Unified Modeling Language (UML) Part 2

Dr. Krishnendu Guha

Assistant Professor
(On Contract)

National Institute of Technology (NIT), Jamshedpur

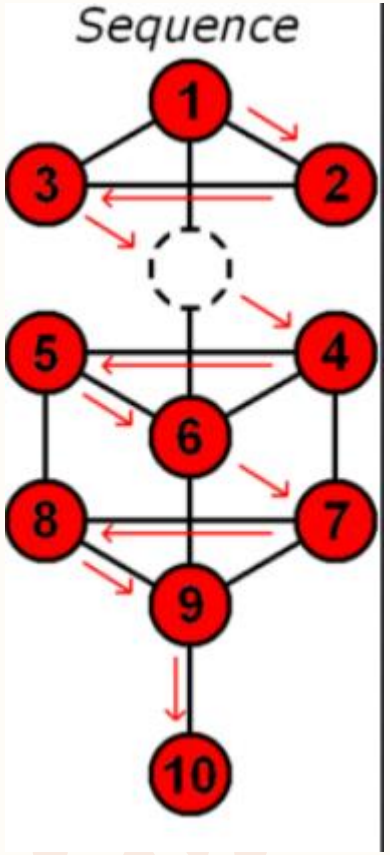
Email: krishnendu.ca@nitjsr.ac.in

Interaction Diagram

- An interaction diagram is either a sequence diagram or a communication diagram
- both of which show essentially the same information.
- These diagrams, along with class diagrams, are used in a use case realization, which is a way to achieve or accomplish a use case

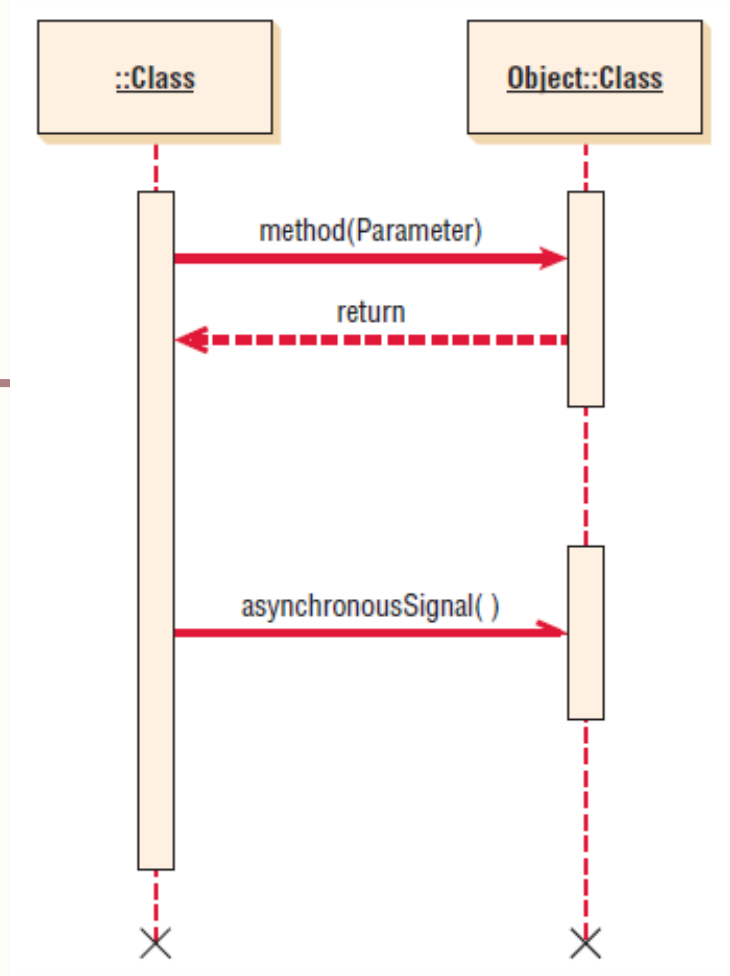
Sequence Diagrams

- Sequence diagrams can illustrate a succession of interactions between classes or object instances over time.
- Sequence diagrams are often used to illustrate the processing described in use case scenarios.
- In practice, sequence diagrams are derived from use case analysis and are used in systems design to derive the interactions, relationships, and methods of the objects in the system.
- Sequence diagrams are used to show the overall pattern of the activities or interactions in a use case.
- Each use case scenario may create one sequence diagram, although sequence diagrams are not always created for minor scenarios.
- Sequence diagrams can be used to translate the use case scenario into a visual tool for systems analysis.
- The initial sequence diagram used in systems analysis shows the actors and classes in the system and the interactions between them for a specific process.
- You can use this version of the sequence diagram to verify processes with the business area experts who have assisted you in developing the system requirements.
- A sequence diagram emphasizes the time ordering (sequence) of messages.



