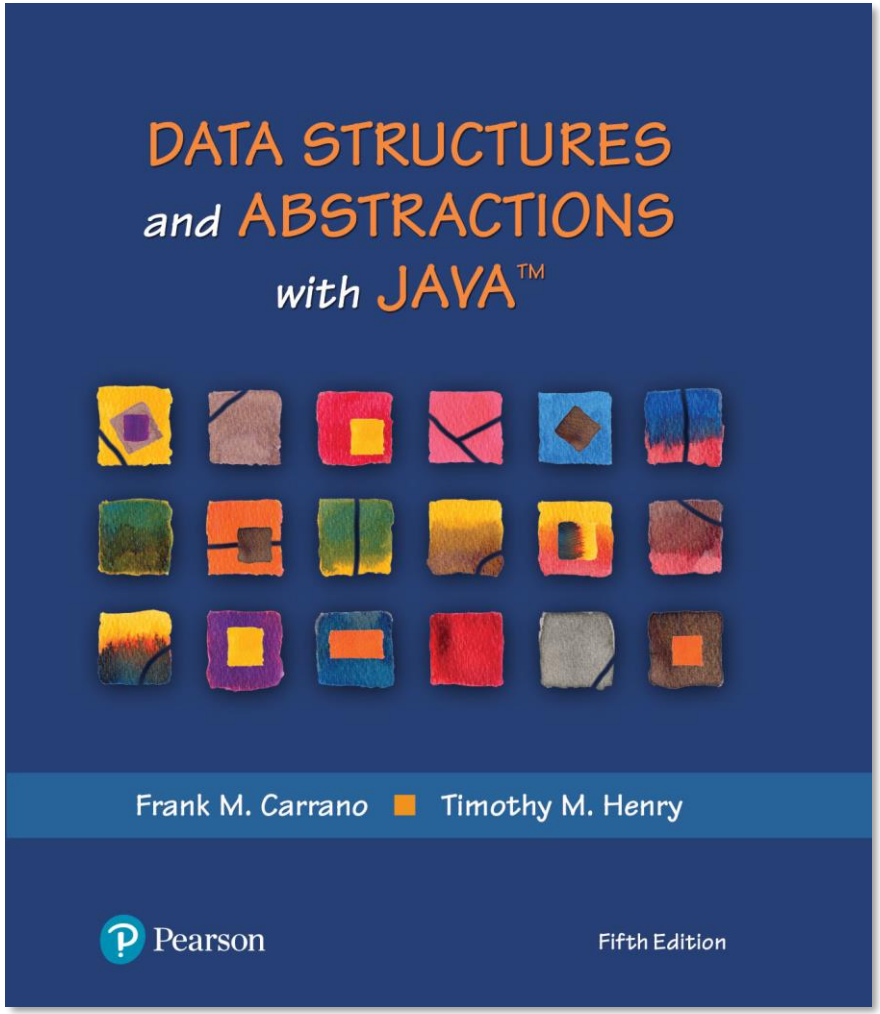
Data Structures and Abstractions with Java™

