

# Object-Oriented Programming

Weng Kai  
Fall 2020

# Problem Solving

## KEY CONCEPT

Program design involves breaking a solution down into manageable pieces.

bj6c62\*

© 2010 Pearson Education, Inc. All rights reserved.

- In general, problem solving consists of multiple steps:
  1. Understanding the problem.
  2. Designing a solution.
  3. Considering alternatives to the solution and refining the solution.
  4. Implementing the solution.
  5. Testing the solution and fixing any problems that exist.

# Object-Oriented Software Principles

- Object-oriented programming ultimately requires a solid understanding of the following terms:
  - object
  - attribute
  - method
  - class
  - encapsulation
  - inheritance
  - polymorphism

# Object

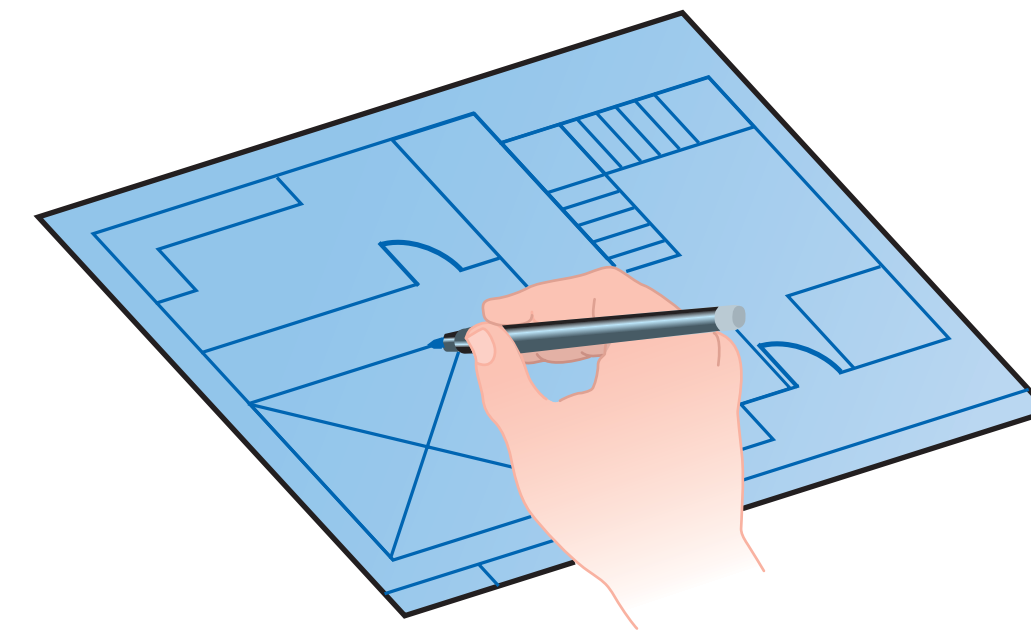
## KEY CONCEPT

Each object has a state, defined by its attributes, and a set of behaviors, defined by its methods.

- An object is a fundamental element in a program. A software object often represents a real object in our problem domain, such as a bank account. Every object has a state and a set of behaviors. By “state” we mean state of being—fundamental characteristics that currently define the object.
- An object’s attributes are the values it stores internally, which may be represented as primitive data or as other objects. Collectively, the values of an object’s attributes define its current state.
- A method is a group of programming statements that is given a name. The methods of an object define its potential behaviors.