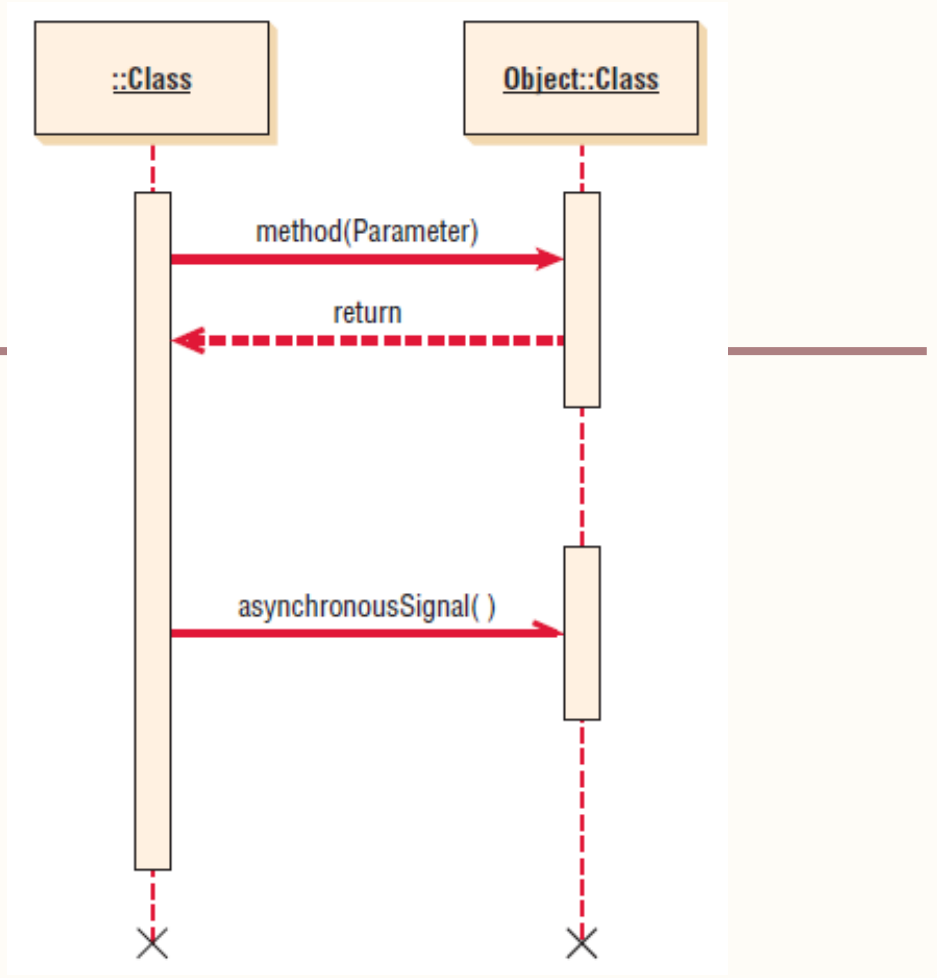
– Symbols used in Sequence Diagrams

- Actors and classes or object instances are shown in boxes along the top of the diagram.
- The leftmost object is the starting object and may be a person (for which a use case actor symbol is used), window, dialog box, or other user interface.
- Some of the interactions are physical only, such as signing a contract.
- The top rectangles use indicators in the name to indicate whether the rectangle represents an object, a class, or a class and object
- A vertical line represents the lifeline for the class or object, which corresponds to the time from when it is created through when it is destroyed.
- An X on the bottom of the lifeline represents when the object is destroyed.
- A lateral bar or vertical rectangle on the lifeline shows the focus of control when the object is busy doing things.

