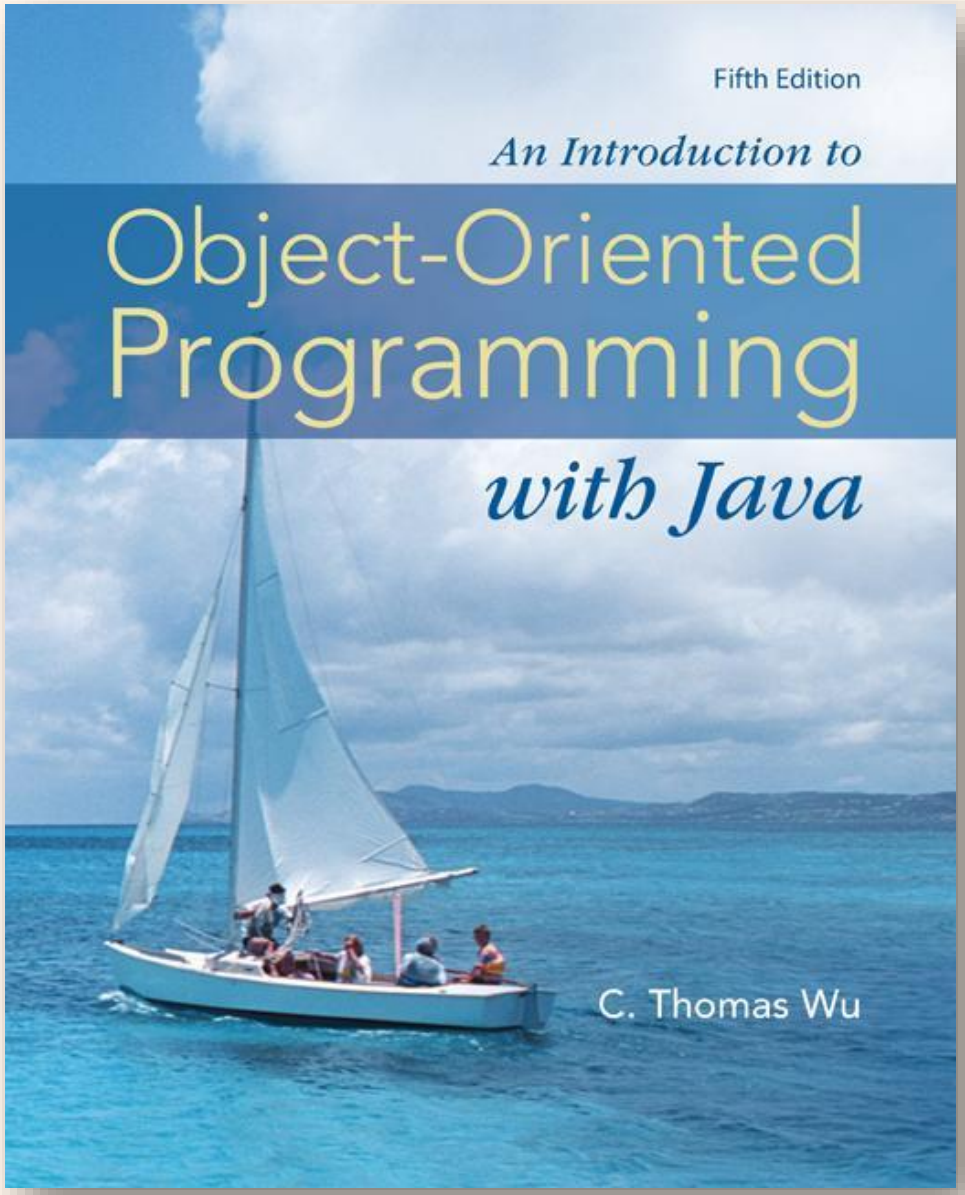


Chapter 1

Introduction to Object-Oriented Programming

Animated Version

Wholeness of the Lesson

In the OO paradigm of programming, execution of a program involves objects interacting with objects. Each object has a type, which is embodied in a Java *class.* The intelligence underlying the functioning of any object in a Java program resides in its underlying class, which is the silent basis for the dynamic behavior of the objects. Likewise, pure consciousness is the silent level of intelligence that underlies all expressions of intelligence in the form of thoughts and actions in life.

