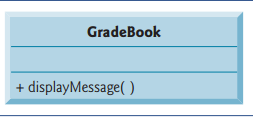
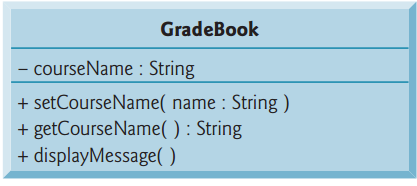
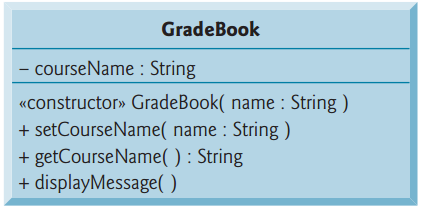
* **In the UML, each class is modeled in a class diagram as a rectangle with three compartments. The top compartment contains the name of the class centered horizontally in boldface type. The middle compartment contains the class’s attributes, which correspond to instance variables. The bottom compartment contains the** **class’s operations, which correspond to methods in Java**



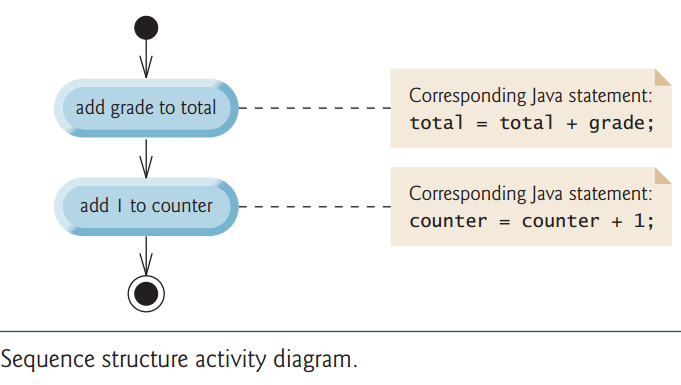
* **Instance variable courseName is private in Java, so the class diagram lists a minus sign (–) access modifier in front of the corresponding attribute’s name. Class GradeBook contains three public methods, so the class diagram lists three operations in the third compartment. Recall that the plus sign (+) before each operation name indicates that the operation is public. Operation setCourseName has a String parameter called name. The UML indicates the return type of an operation by placing a colon and the return type after the parentheses following the operation name.**

****

* **the UML requires that the word “constructor” be placed between guillemets (« and »)before the constructor’s name. It’s customary to list constructors before other operations in the third compartment.**

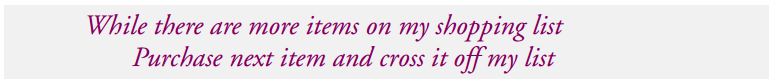
****

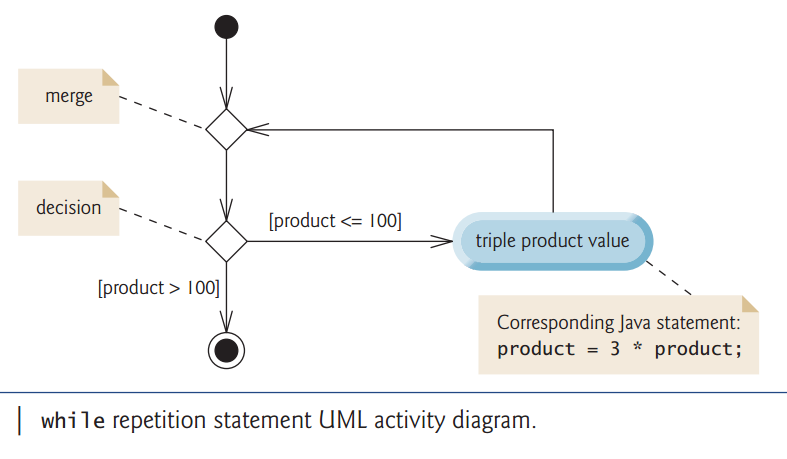
* **Pseudocode is an informal language that helps you develop algorithms without having to worry about the strict details of Java language syntax**
* **A UML activity diagram models the workflow(also called the activity) of a portion of a software system**