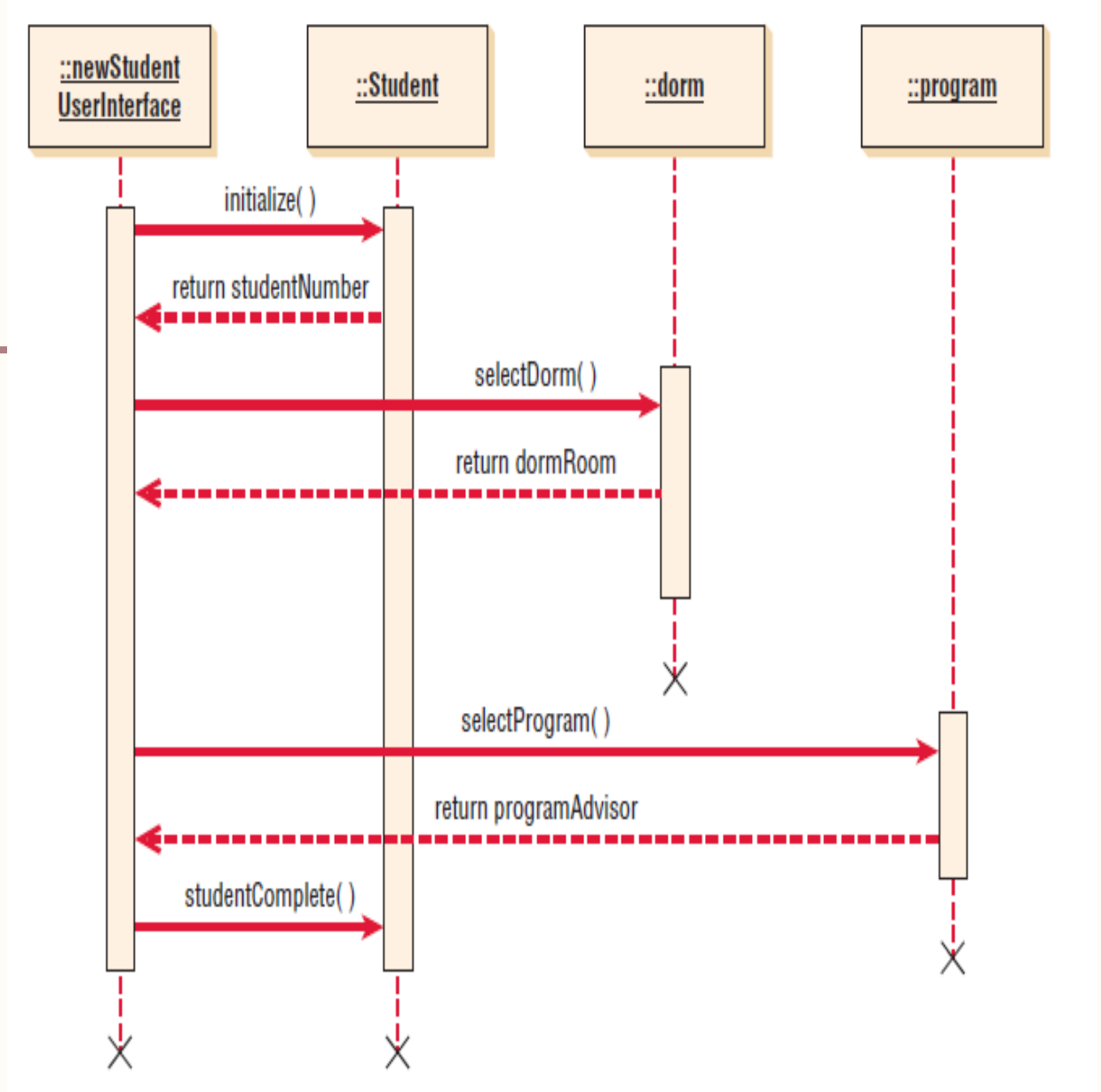


objectName:	A name with a colon after it represents an object.
:class	A colon with a name after it represents a class.
objectName:class	A name, followed by a colon and another name, represents an object in a class.