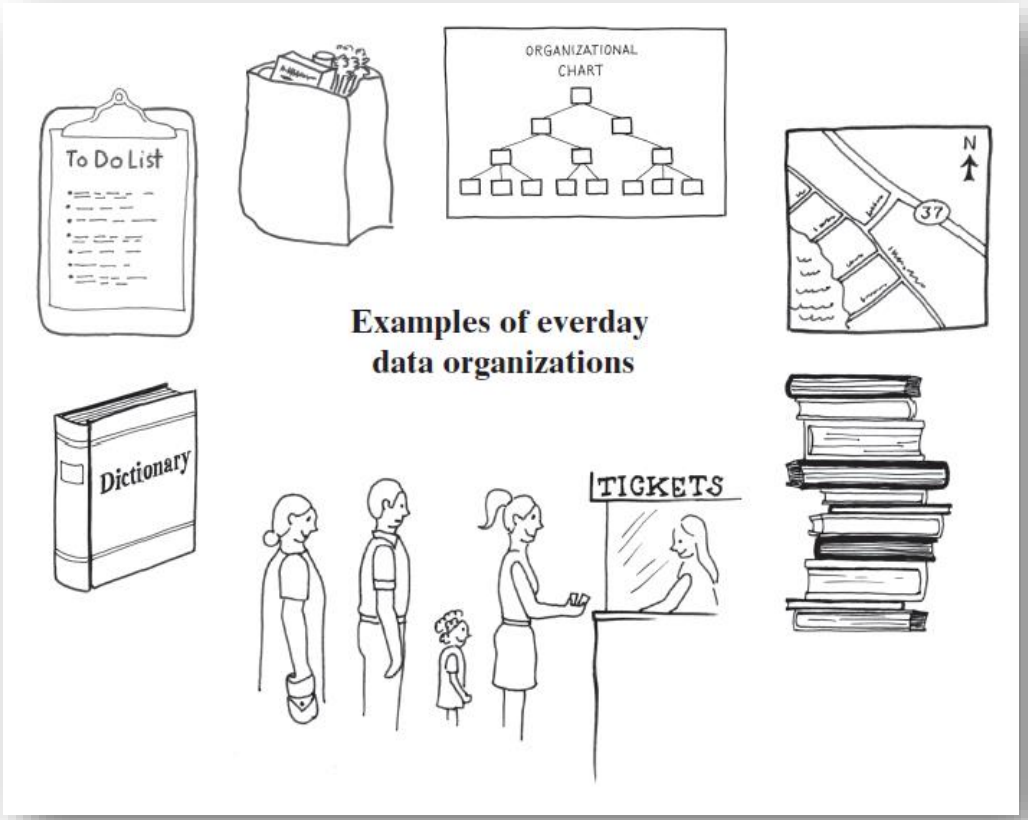
5th Edition



Introduction

Data Organization in Life

- Standing in a line
- Stack of books
- To-Do list
- Dictionary
- Folders, directories on your computer
- Road map



Computer Data Organization

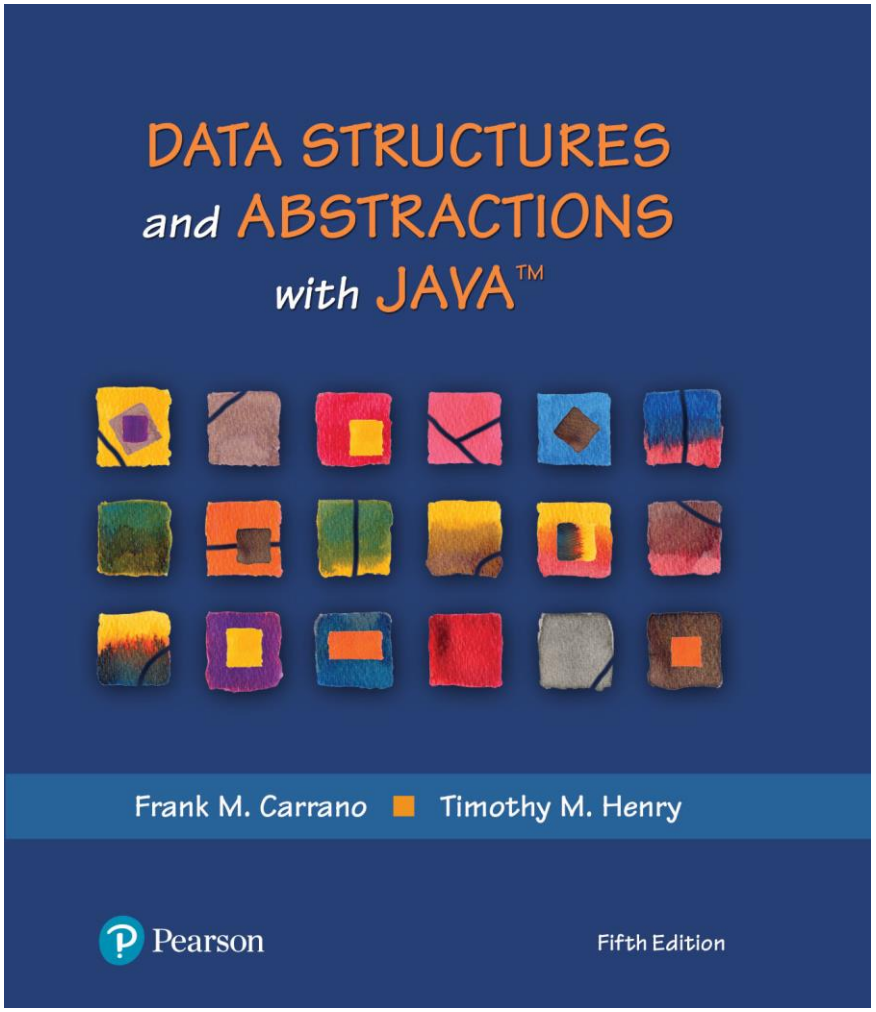
- Abstract Data Type: ADT
- Data Structure
- Collection
- Examples of containers
  - Bag
  - List
  - Stack
  - Queue
  - Dictionary
  - Tree
  - Graph