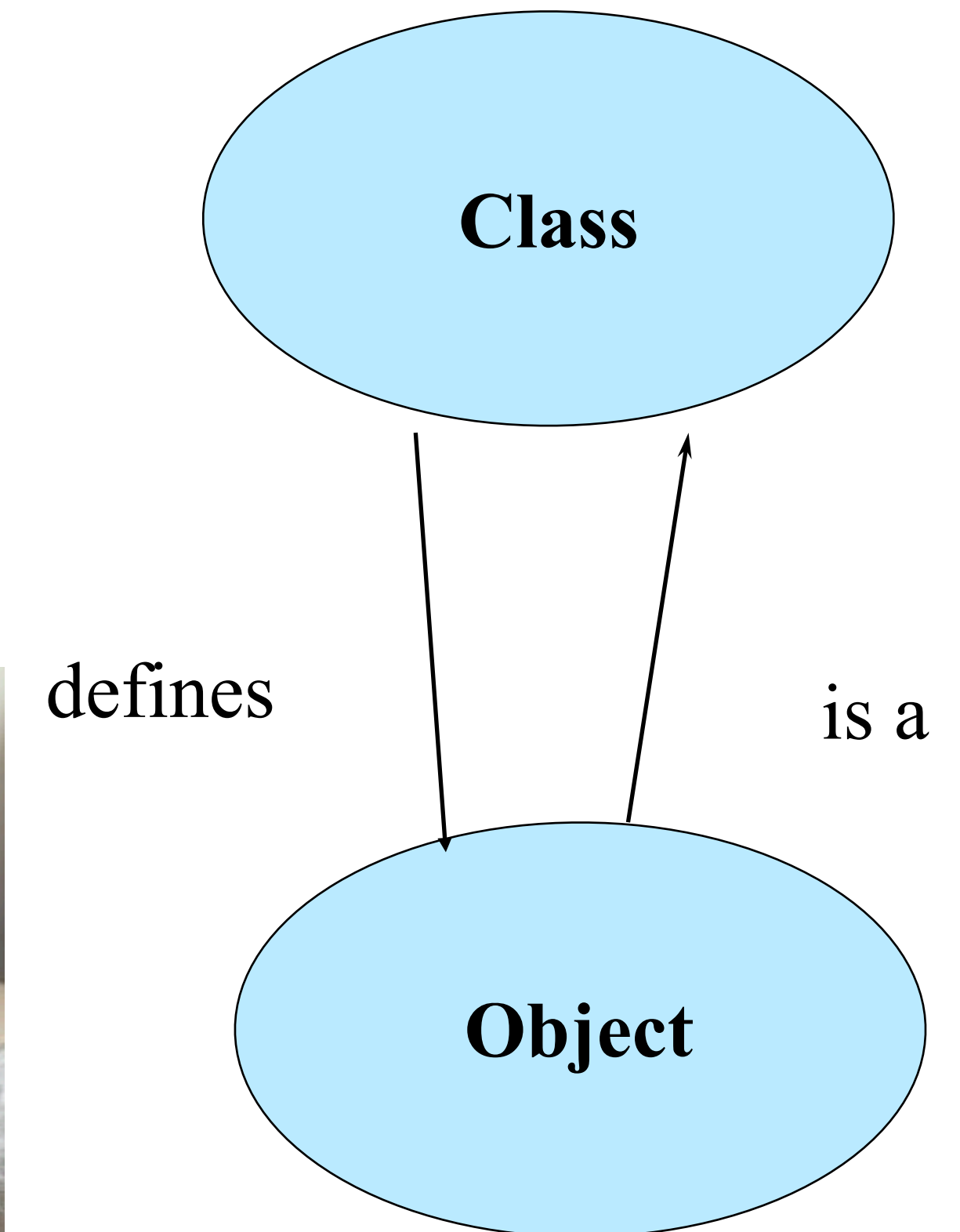
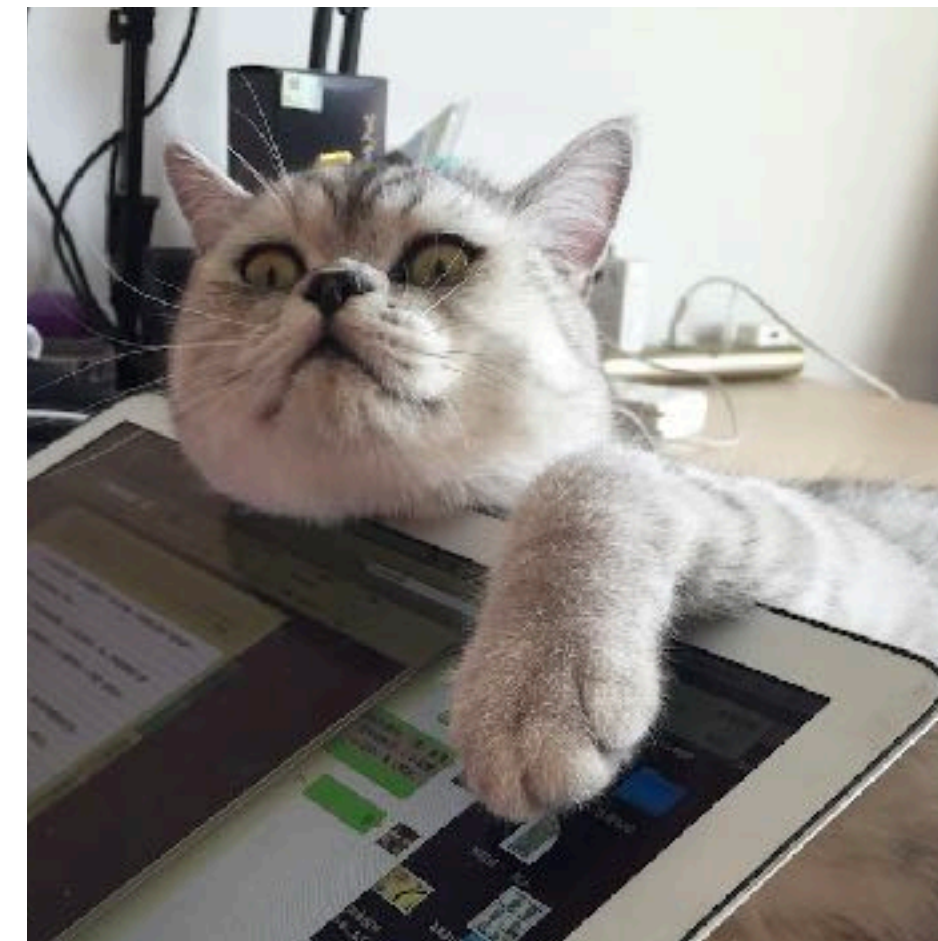
# Class

- An object is defined by a class. A class is the model or blueprint from which an object is created.
- It establishes the kind of data an object of that type will hold and defines the methods that represent the behavior of such objects.
- Once a class has been defined, multiple objects can be created from that class.