Objectives

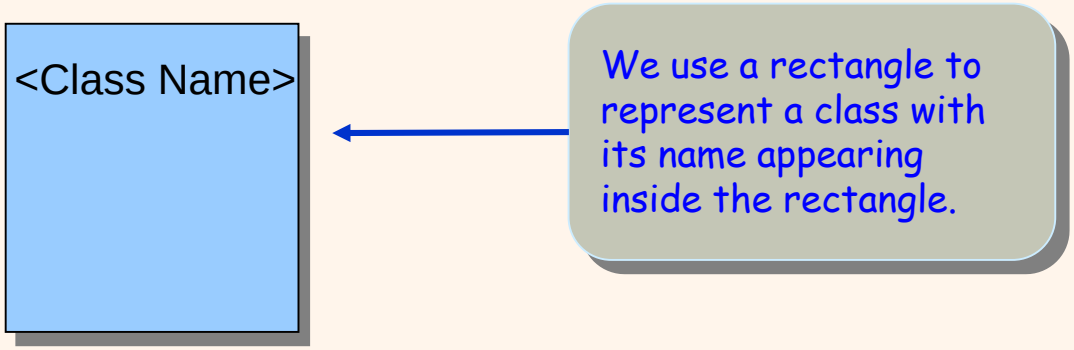
After you have read and studied this chapter, you should be able to

- Name the basic components of object-oriented programming
- Differentiate classes and objects.
- Differentiate class and instance methods.
- Differentiate class and instance data values.
- Draw uml diagrams
- Describe significance of inheritance in object-oriented programs

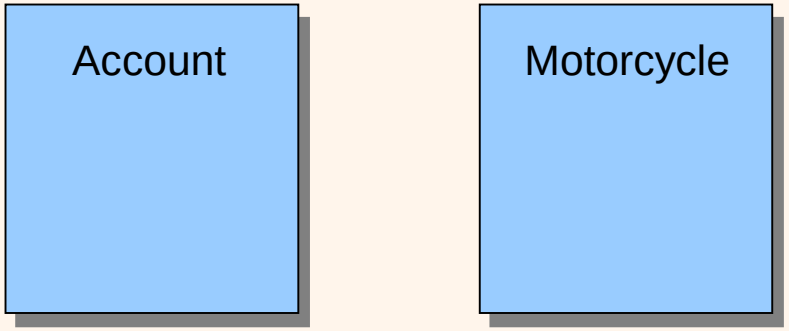
Classes and Objects

- 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

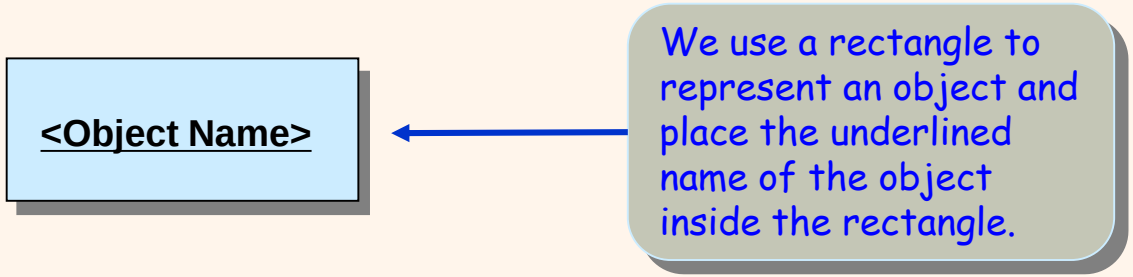


Example:

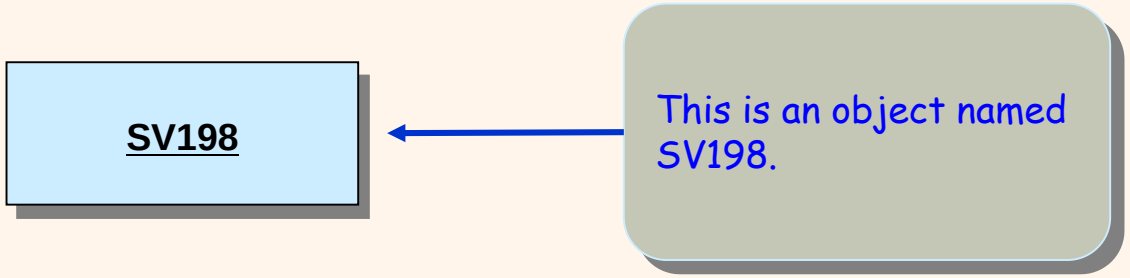


The notation we used here is based on the industry standard notation called *UML*, which stands for Unified Modeling Language.