End

Introduction

Data Structures and Abstractions with Java™

5th Edition



Prelude

Designing Classes

Object Oriented Programming

- Encapsulation
- Inheritance
- Polymorphism

Encapsulation

- Information hiding
- Enclose data and methods within a class
- Hide implementation details
- Programmer receives only enough information to be able to use the class

Encapsulation

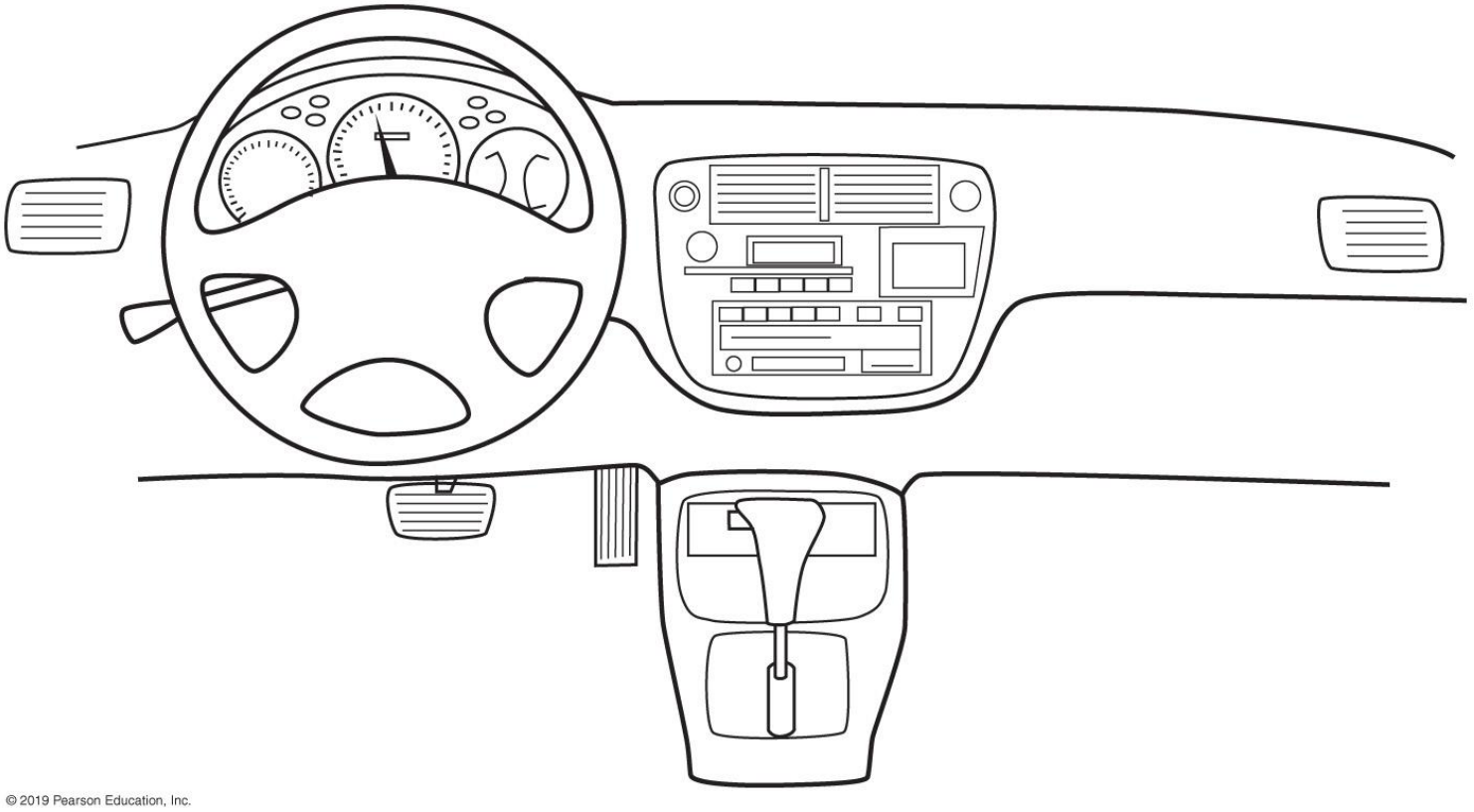


FIGURE P-1 An automobile's controls are visible to the driver, but its inner workings are hidden

Abstraction

- Focus on what instead of how
  - What needs to be done?
  - For the moment ignore how it will be done.
- Divide class into two parts
  - Client interface
  - Implementation