# Object vs. Class

- Objects (cat)
  - Represent things, events, or concepts
  - Respond to messages at run-time
- Classes (cat class)
  - Define properties of instances
  - Act like types in C++



# OOP Characteristics

- Everything is an object.
- A program is a bunch of objects telling each other what to do by sending messages.
- Each object has its own memory made up of other objects.
- Every object has a type.
- All objects of a particular type can receive the same messages.

# Object-Oriented

- Is a way to map the real world into the virtual world.
  - Not the time-sequence, but the items in the event and their relationship



Class	Attributes	Operations
Student	Name Address Major Grade point average	Set address Set major Compute grade point average
Rectangle	Length Width Color	Set length Set width Set color
Aquarium	Material Length Width Height	Set material Set length Set width Set height Compute volume Compute filled weight
Flight	Airline Flight number Origin city Destination city Current status	Set airline Set flight number Determine status
Employee	Name Department Title Salary	Set department Set title Set salary Compute wages Compute bonus Compute taxes

# Questions

- List some attributes and operations that might be defined for a class called Book that represents a book in a library.
- True or False? Explain.
  - We should use only classes from the Java standard class library when writing our programs—there is no need to define or use other classes.
  - An operation on an object can change the state of an object.
  - The current state of an object can affect the result of an operation on that object.
  - In Java, the state of an object is represented by its methods.

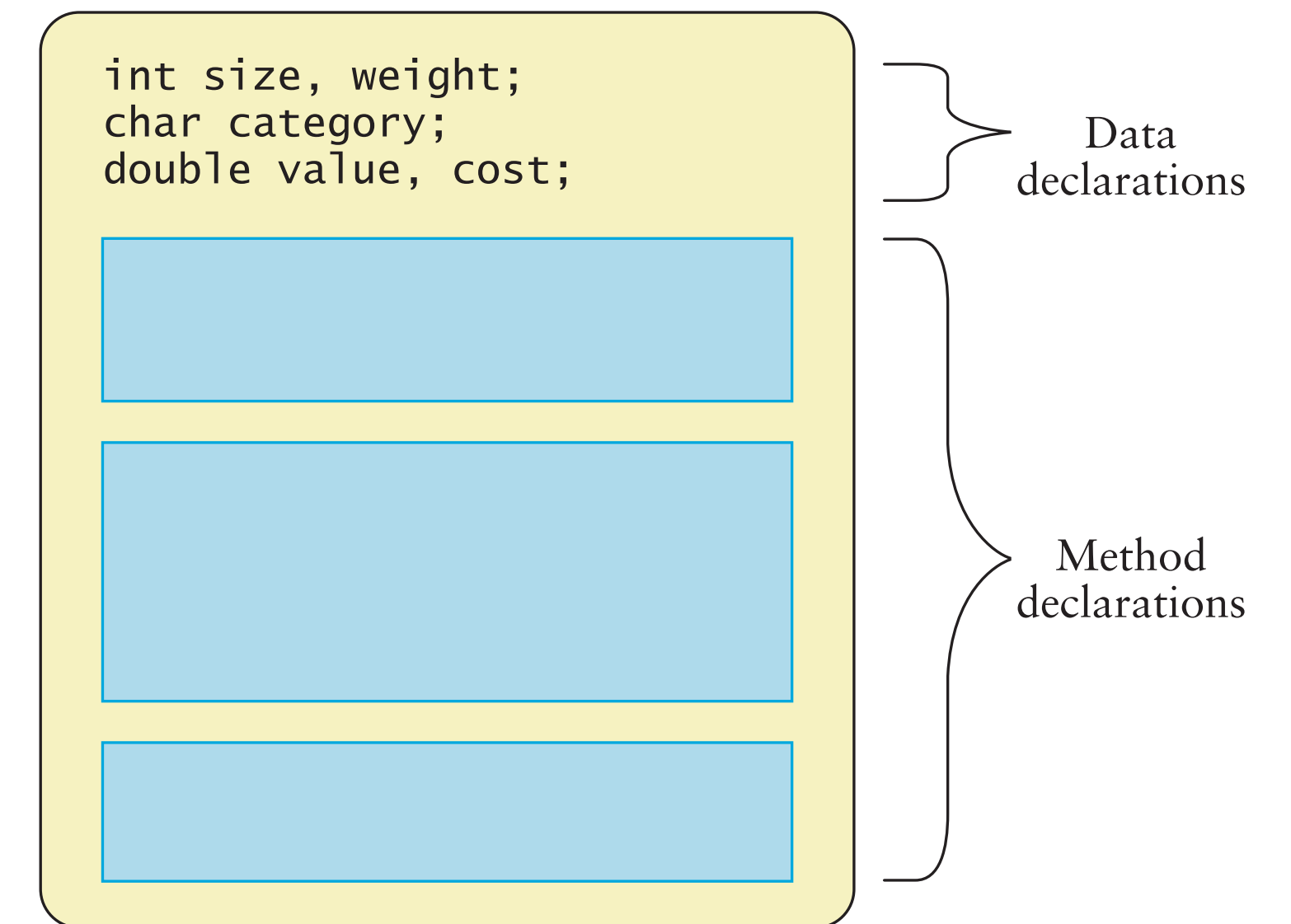
# RollingDice

```
public class RollingDice {  
    public static void main (String[] args) {  
        Die die1, die2;  
        int sum;  
        die1 = new Die();  
        die2 = new Die();  
        die1.roll();  
        die2.roll();  
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);  
        die1.roll();  
        die2.setFaceValue(4);  
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);  
        sum = die1.getFaceValue() + die2.getFaceValue();  
        System.out.println ("Sum: " + sum);  
        sum = die1.roll() + die2.roll();  
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);  
        System.out.println ("New sum: " + sum);  
    }  
}
```