- Horizontal arrows show messages or signals that are sent between the classes.
- Messages belong to the receiving class. There are some variations in the message arrows.
- Solid arrowheads represent synchronous calls, which are the most common. These are used when the sending class waits for a response from the receiving class, and control is returned to the sending class when the class receiving the message finishes executing.
- Half (or open) arrowheads represent asynchronous calls, or those that are sent without an expectation of returning to the sending class. An example would be using a menu to run a program.
- A return is shown as an arrow, sometimes with a dashed line.
- Timing in the sequence diagram is displayed from top to bottom; the first interaction is drawn at the top of the diagram, and the interaction that occurs last is drawn at the bottom of the diagram.
- The interaction arrows begin at the bar of the actor or object that initiates the interaction, and they end pointing at the bar of the actor or object that receives the interaction request.
- The starting actor, class, or object is shown on the left.
- This may be the actor that initiates the activity or it may be a class representing the user interface.

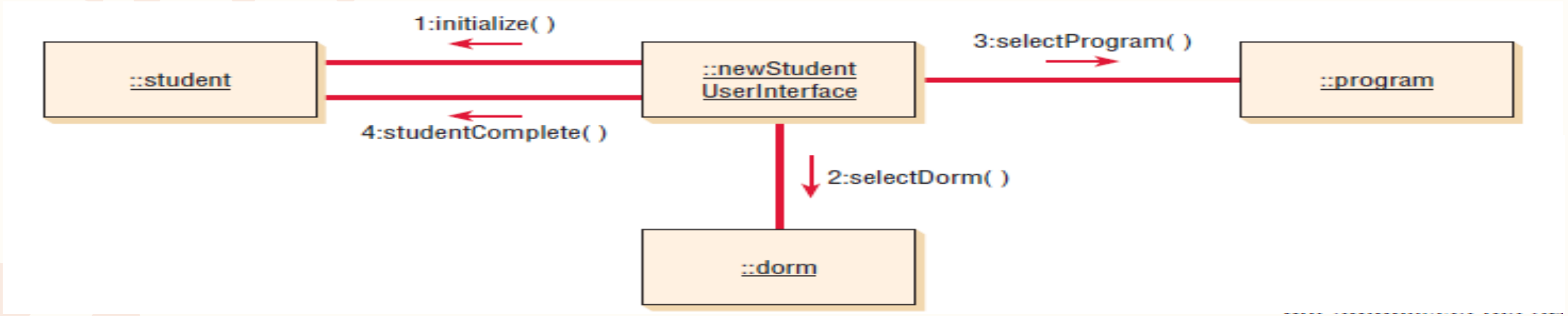


sequence diagram for a use case that admits a student to a university. On the left is the **newStudentUserInterface** class that is used to obtain student information. The **initialize()** message is sent to the **Student** class, which creates a new student record and returns the student number. To simplify the diagram, the parameters that are sent to the **Student** class have been omitted, but would include the student name, address, and so on. The next activity is to send a **selectDorm** message to the **Dorm** class. This message would include dorm selection information, such as a health dorm or other student requirements. The **Dorm** class returns the dorm name and room number. The third activity is to send a **selectProgram** message to the **Program** class, including the program name and other course of study information. The program advisor name is returned to the **newStudentUserInterface** class. A **studentComplete** message is sent to the **Student** class with the dorm, advisor name, and other information



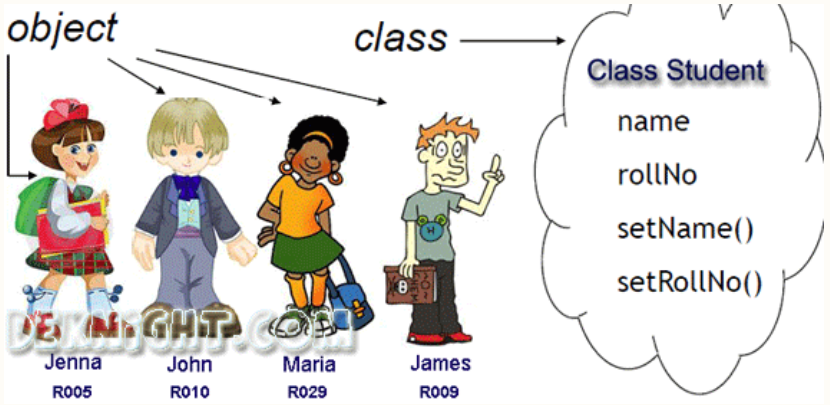
Communications Diagram

- Communication diagrams were introduced in UML 2.0.
- Their original name in UML 1.x was collaboration diagrams.
- A communication diagram emphasizes the organization of objects, whereas a sequence diagram emphasizes the time ordering of messages.
- A communication diagram will show a path to indicate how one object is linked to another.
- Communication diagrams describe the interactions of two or more things in the system that perform a behavior that is more than any one of the things can do alone.
- For instance, a car can be broken down into several thousand individual parts. The parts are put together to form the major subsystems of the vehicle: the engine, the transmission, the brake system, and so forth.
- The individual parts of the car can be thought of as classes, because they have distinct attributes and functions.
- The individual parts of the engine form a collaboration, because they “communicate” with each other to make the engine run when the driver steps on the accelerator.