Abstraction

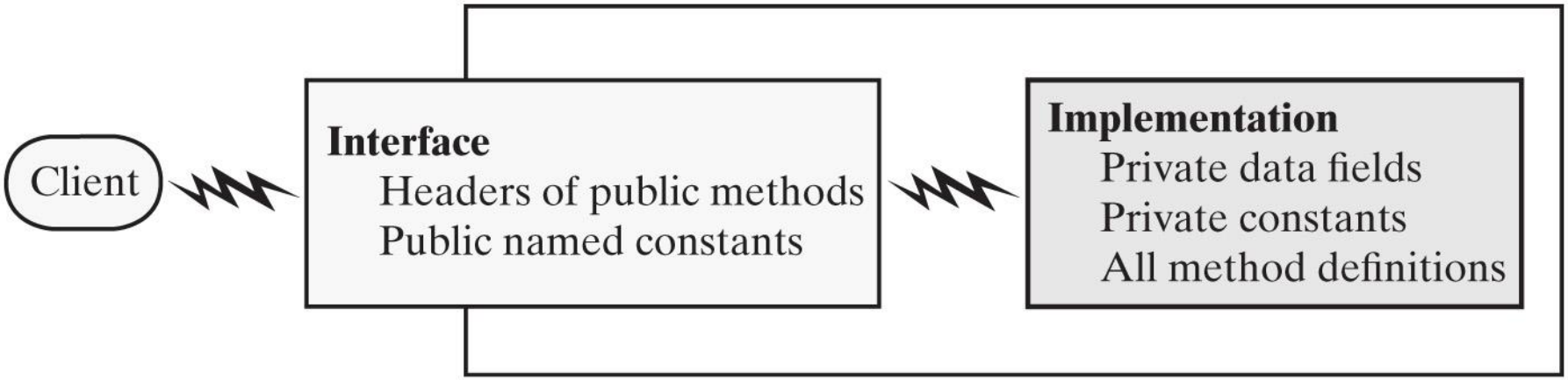


FIGURE P-2 An interface provides well-regulated communication between a hidden implementation and a client

Specifying Methods

- Preconditions
  - What must be true before method executes
  - Implies responsibility for client
- Postconditions
  - Statement of what is true after method executes
- Use assertions
  - In comments or with assert statement

Java Interfaces

- Program component that declares a number of public methods
  - Should include comments to inform programmer
  - Any data fields here should be public, final, static

constant

Interface Measurable

```
/**
   An interface for methods that return
   the perimeter and area of an object.
*/
public interface Measurable
{
   /** Gets the perimeter.
      @return  The perimeter. */
   public double getPerimeter();

   /** Gets the area.
      @return  The area. */
   public double getArea();
} // end Measurable
```