# Die?

- The primary difference between this example and previous examples is that the Die class is not a predefined part of the Java class library. We have to write the Die class ourselves, defining the services we want Die objects to perform, if this program is to compile and run
- Every class can contain data declarations and method declarations



# Die

```
public class Die {
    private final int MAX = 6; // maximum face value
    private int faceValue; // current value showing on the die
    public Die()
    {
        faceValue = 1;
    }
    public int roll()
    {
        faceValue = (int)(Math.random() * MAX) + 1;
        return faceValue;
    }
    public void setFaceValue (int value)
    {
        faceValue = value;
    }
    public int getFaceValue()
    {
        return faceValue;
    }
    public String toString() {
        String result = Integer.toString(faceValue);

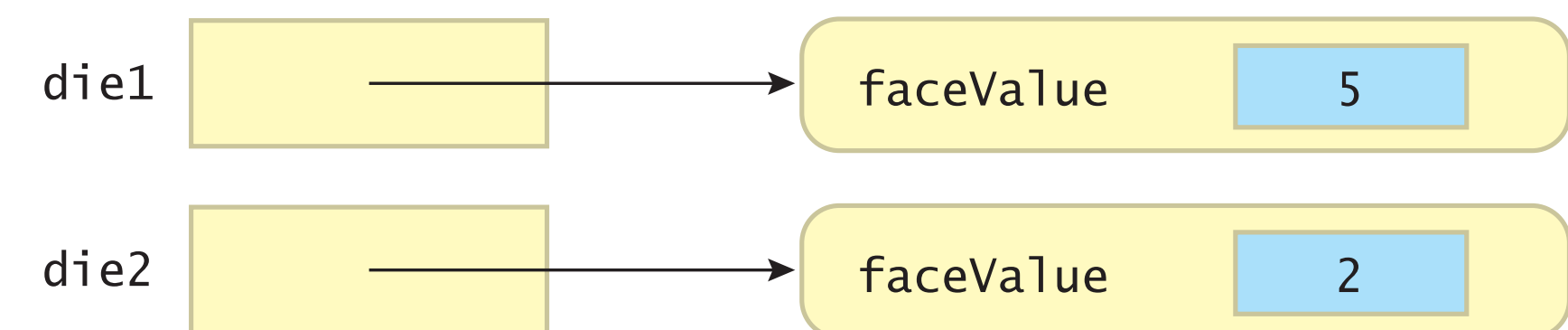
        return result;
    }
}
```