****

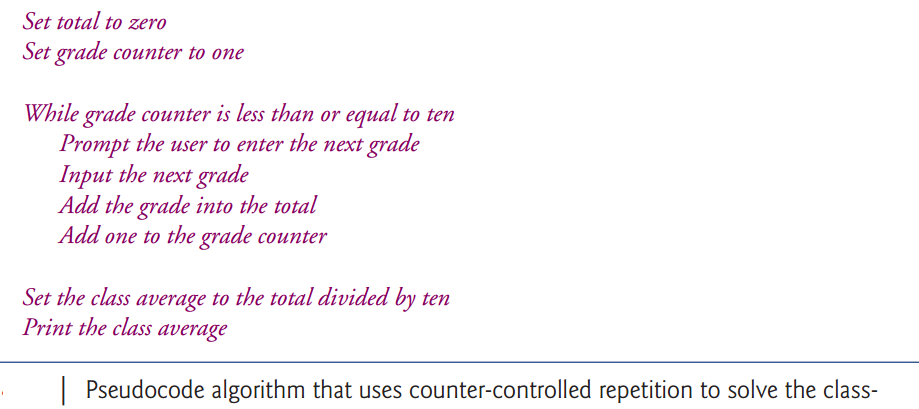
* **The solid circle at the top of the activity diagram represents the initial state—the solid circle surrounded by a hollow circle that appears at the bottom of the diagram represents the final state—the end of the workflow after the program performs its actions.**
* **Pseudocode Example**

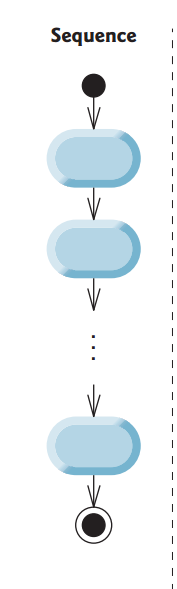
****

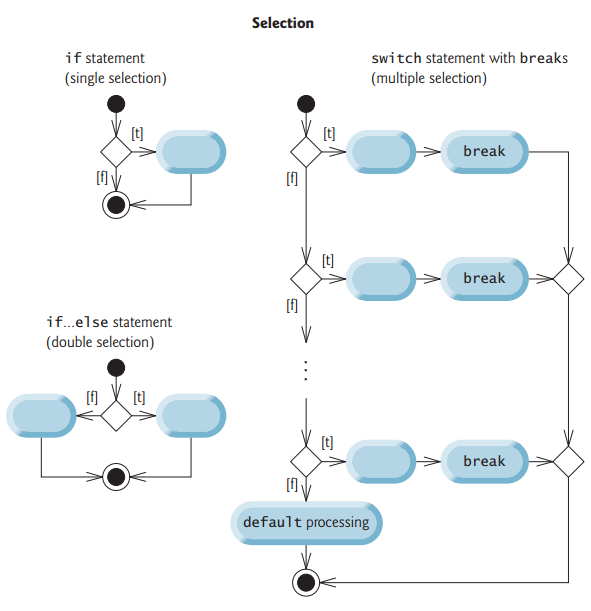
****

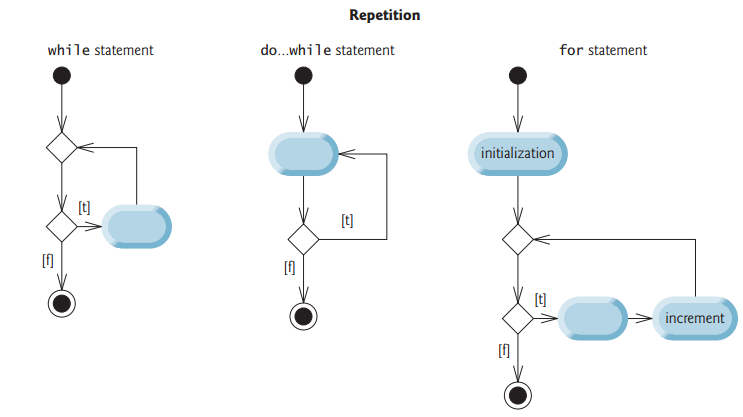
****

* **Experience has shown that the most difficult part of solving a problem on a computer is developing the algorithm for the solution. Once a correct algorithm has been specified, producing a working Java program from the algorithm is usually straightforward.**