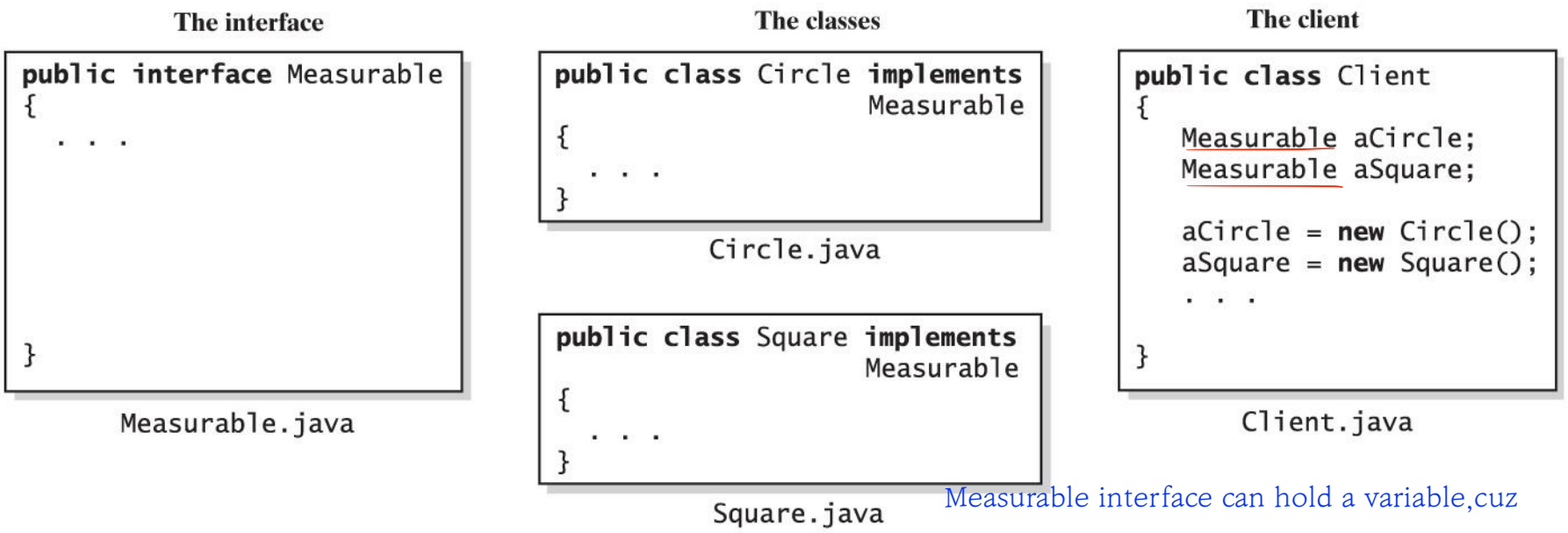
Listing 2-1

Interface NameMeasurable

```
/** An interface for a class of names. */
public interface NameInterface
{
   /** Sets the first and last names.
      @param firstName  A string that is the desired first name.
      @param lastName   A string that is the desired last name. */
  public void setName(String firstName, String lastName);

   /** Gets the full name.
      @return  A string containing the first and last names. */
   public String getName();

  public void setFirst(String firstName);
  public String getFirst();

  public void setLast(String lastName);
  public String getLast();

  public void giveLastNameTo(NameInterface aName);
  public String toString();
} // end NameInterface
```

Listing 2-2

Implementing an Interface



Measurable interface can hold a variable,cuz

Circle object will implements Measurable variable

FIGURE P-3 The files for an interface, a class that implements the interface, and the client

Implementing an Interface

- A way for programmer to guarantee a class has certain methods
- Several classes can implement the same interface
- A class can implement more than one interface

Interface as a Data Type

- You can use a Java interface as you would a data type
- Indicates variable can invoke certain set of methods and only those methods.
- An interface type is a reference type
- An interface can be used to derive another interface by using inheritance

Interface vs. Abstract Class

- Purpose of interface similar to that of abstract class
  - But an interface is not a class
- Use an abstract class …
  - If you want to provide a method definition
  - Or declare a private data field that your classes will have in common
- A class can implement several interfaces but can extend only one abstract class.

Named Constants Within an Interface

- An interface can contain named constants,
  - Public data fields that you initialize and declare as final.
- Options:
  - Define the constants in an interface that the classes implement
  - Define your constants in a separate class instead of an interface

Choosing Classes

- Consider a registration system for your school …
- Issues:
  - Who, what will use the system?
  - What can each actor do with the system?
  - Which scenarios involve common goals?

