriented programs use objects.
- An *object* is a thing, both tangible and intangible. Account, Vehicle, Employee, etc.
- To create an object inside the computer program, we must provide a definition for objects—how they behave and what kinds of information they maintain —called a *class*.
- An object is called an *instance* of a class.

Graphical Representation of a Class



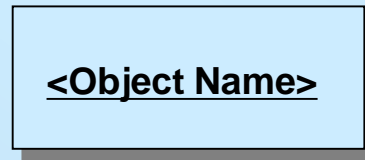
We use a rectangle to represent a class with its name appearing inside the rectangle.

Example:



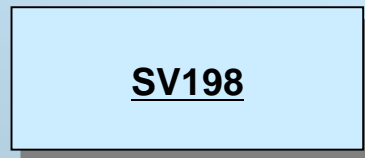
The notation we used here is based on the industry standard notation called *UML*, which stands for Unified Modeling Language.

Graphical Representation of an Object



We use a rectangle to represent an object and place the underlined name of the object inside the rectangle.

Example:



This is an object named SV198.

An Object with the Class Name

<Object Name> : <Class Name>

This notation indicates the class which the object is an instance.

Example:

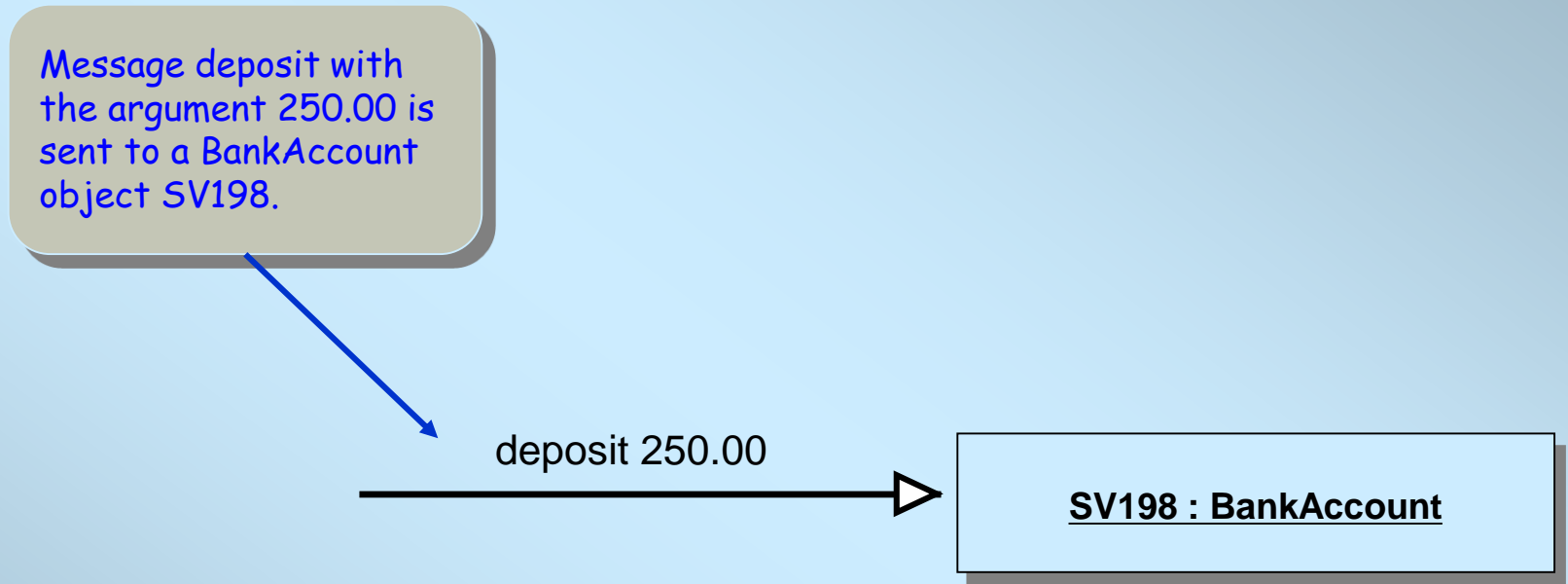
SV198 : BankAccount

This tells an object SV198 is an instance of the BankAccount class.