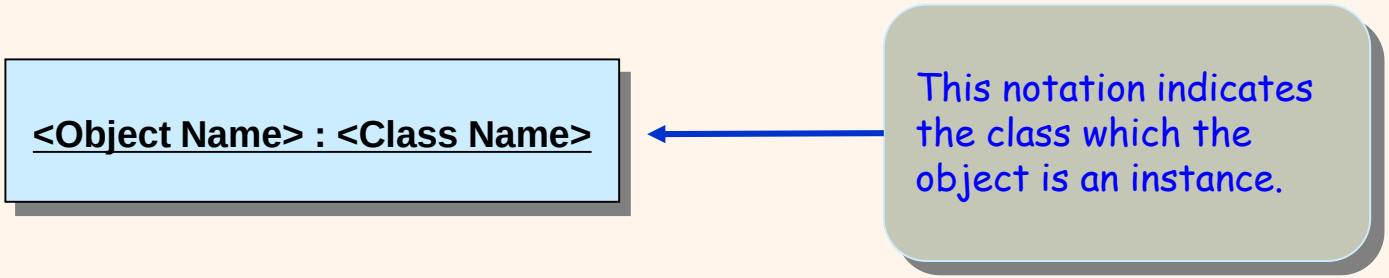
Graphical Representation of an Object



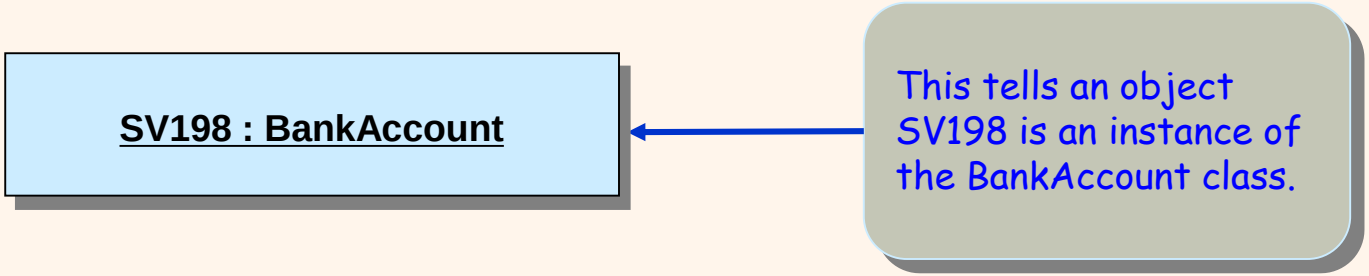
Example:



An Object with the Class Name



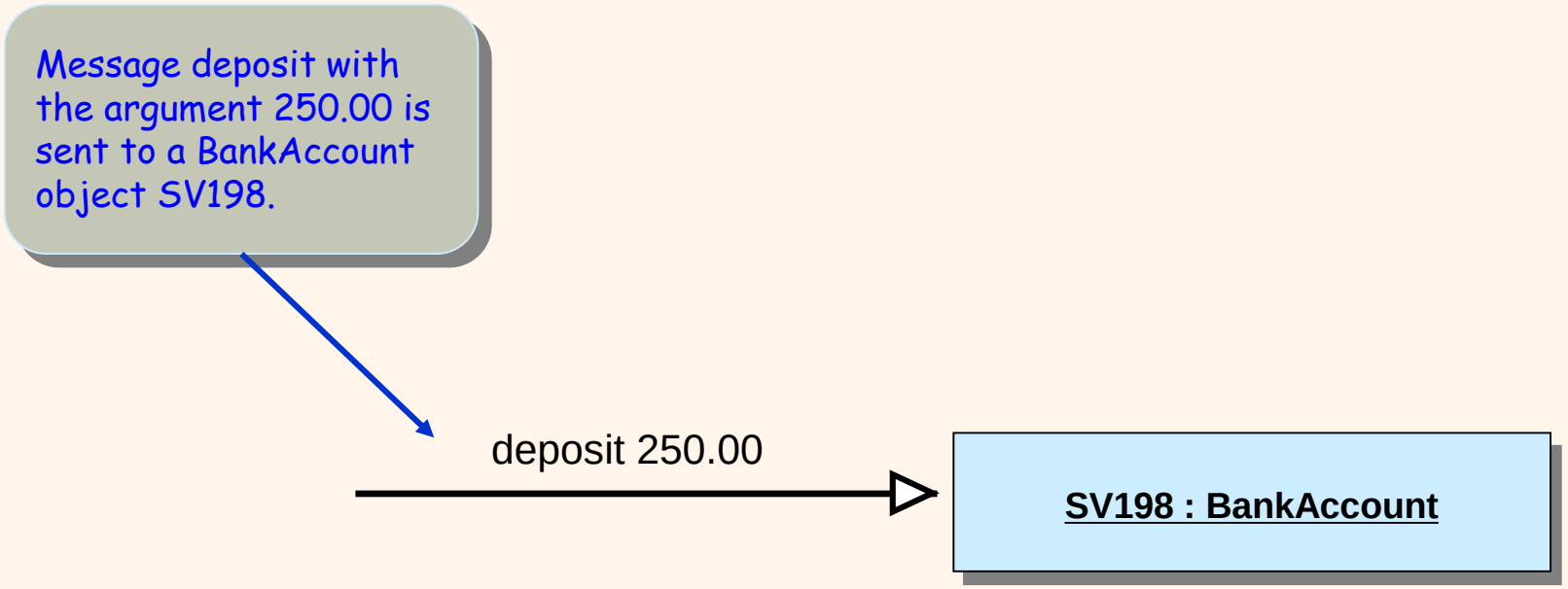
Example:



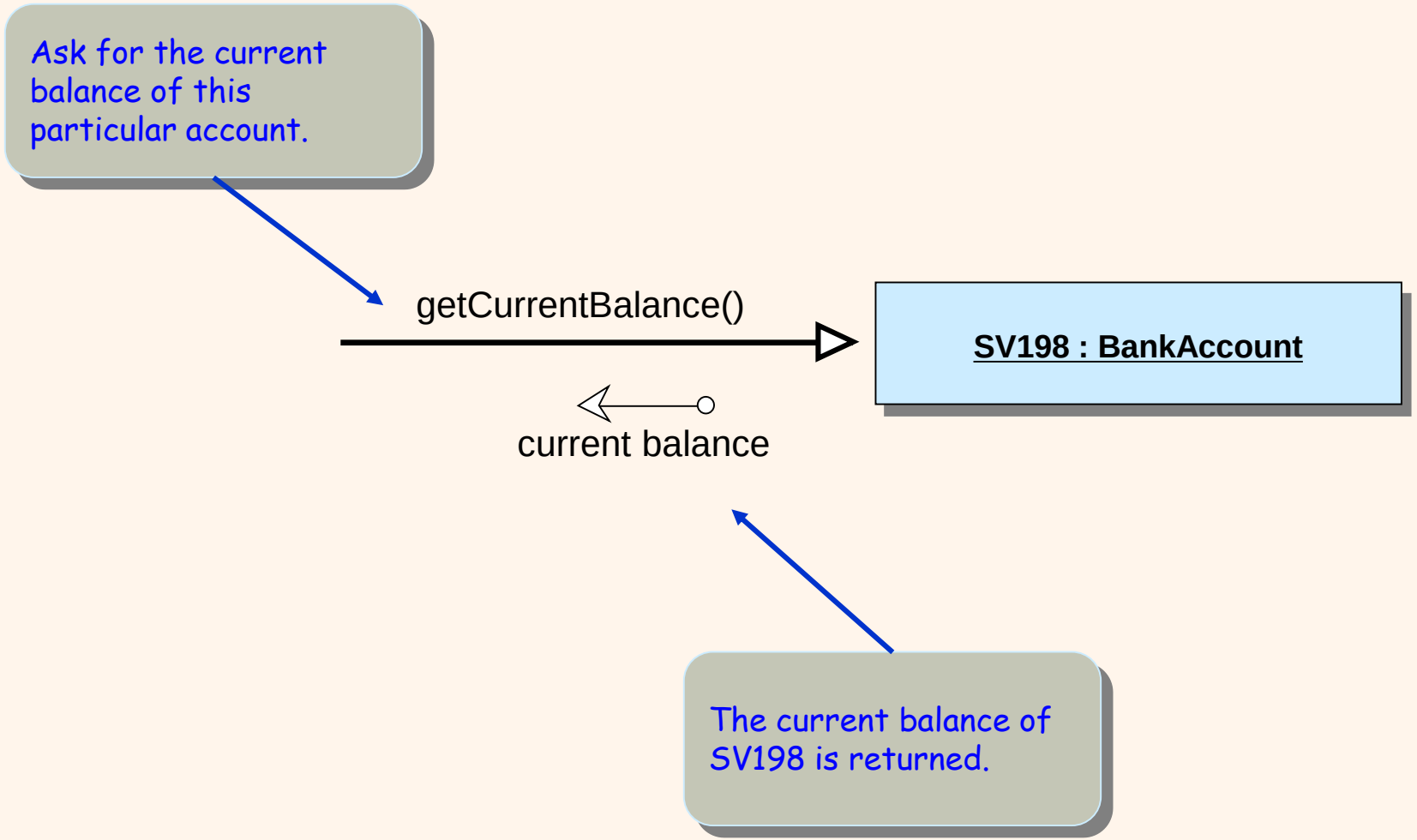
Messages and Methods

- 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.