# Instance Data

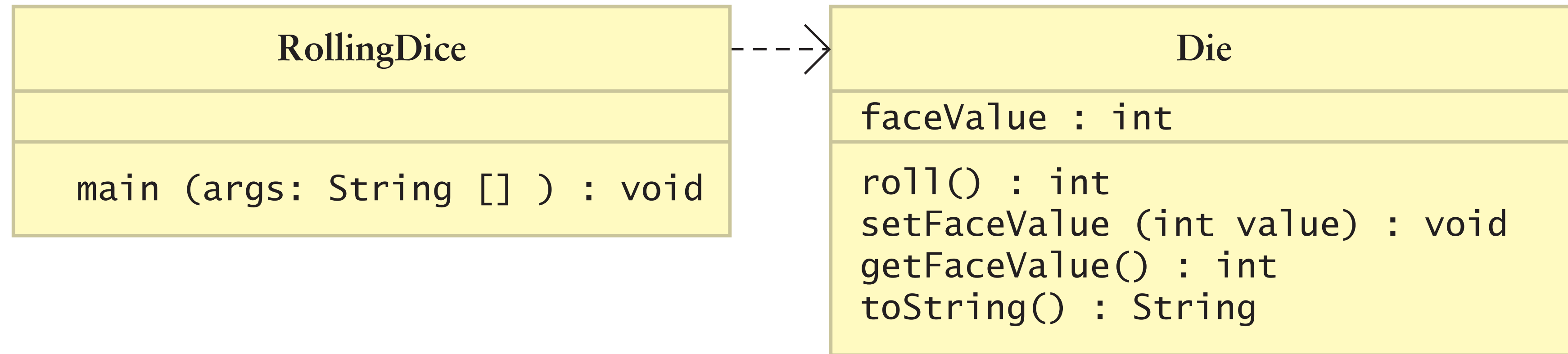
## KEY CONCEPT

The scope of a variable, which determines where it can be referenced, depends on where it is declared.

- The location at which a variable is declared defines its *scope*, which is the area within a program in which that variable can be referenced
- Attributes such as the variable `faceValue` are called instance data because new memory space is reserved for that variable every time an instance of the class that is created
- Java automatically initializes any variables declared at the class level to zero