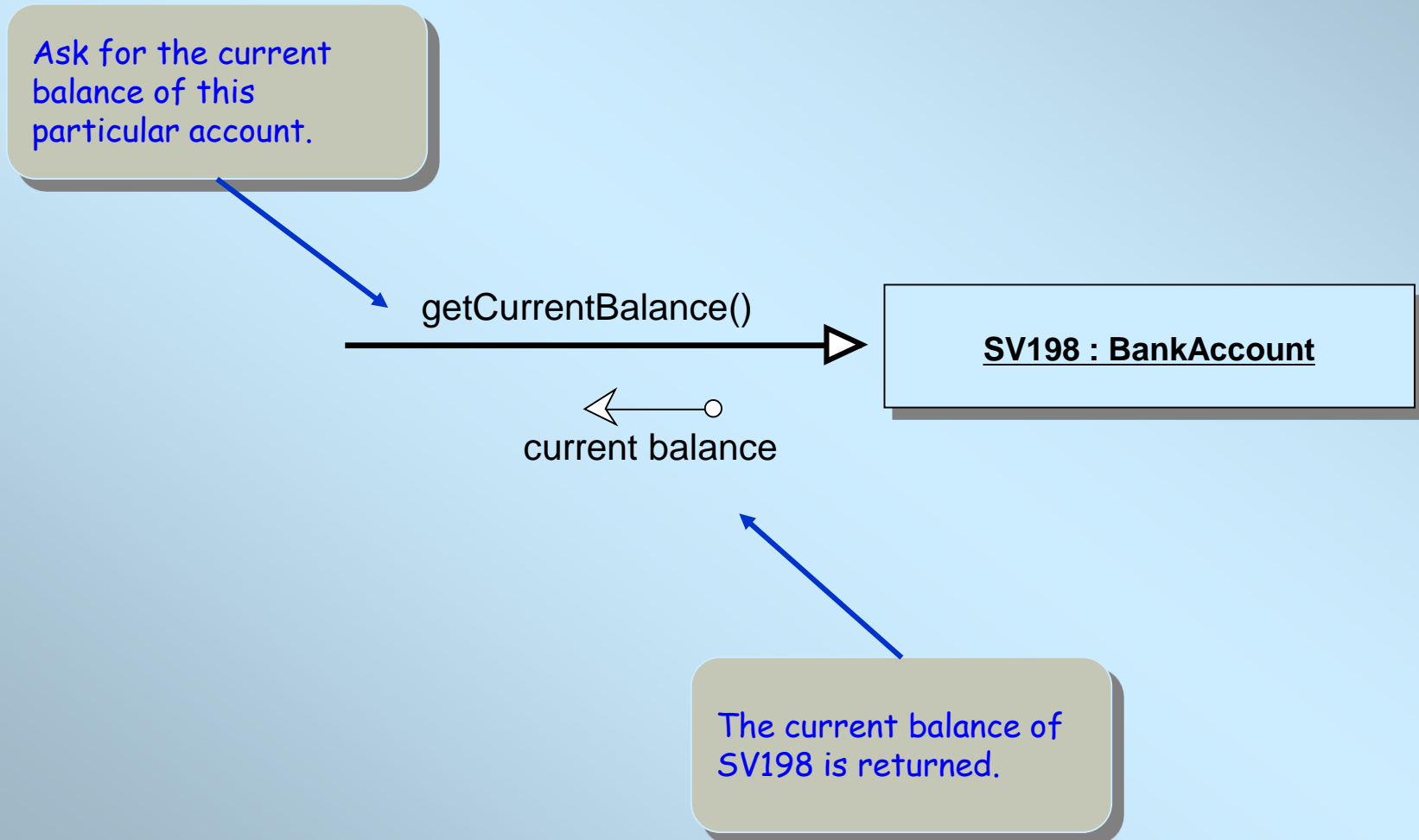
Messages and Methods

- To instruct a class or an object to perform a task, we send a *message* to it.
- You can send a message only to the classes and objects that understand the message you sent to them.
- A class or an object must possess a matching *method* to be able to handle the received message.
- A method defined for a class is called a *class method*, and a method defined for an object is called an *instance method*.
- A value we pass to an object when sending a message is called an *argument* of the message.