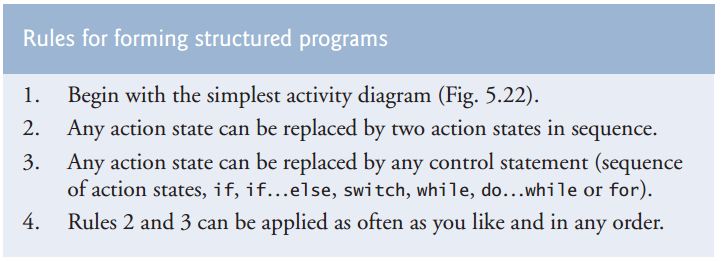
****

****

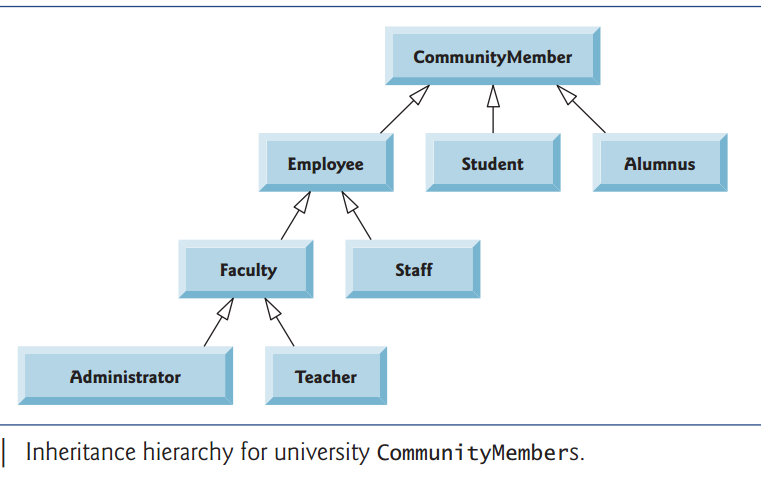
****

****

* **Structured programming promotes simplicity. Only three forms of control are needed to implement an algorithm:**
* **Sequence**
* **Selection**
* **Repetition**

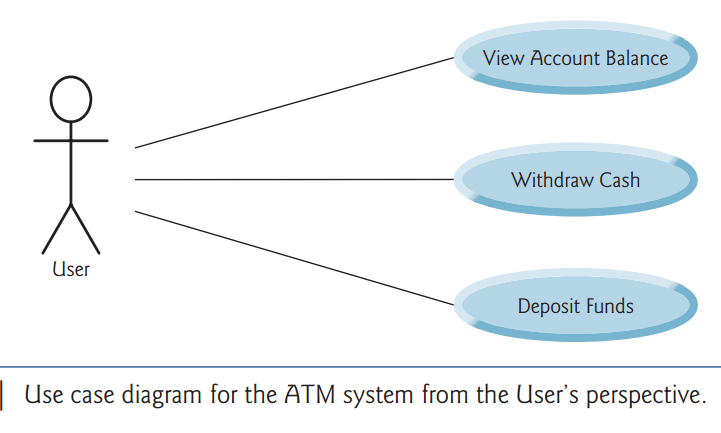
****

* **Inheritance relationships form treelike hierarchical structures. A superclass exists in a hierarchical relationship with its subclasses. Let’s develop a sample class hierarchy (Fig. 9.2),also called an inheritance hierarchy**

****

* **Case Diagram:**

**We now introduce the first of several UML diagrams in the case study. We create a use case diagram to model the interactions between a system’s clients (in this case study, bank customers) and its use cases. The goal is to show the kinds of interactions users have with a system without providing the details—these are provided in other UML diagrams (which we present throughout this case study). Use case diagrams are often accompanied by informal text that gives more detail**

****

* **Steps to make the system**

**1- Use case diagrams: such as the one model the interactions between a system and its external entities (actors) in terms of use cases**

**2- Class diagrams: which model the classes**

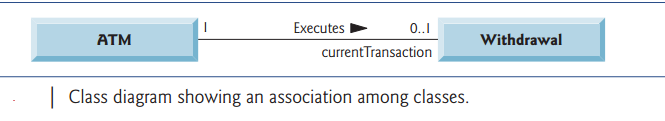
**3- State machine: diagrams, which model the ways in which an object changes state. An object’s state is indicated by the values of all it attributes at a given time. When an object changes state, it may behave differently in the system**

**4- Activity diagrams: which model an object’s activity—is workflow (sequence of events) during program execution. An activity diagram models the actions the object performs and specifies theorderin which it performs them**

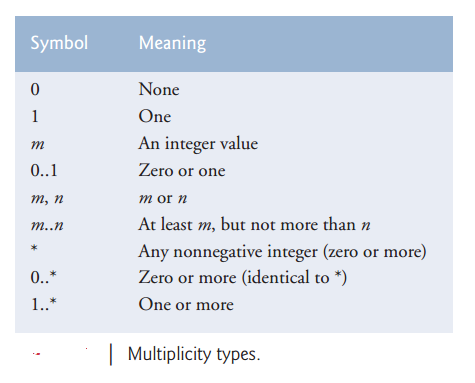
**5- Communication diagrams: (called collaboration diagrams in earlier versions of the UML) model the interactions among objects in a system, with an emphasis on what interactions occur.**