# Classes



- RollingDice uses Die



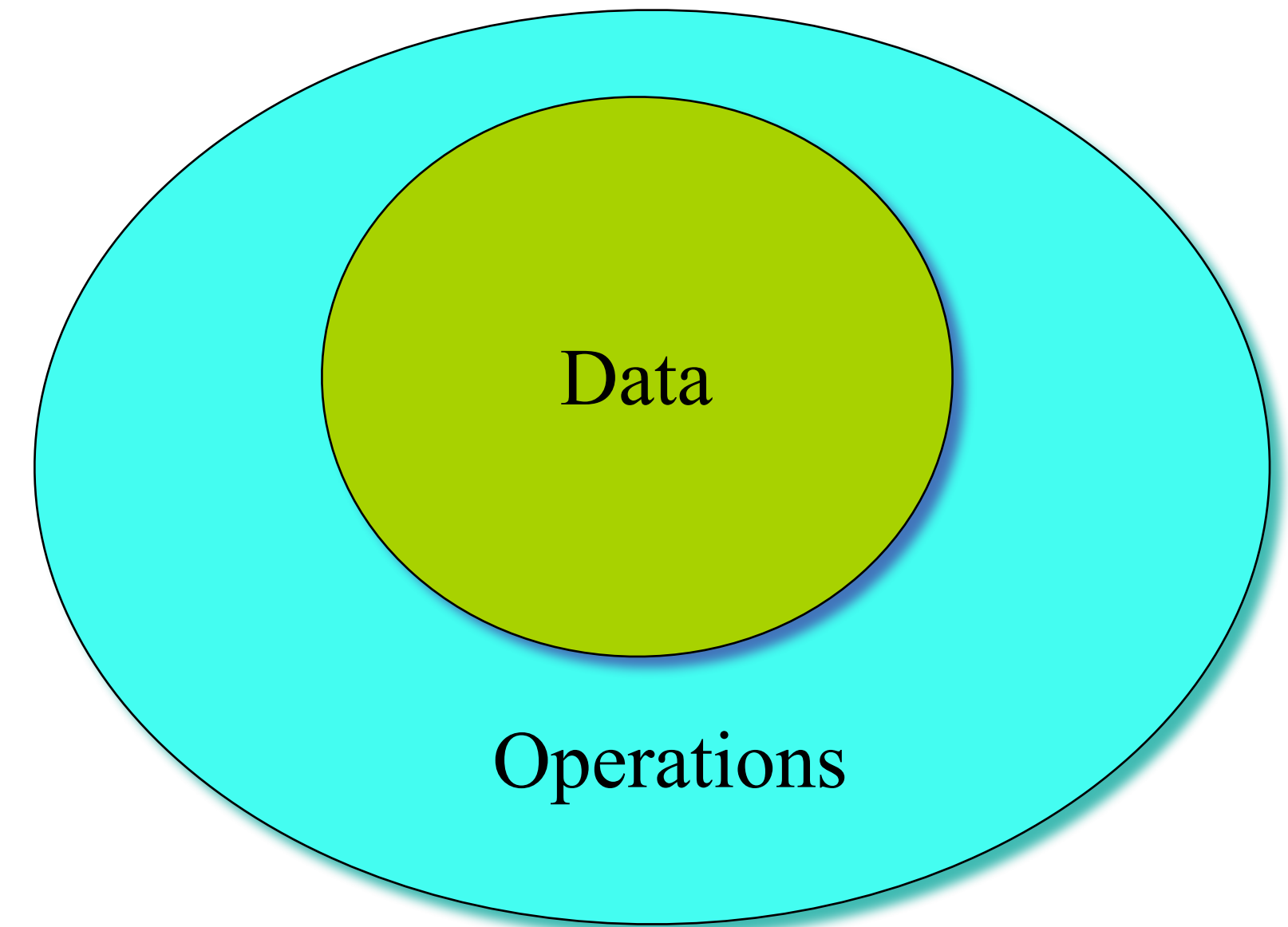
# Compilation Unit

- Each compilation unit must have a name ending in .java, and inside the compilation unit there can be a public class that must have the same name as the file. There can be only one public class in each compilation unit
- When you compile a .java file you get an output file with exactly the same name but an extension of .class for each class in the .java file
- Usually there's only one class in one source code file
  - Or other classes are intended to be hidden
- A working program is a bunch of .class files



# Encapsulation

- Objects = Attributes + Services
  - Data: the properties or status
  - Operations: the functions
- The only changes made to the state of the object should be accomplished by that object's methods.



# Encapsulation

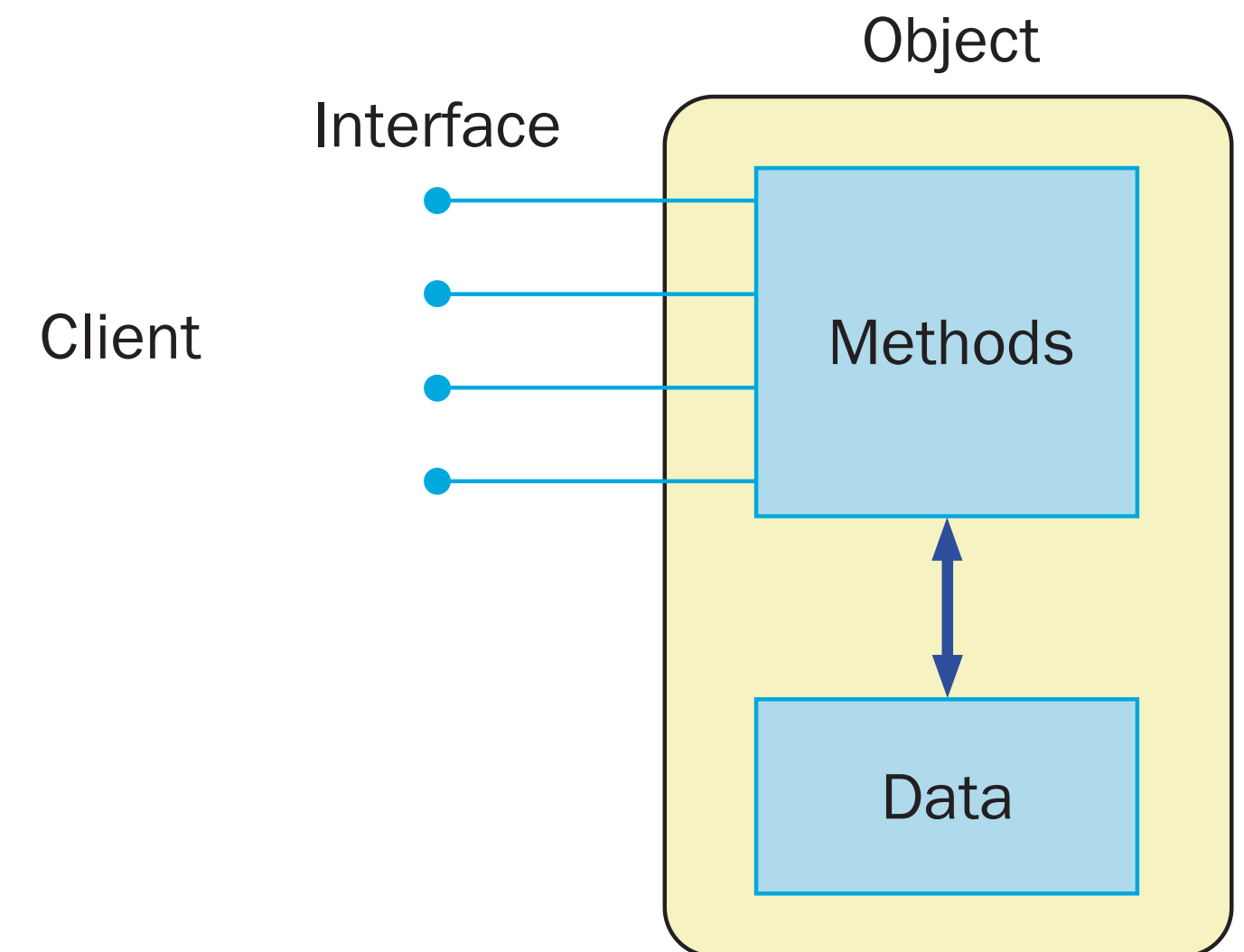
## KEY CONCEPT

An object should be encapsulated, guarding its data from inappropriate access.