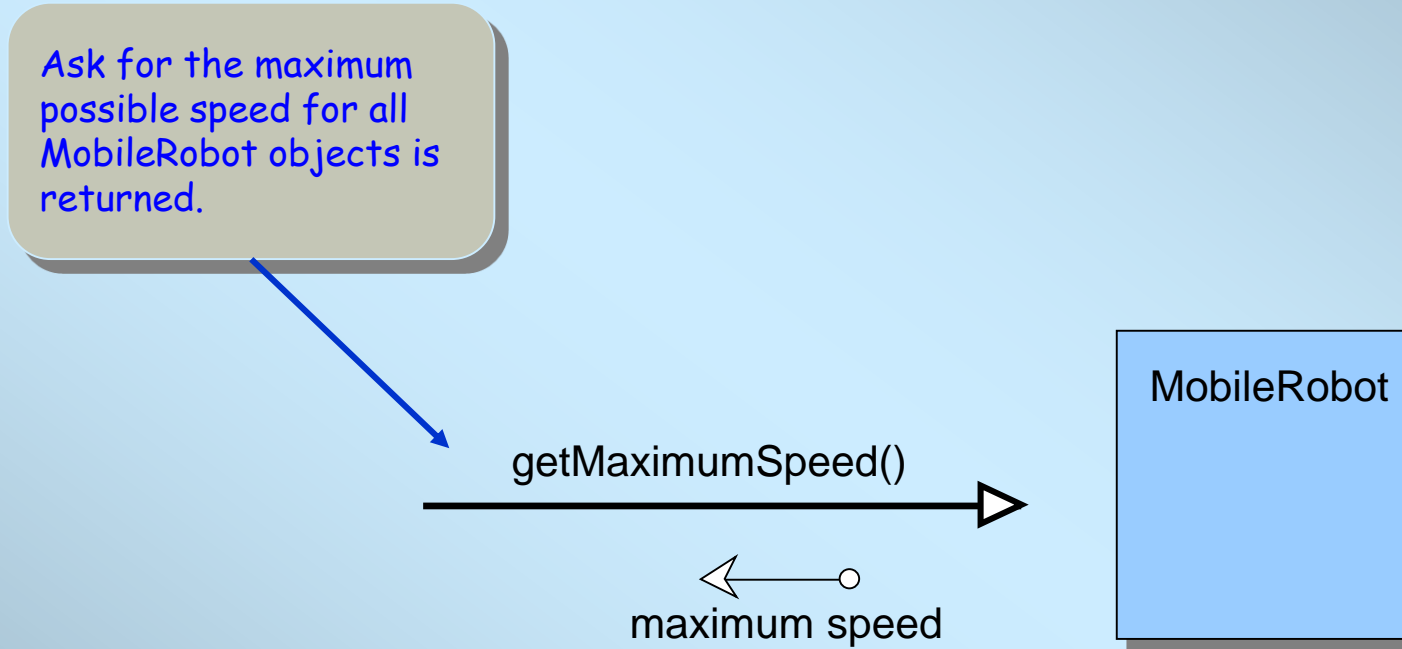
Sending a Message



Sending a Message and Getting an Answer



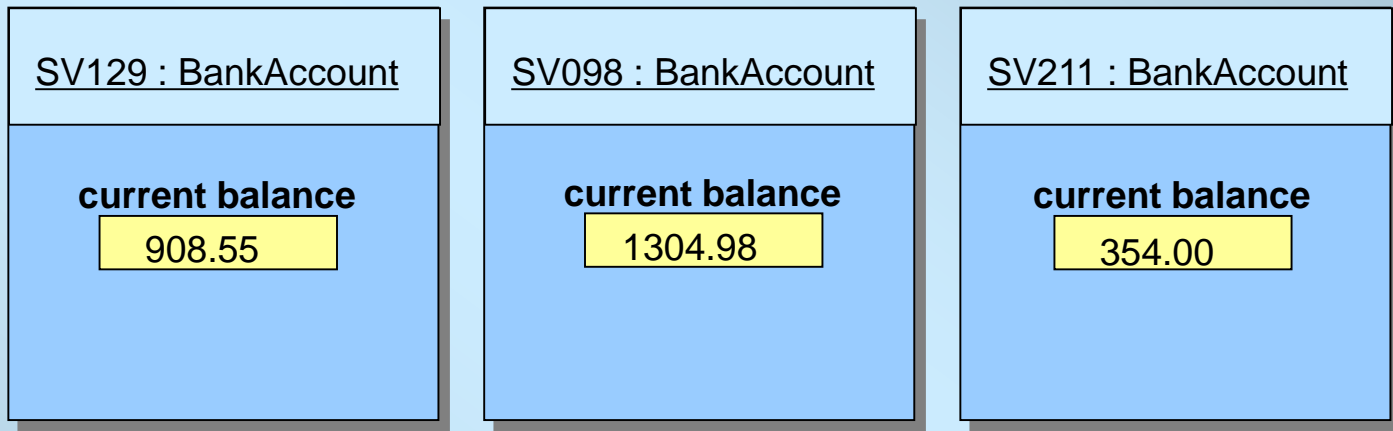
Calling a Class Method



Class and Instance Data Values

- An object is comprised of data values and methods.
- An *instance data value* is used to maintain information specific to individual instances. For example, each BankAccount object maintains its balance.
- A *class data value* is used to maintain information shared by all instances or aggregate information about the instances.
- For example, minimum balance is the information shared by all Account objects, whereas the average balance of all BankAccount objects is an aggregate information.