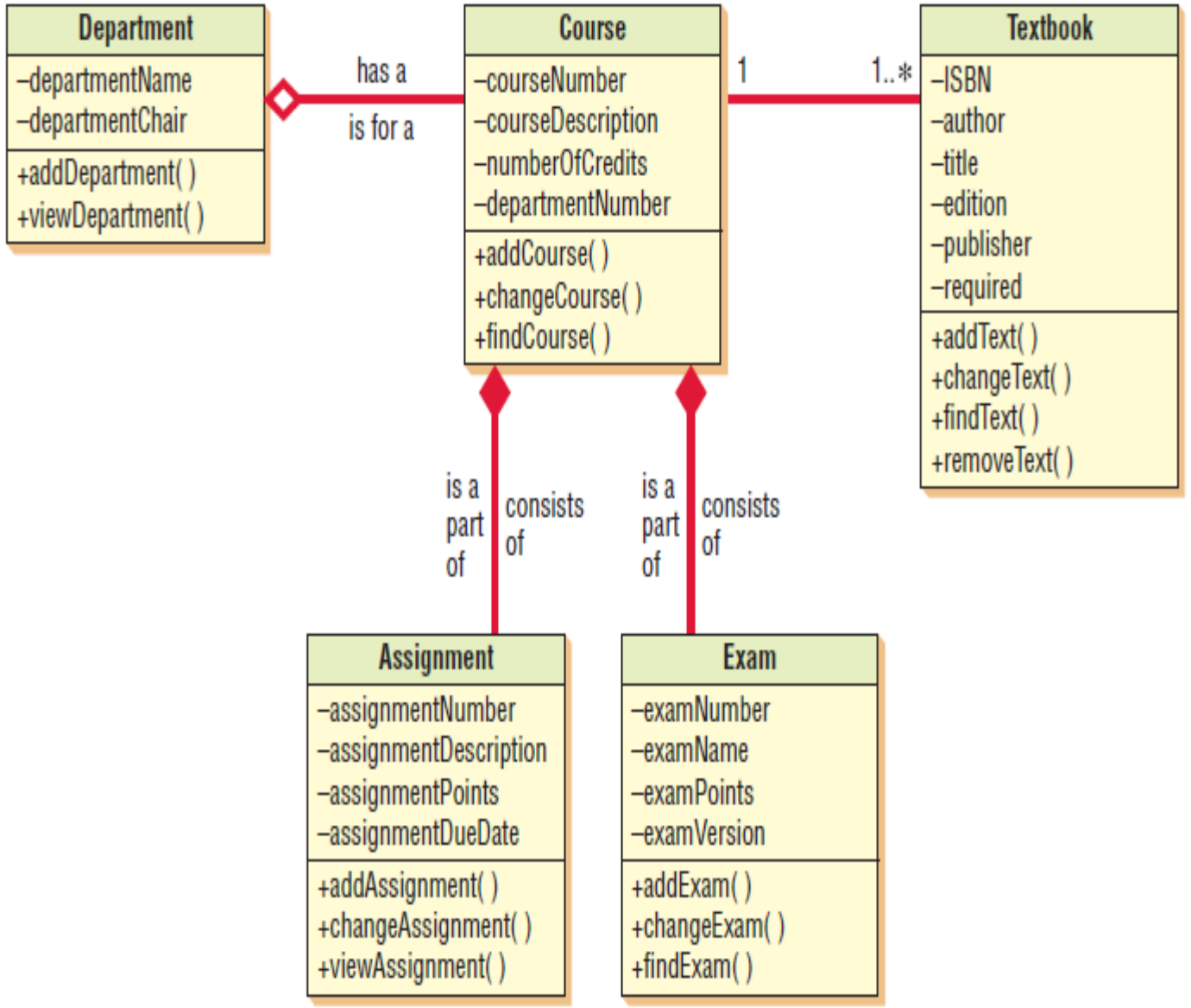
- A communication diagram is made up of three parts: objects (also called participants), the communication links, and the messages that can be passed along those links.
- In order to show time ordering, you must indicate a sequence number and describe the message.
- Some UML modeling software, such as IBM's Rational Rose, will automatically convert a sequence diagram to a communication diagram or a communication diagram to a sequence diagram with the click of a button.
- Each rectangle represents an object or a class.
- Connecting lines show the classes that need to collaborate or work with each other.
- The messages sent from one class to another are shown along connecting lines.
- Messages are numbered to show the time sequence.
- Return values may also be included and numbered to indicate when they are returned within the time sequence.