9CC622'

Remember to pass your neighbours

- An object should be encapsulated from the rest of the system. It should interact with other parts of a program only through the specific set of methods that define the services that that object provides. These methods define the interface between that object and the program that uses it



# Visibility Modifiers

## KEY CONCEPT

Instance variables should be declared with private visibility to promote encapsulation.

promote encapsulation

accessed with public visibility

	public	private
Variables	Violate encapsulation	Enforce encapsulation
Methods	Provide services to clients	Support other methods in the class

- Some of the Java modifiers are called visibility modifiers because they control access to the members of a class. The reserved words **public** and **private** are visibility modifiers that can be applied to the variables and methods of a class
- If a member of a class has public visibility, it can be directly referenced from outside of the object.
- If a member of a class has private visibility, it can be used anywhere inside the class definition but cannot be referenced externally.

# Accessors and Mutators

- Generally, accessor method names have the form getX, where X is the value to which it provides access. Likewise, mutator method names have the form setX, where X is the value they are setting. Therefore these types of methods are some- times referred to as “getters” and “setters.”
- IDEA and other IDEs provide facility to generate getters and setters

# Constructors

- A constructor is a special method that has the same name as the class
- A constructor is similar to a method that is invoked when an object is instantiated, automatically
- In particular, we often use a constructor to initialize the variables associated with each object

# Init at Declaration

- Member variables can be initialized at declaration
- This kind of initialization happens before constructor
- Can be used to new object or call functions

# Rules to Initialization

- Leave zero blank
- Make common init value as declaration init
- Constructor for individual init
  - Using parameter values to init member variables

# Identifying Classes and Objects

The user must be allowed to specify each product by its primary characteristics, including its name and product number. If the bar code does not match the product, then an error should be generated to the message window and entered into the error log. The summary report of all transactions must be structured as specified in section 7.A.

- One way to identify potential classes is to identify the objects discussed in the program requirements. Objects are generally nouns
- A class represents a group of objects with similar behavior.
- Classes that represent objects should generally be given names that are singular nouns, such as Coin, Student, and Message
- Given the needs of a particular program, we want to strike a good balance between classes that are too general and those that are too specific



# Assigning Responsibilities

- Assigning responsibilities to each class: Each class represents an object with certain behaviors that are defined by the methods of the class. Any activity that the program must accomplish must be represented somewhere in the behaviors of the classes. That is, each class is responsible for carrying out certain activities, and those responsibilities must be assigned as part of designing a program
- Sometimes it is challenging to determine which is the best class to carry out a particular responsibility

# Question

- Think about an activity in your daily life, identify objects involved, draw a diagram about all the objects and connections between them. After that, describe the activity and its procedure. There has to be at least three objects involved in the activity.

# Class Relationships

- The classes in a software system have various types of relationships to each other.
- Three of the more common relationships are
  - dependency,
  - aggregation, and
  - inheritance.
- Generally, if class A uses class B, then one or more methods of class A invoke one or more methods of class B. If an invoked method is static, then A merely references B by name. If the invoked method is not static, then A must have access to a specific instance of class B in order to invoke the method. That is, A must have a reference to an object of class B.

# Clock display

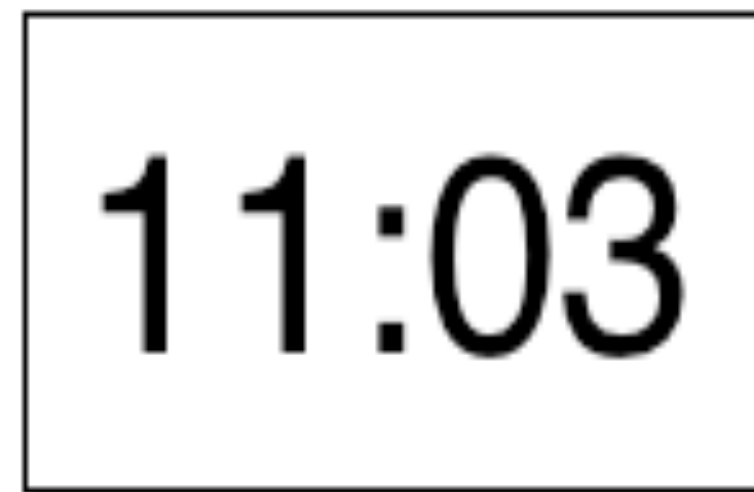


11:03

# Modularization

- Modularization is the process of dividing a whole into well-defined parts, which can be built and examined separately, and which interact in well-defined ways.