Choosing Classes

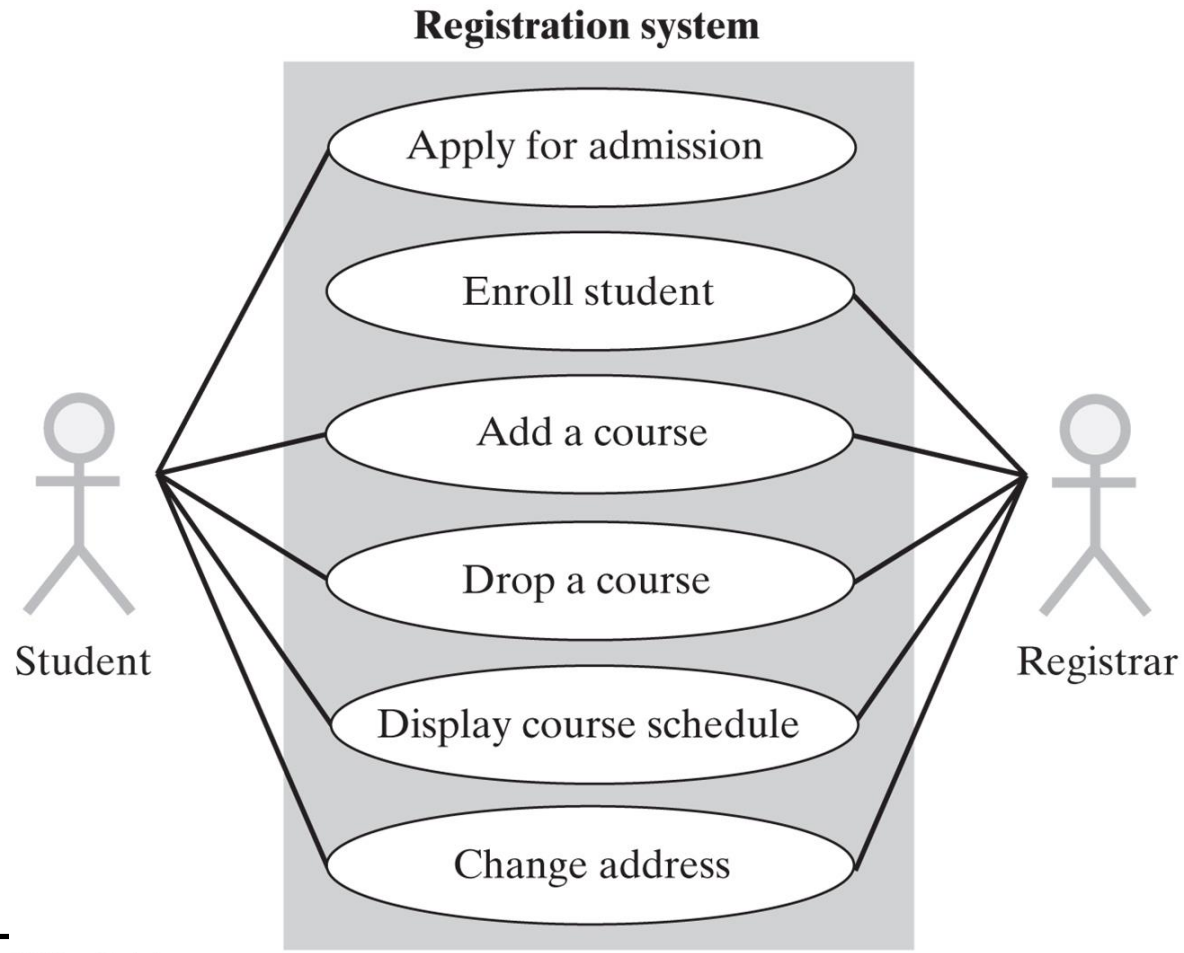


FIGURE [illegible] ion system

Identifying Classes

System: Registration
Use case: Add a course
Actor: Student
Steps:

1. Student enters identifying data.
2. System confirms eligibility to register.
    a. If ineligible to register, ask student to enter identification data again.
    b. Student chooses a particular section of a course from a list of course offerings.
    c. System confirms availability of the course.
    d. If course is closed, allow student to return to Step 3 or quit.
    e. System adds course to student's schedule.
    f. System displays student's revised schedule of courses.

FIGURE P-5 A description of a use case for adding a course

CSC Card Example

CourseSchedule
Responsibilities
Add a course
Remove a course
Check for time conflict
List course schedule
Collaborations
Course
Student