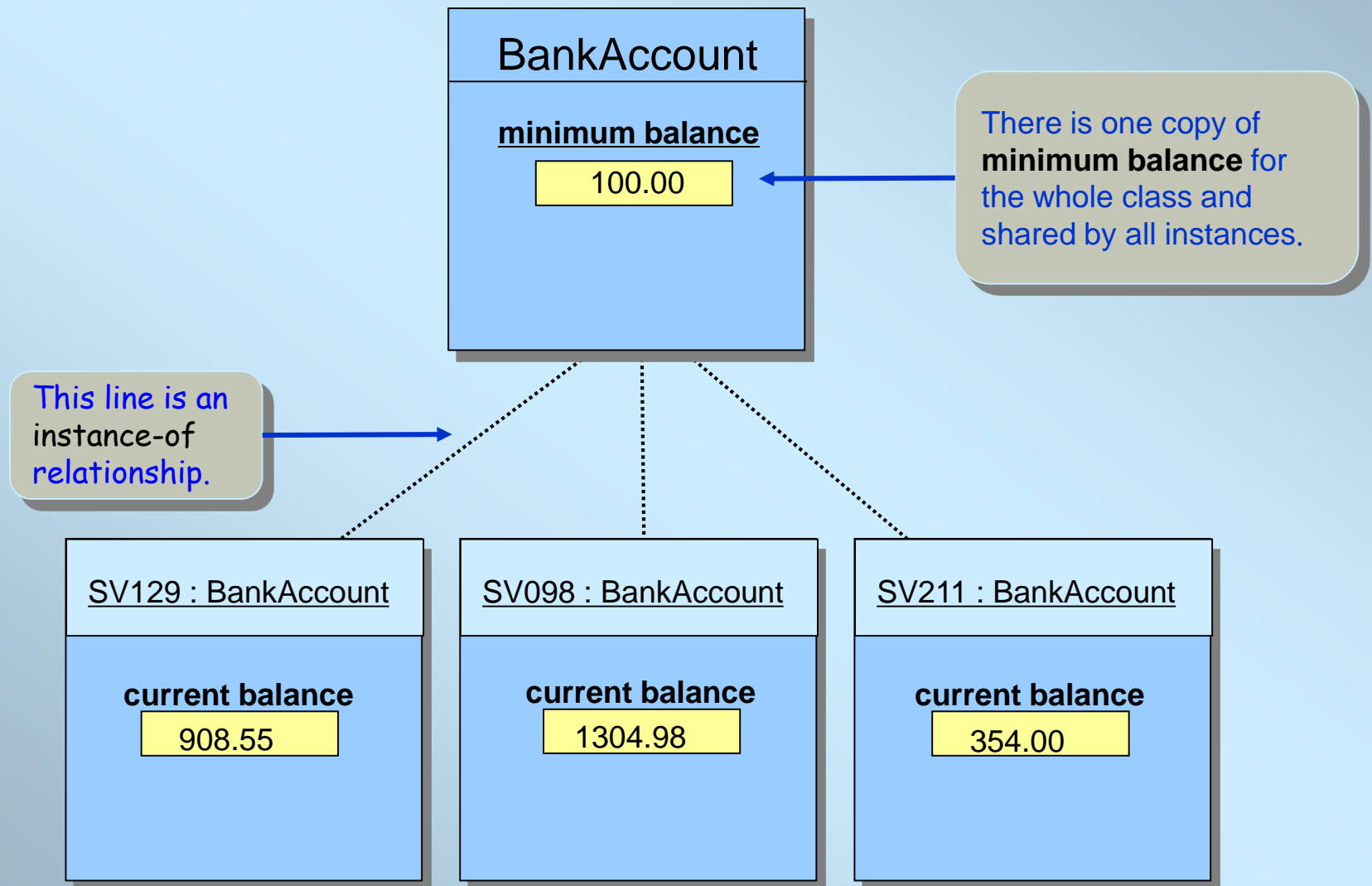
Sample Instance Data Value



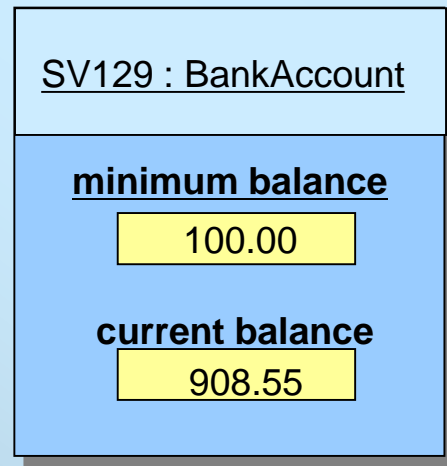
All three BankAccount objects possess the same instance data value current balance.

The actual dollar amounts are, of course, different.