Class Diagrams



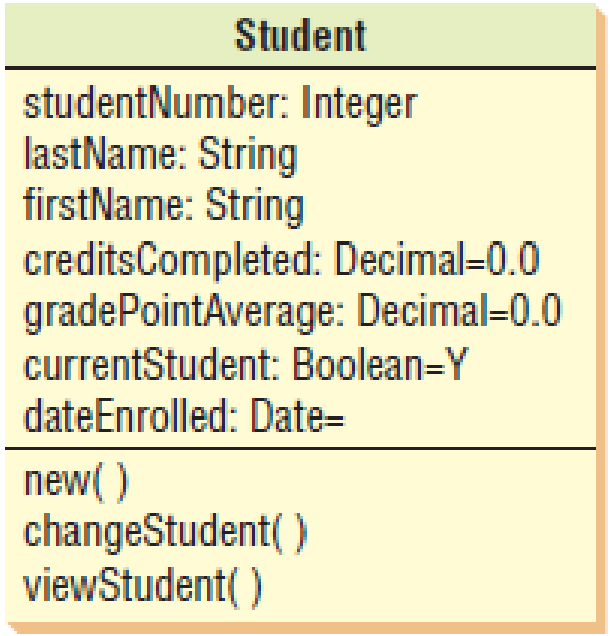
- Object-oriented methodologies work to discover classes, attributes, methods, and relationships between classes.
- Because programming occurs at the class level, defining classes is one of the most important object-oriented analysis tasks.
- Class diagrams show the static features of the system and do not represent any particular processing.
- A class diagram also shows the nature of the relationships between classes.
- Classes are represented by a rectangle on a class diagram.
- In the simplest format, the rectangle may include only the class name, but may also include the attributes and methods.
- Attributes are what the class knows about characteristics of the objects, and methods (also called operations) are what the class knows about how to do things.
- Methods are small sections of code that work with the attributes.
- The class diagram shows data storage requirements as well as processing requirements

- The filled-in diamonds show aggregation and the empty diamond shows a whole-part relationship.
- The attributes (or properties) are usually designated as private, or only available in the object.
- This is represented on a class diagram by a minus sign in front of the attribute name.
- Attributes may also be protected, indicated with a pound symbol (#). These attributes are hidden from all classes except immediate subclasses.
- Under rare circumstances, an attribute is public, meaning that it is visible to other objects outside its class.
- Making attributes private means that the attributes are only available to outside objects through the class methods, a technique called encapsulation, or information hiding.



A class diagram for course offerings

- The type of data (such as string, double, integer, or date) may be included on the class diagram.
- The most complete descriptions would include an equal sign () after the type of data followed by the initial value for the attribute
- If the attribute must take on one of a finite number of values, such as a student type with values of F for full-time, P for part-time, and N for nonmatriculating, these may be included in curly brackets separated by commas: **studentType:char{F,P,N}.**
- Information hiding means that objects'methods must be available to other classes,
- so methods are often public, meaning that they may be invoked from other classes.
- On a class diagram, public messages (and any public attributes) are shown with a plus sign () in front of them.
- Methods also have parentheses after them, indicating that data may be passed as parameters along with the message.
- The message parameters, as well as the type of data, may be included on the class diagram.
- There are two types of methods: standard and custom.
- Standard methods are basic things that all classes of objects know how to do, such as create a new object instance.
- Custom methods are designed for a specific class.



An extended **Student** class that shows the type of data and, in some cases, its initial value or default value


THANK YOU!