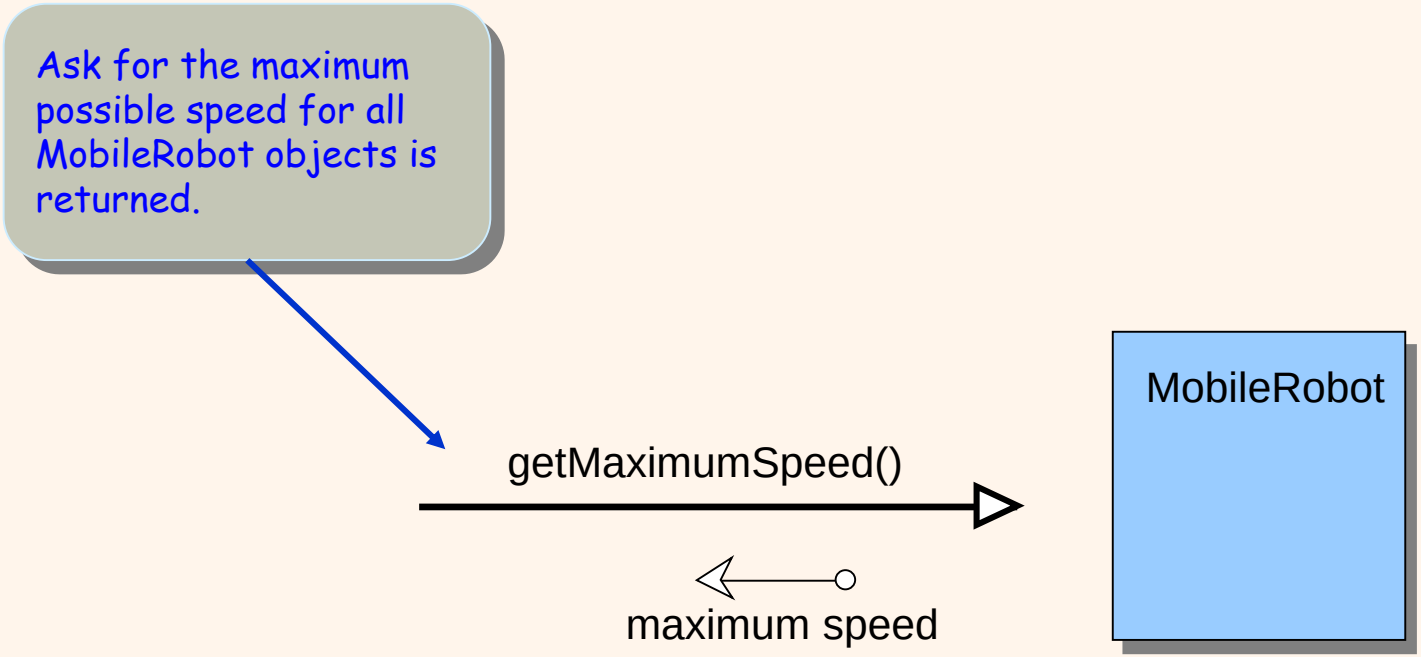
Sending a Message



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.

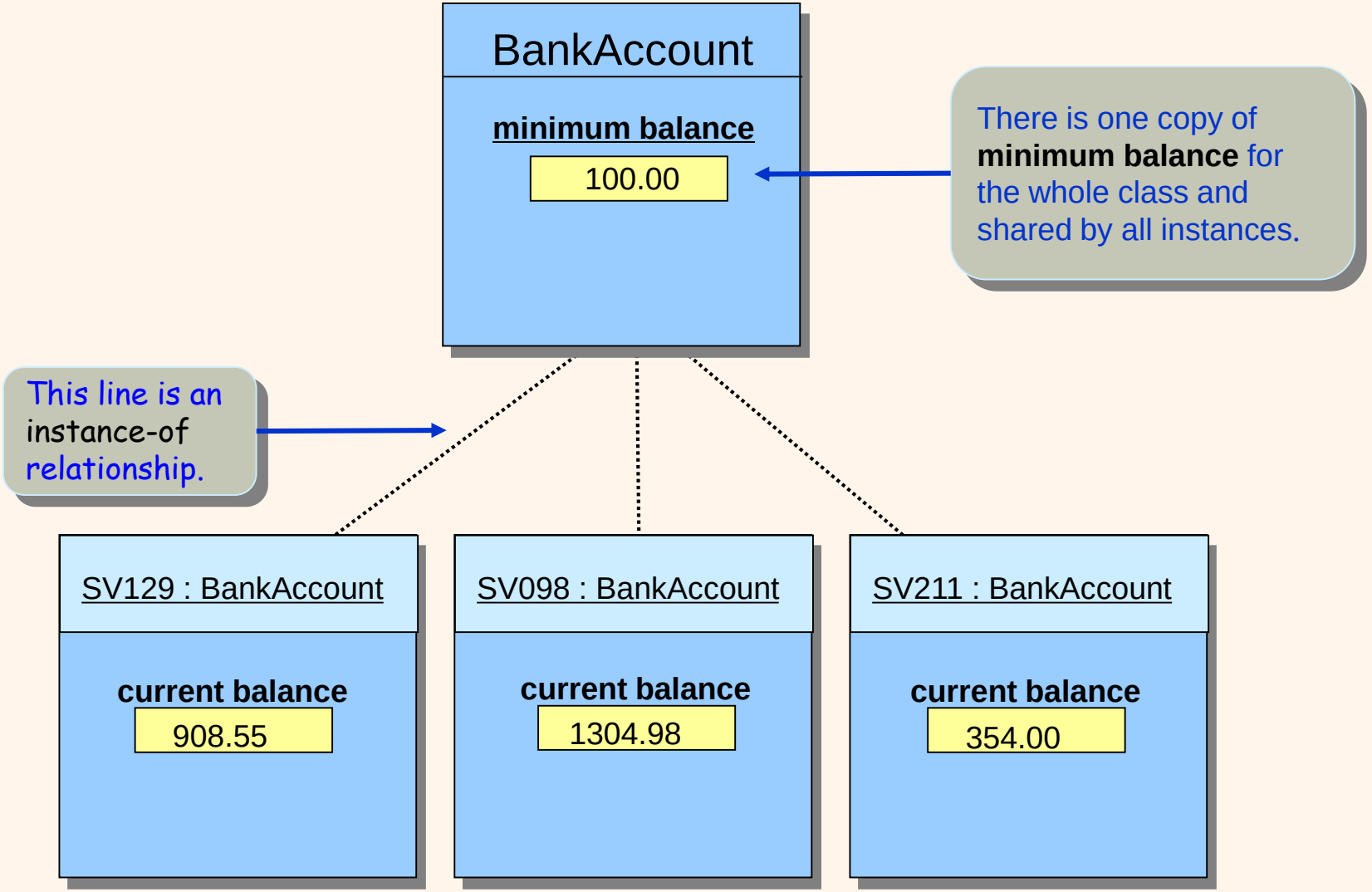
Sample Instance Data Value

SV129 : BankAccount	SV098 : BankAccount	SV211 : BankAccount
current balance 908.55	**current balance** 1304.98	**current balance** 354.00