Sample Class Data Value



Object Icon with Class Data Value



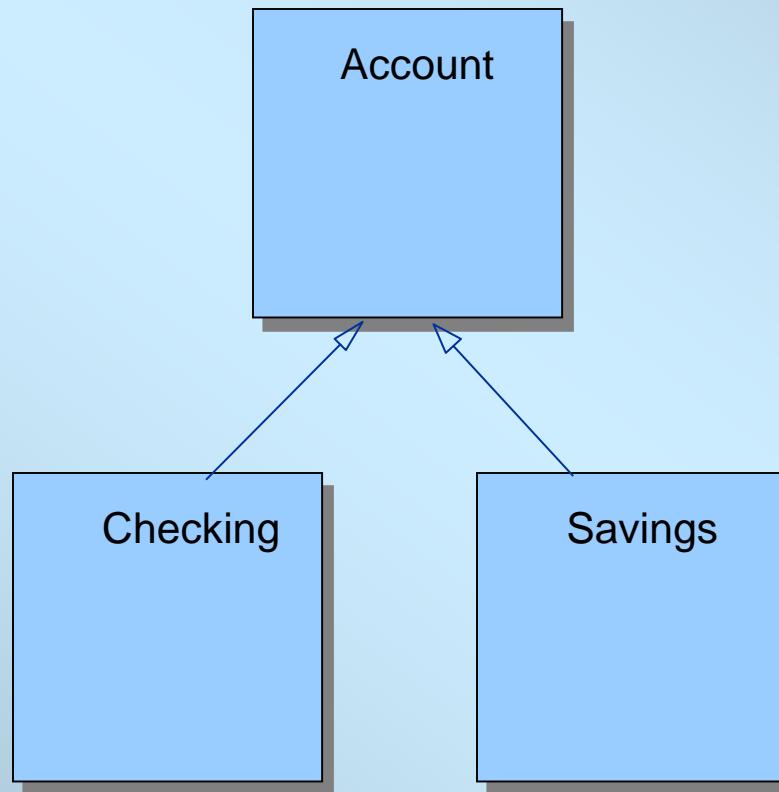
When the class icon is not shown, we include the class data value in the object icon itself.