FIGURE P-6 A class-responsibility-collaboration (CRC) card

Unified Modeling Language Class

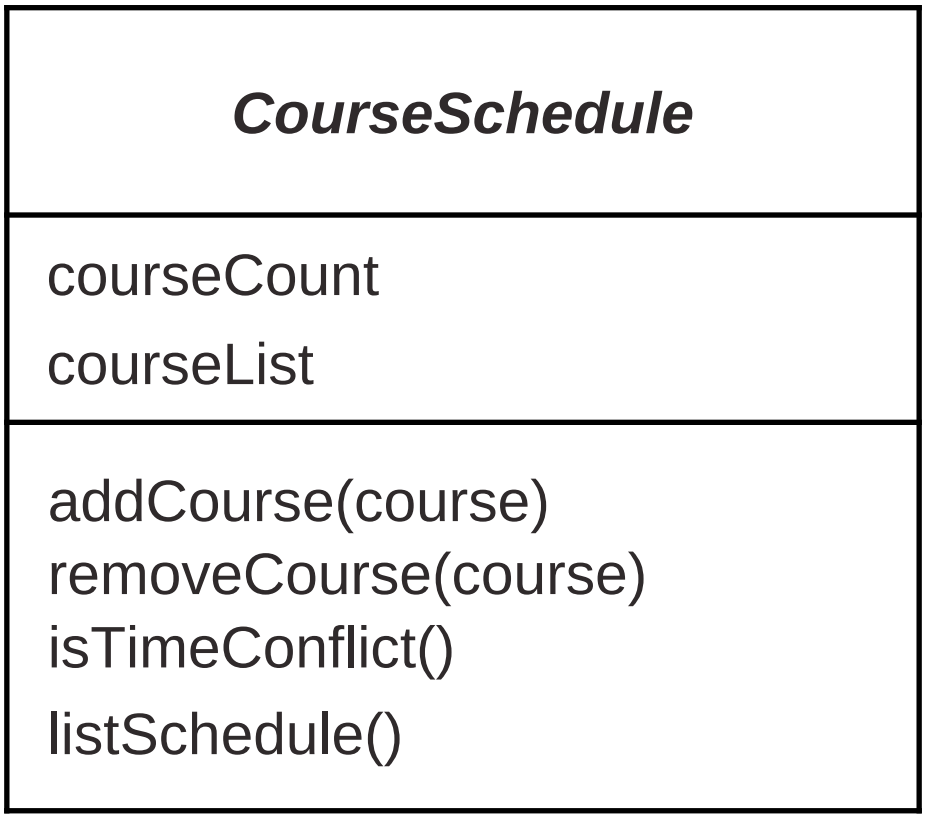


FIGURE P-7 A class representation that can be a part of a class diagram

UML Interface Example

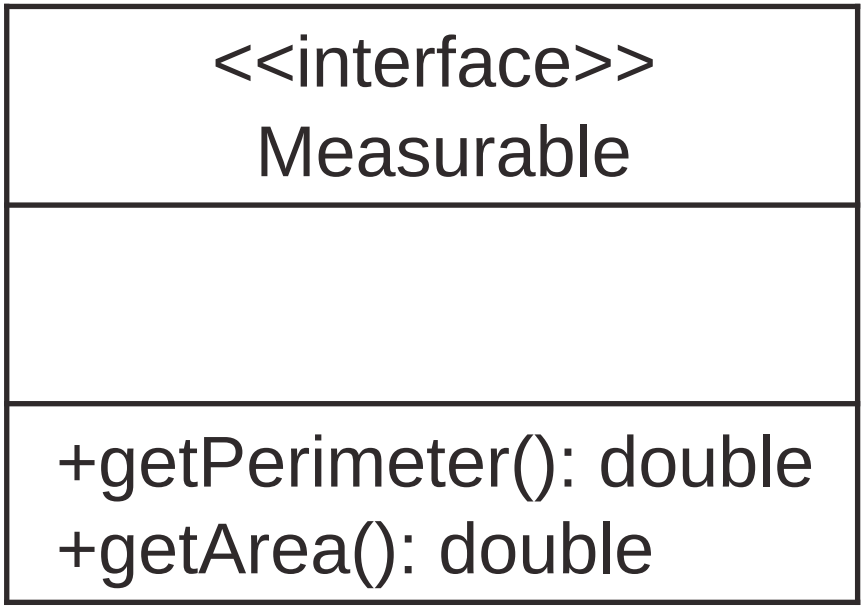


FIGURE P-8 UML notation for the interface `Measurable`

UML Class Hierarchy

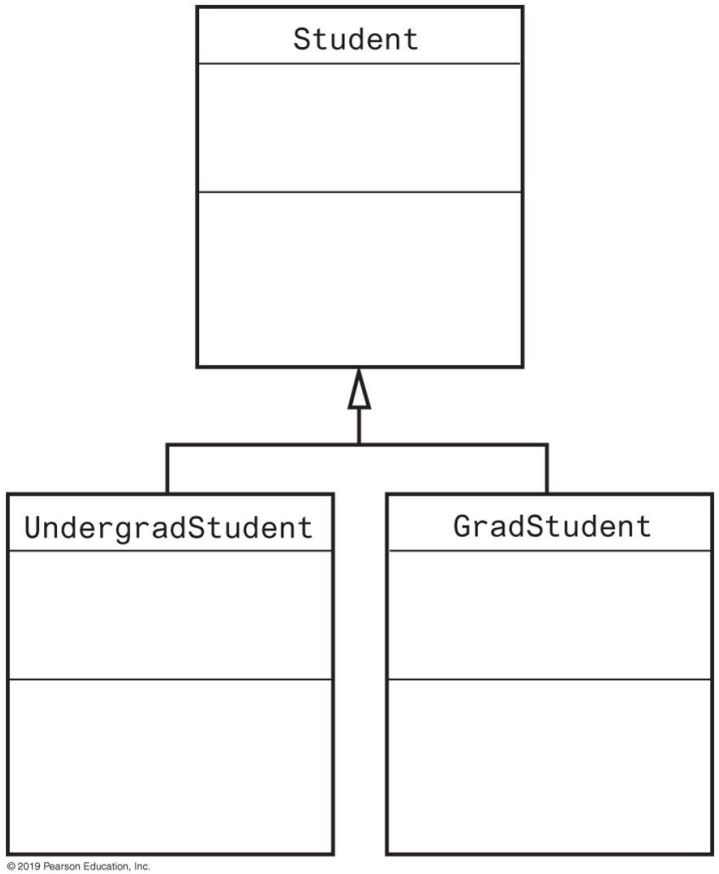


FIGURE P-9 A class diagram showing the base class `Student` and two subclasses

UML Interface Implementation