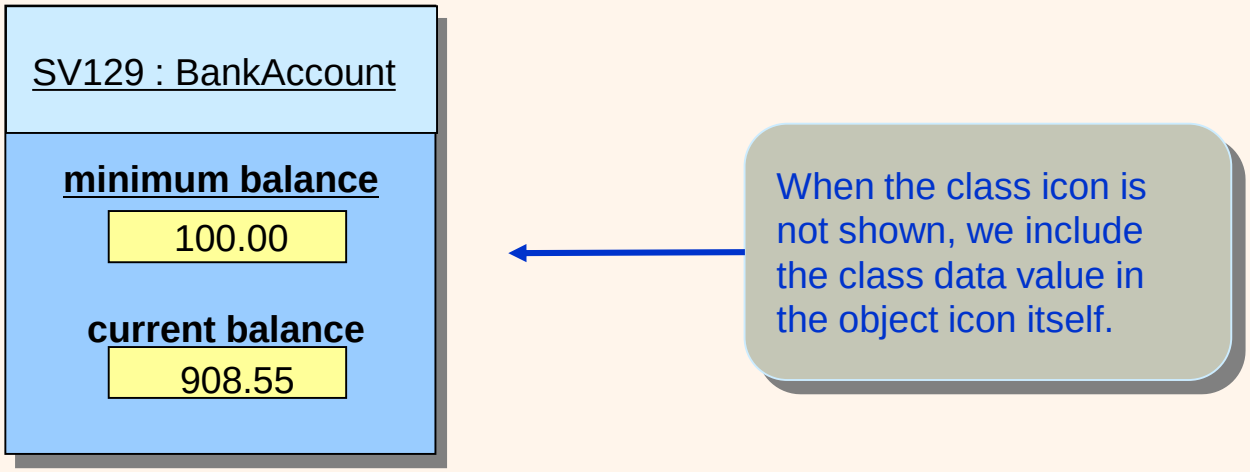
All three BankAccount objects possess the same instance data value current balance.

The actual dollar amounts are, of course, different.

Sample Class Data Value



Object Icon with Class Data Value



Main Point

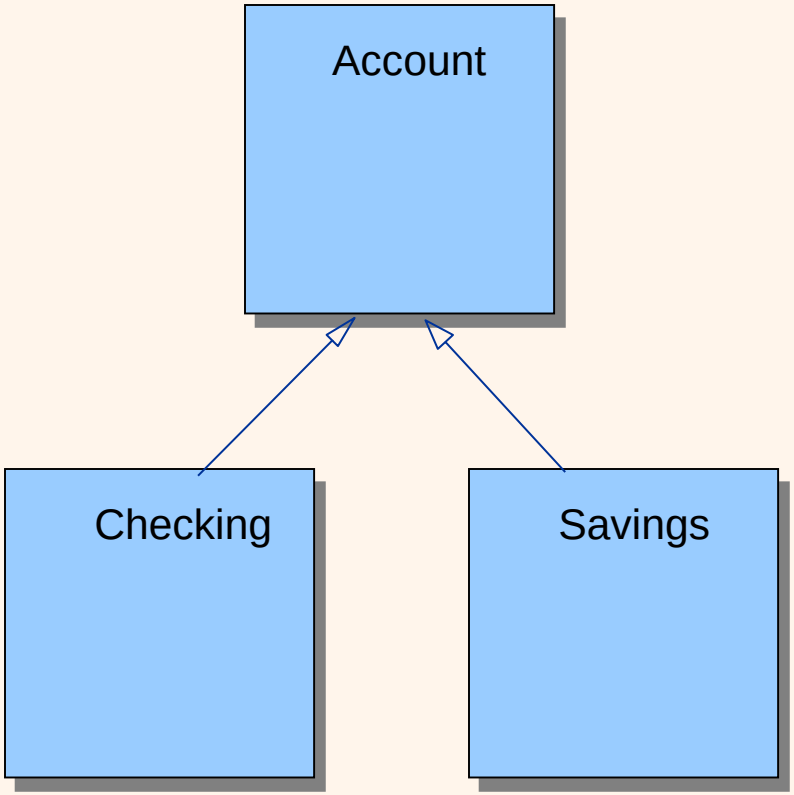
- Static fields and methods are fields and methods whose lifetime persists throughout execution of the application, and when used with the public keyword, are globally accessible. The notion of "static" parallels the recognition that there is a field in life that is globally available and is always located in the same place in “memory”: namely, pure consciousness.