**6- Sequence diagrams also model the interactions among the objects in a system, but unlike communication diagrams, they emphasize when interactions occur.**

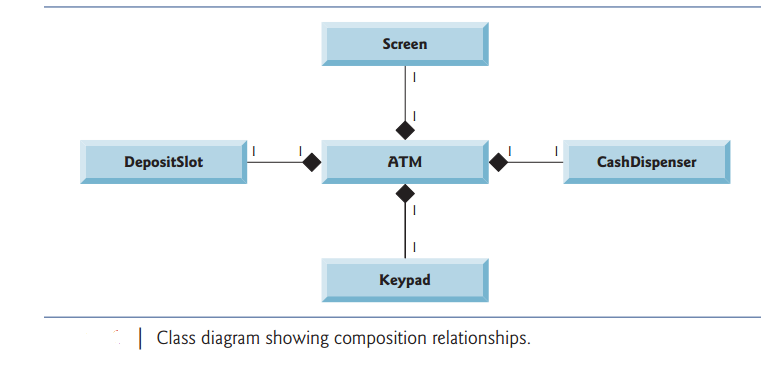
* **First, we review the requirements document and identify key nouns and noun phrases to help us identify classes that comprise the ATM system.**

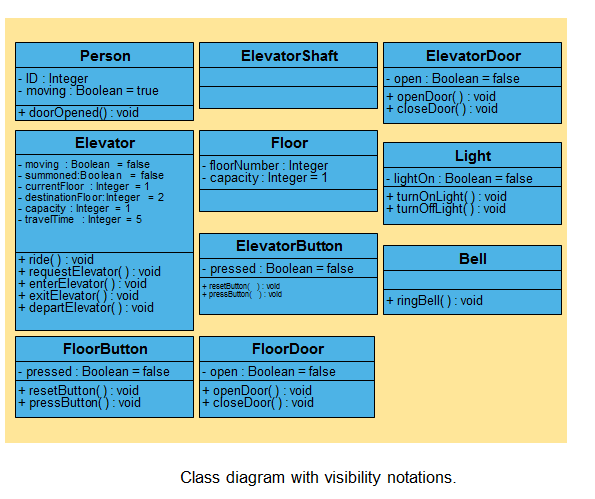
****

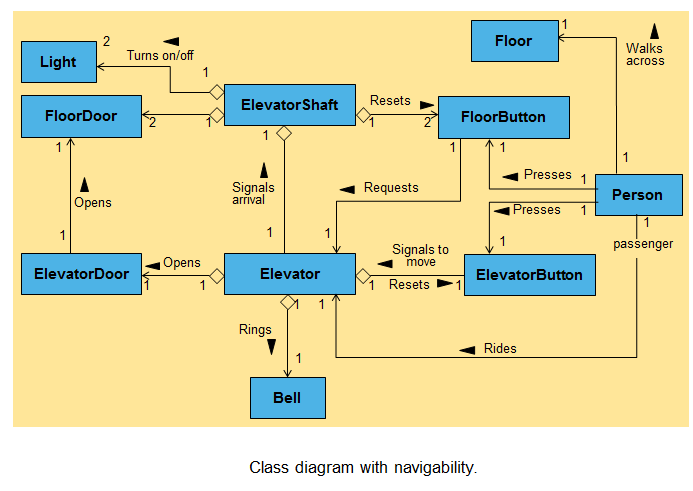
* **The numbers near each end of the line are multiplicity values, which indicate how many objects of each class participate in the association. In this case, following the line from left to right reveals that, at any given moment, one ATM object participates in an association with either zero or one Withdrawal objects—zero if the current user is not currently performing a transaction or has requested a different type of transaction, and one if the user has requested a withdrawal**

****

* **The solid diamonds attached to the ATM class’s association lines indicate that ATM has a composition relationship with classes Screen, Keypad, Cash Dispenser and Deposit Slot. Composition implies a whole/part relationship.**

****

****

****

* **We generate “skeleton code” with our design**

**Four steps:**

1. **Use name in first compartment to declare public class**

**Empty constructor**

1. **Use attributes in second compartment to declare instance variables**
2. **Use associations in class diagram (Fig. 3.19) to declare object references**
3. **Use operations in third compartment to declare methods**