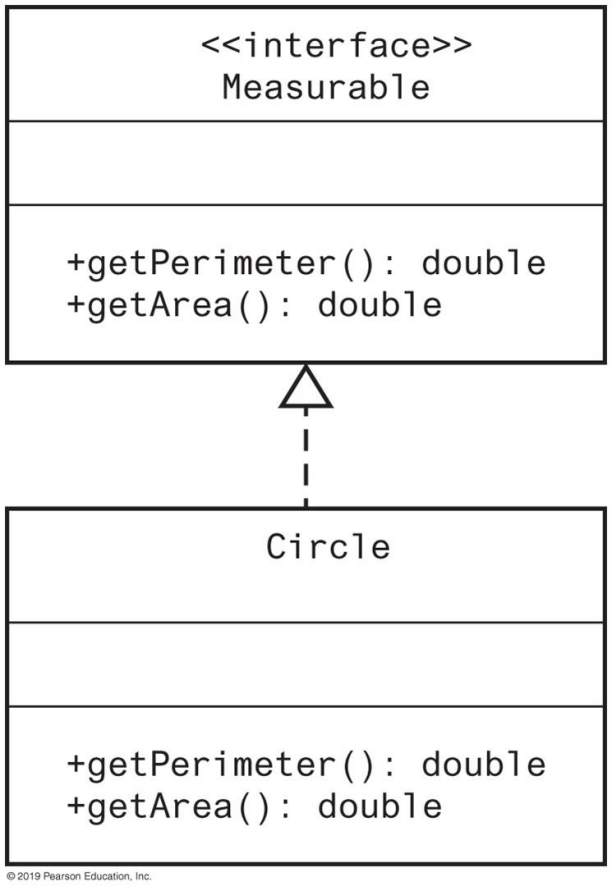


FIGURE P-10 A class diagram showing the class `Circle` that implements the interface `Measurable`

UML Class Associations

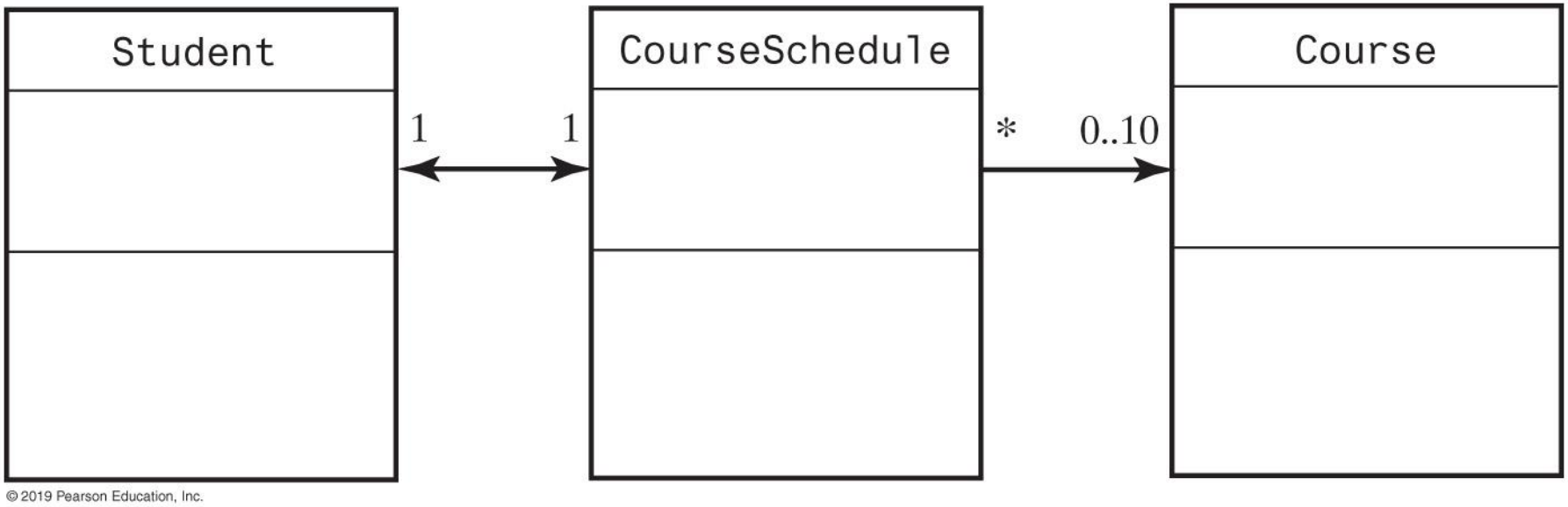


FIGURE P-11 Part of a UML class diagram with associations

Reusing Classes

- Not all programs designed and written “from scratch”
- Actually, most software created by combining
  - Already existing components with
  - New components
- Saves time and money
- Reused components are already tested

End

Prelude