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*.

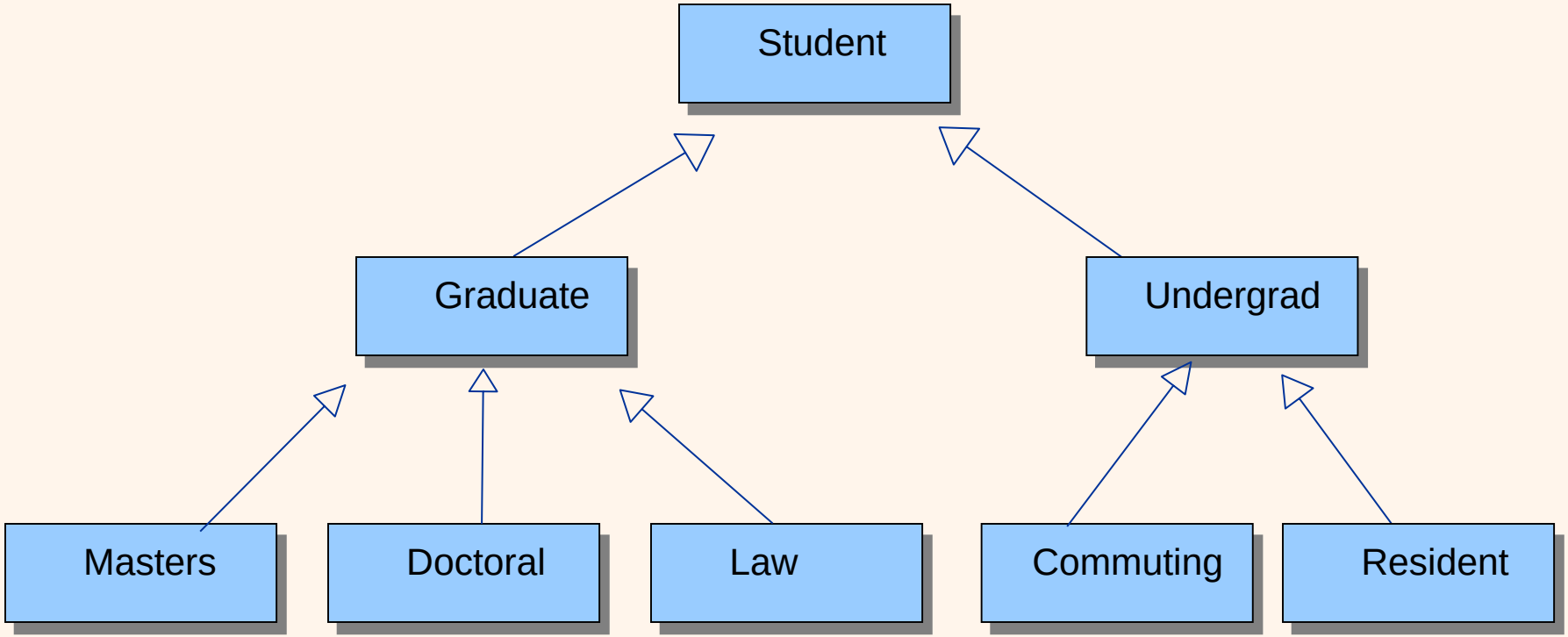
A Sample Inheritance

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



Inheritance Hierarchy

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



Cont...lecture 2