Inheritance

- *Inheritance* is a mechanism in OOP to design two or more entities that are different but share many common features.
 - Features common to all classes are defined in the *superclass*.
 - The classes that inherit common features from the superclass are called *subclasses*.
 - We also call the superclass an *ancestor* and the subclass a *descendant*.

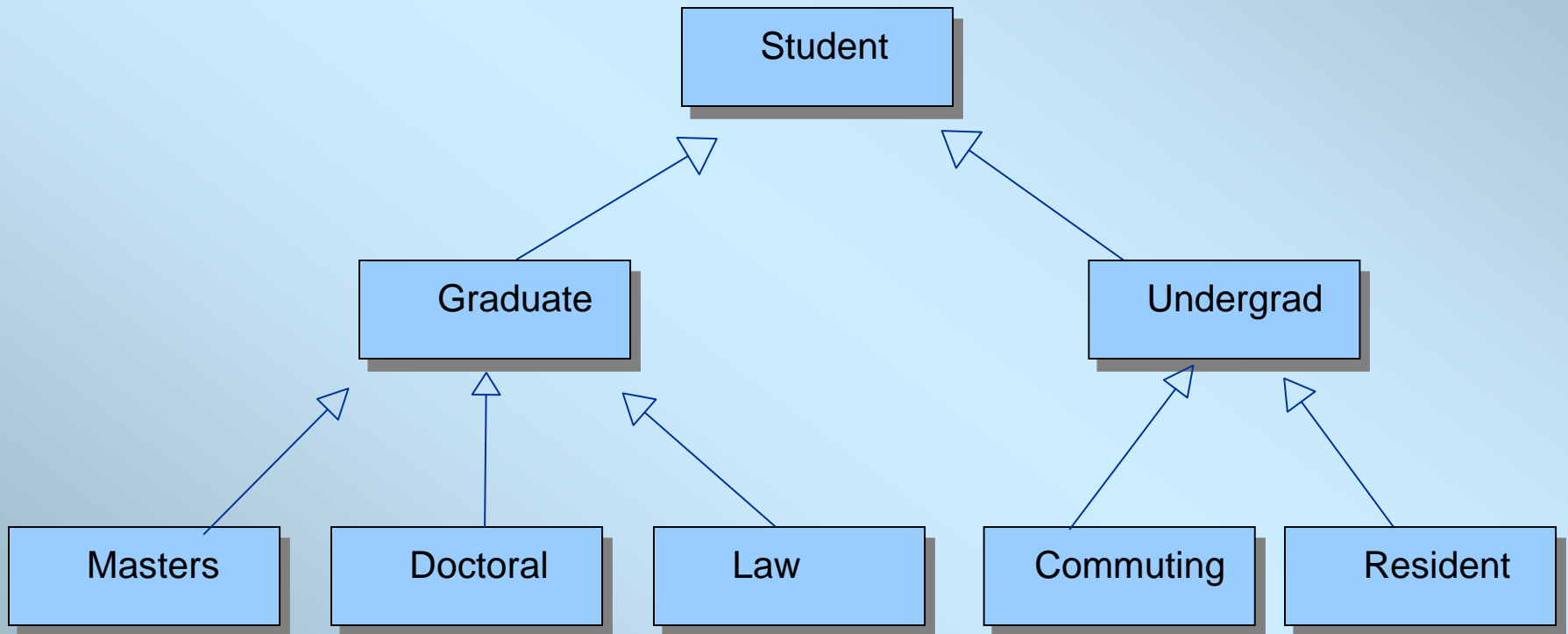
A Sample Inheritance

- Here are the superclass **Account** and its subclasses **Savings** and **Checking**.



Inheritance Hierarchy

- An example of inheritance hierarchy among different types of students.



Software Engineering

- Much like building a skyscraper, we need a disciplined approach in developing complex software applications.
- *Software engineering* is the application of a systematic and disciplined approach to the development, testing, and maintenance of a program.
- In this class, we will learn how to apply sound software engineering principles when we develop sample programs.

Software Life Cycle

- The sequence of stages from conception to operation of a program is called *software life cycle*.
- Phases / stages are:
 - Requirements Analysis/ Engineering
 - Systems Analysis
 - Systems Design
 - Development/ Implementation & Debugging
 - Testing
 - Operation and Maintenance