# Modularizing the clock display



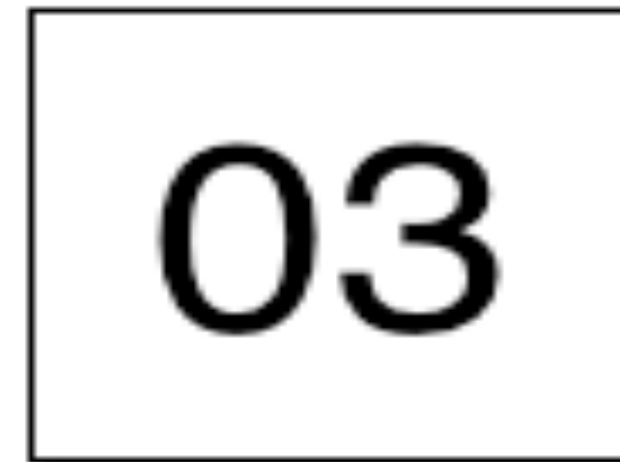
11:03

One four-digit display?

Or two two-digit displays?

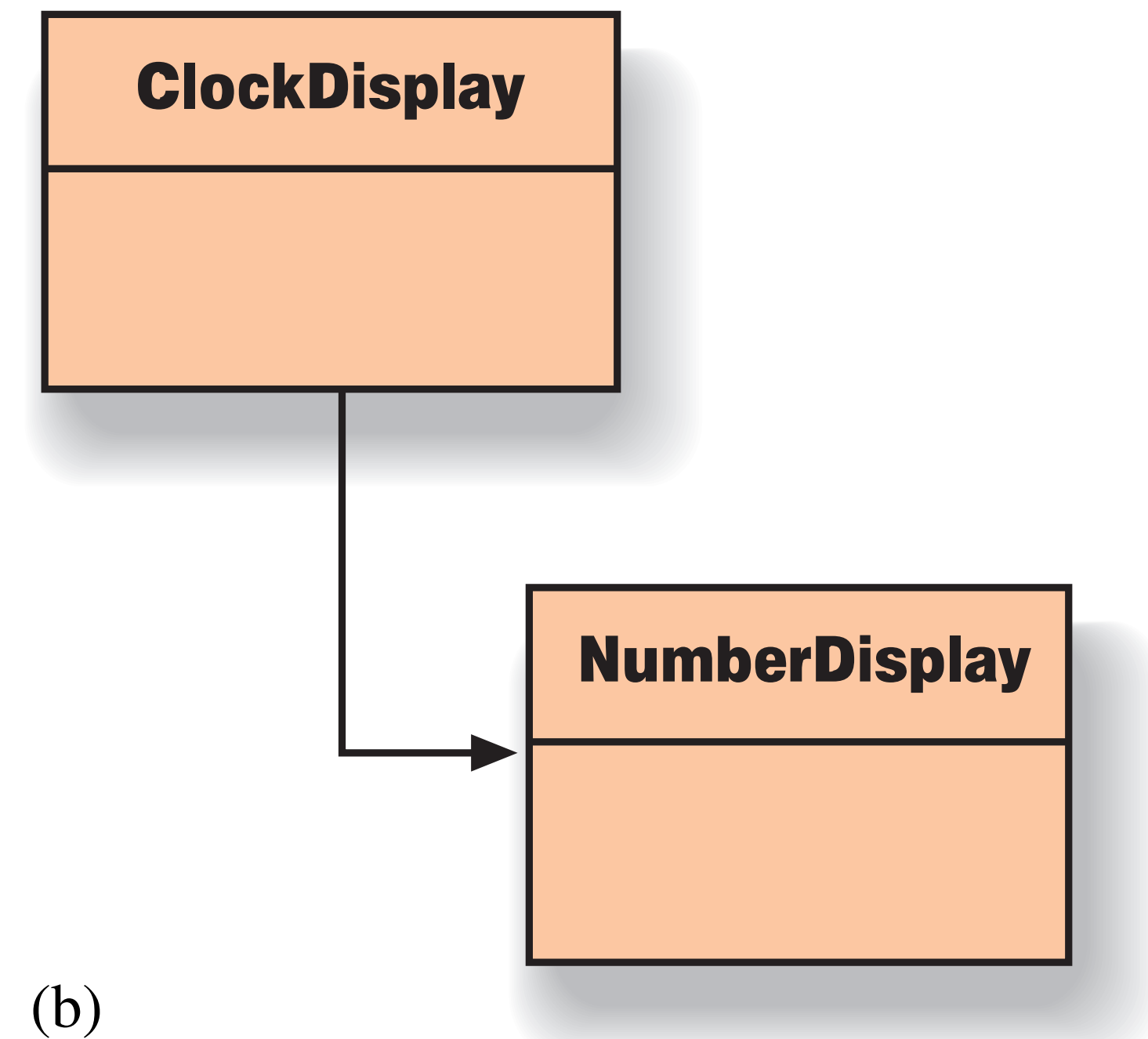
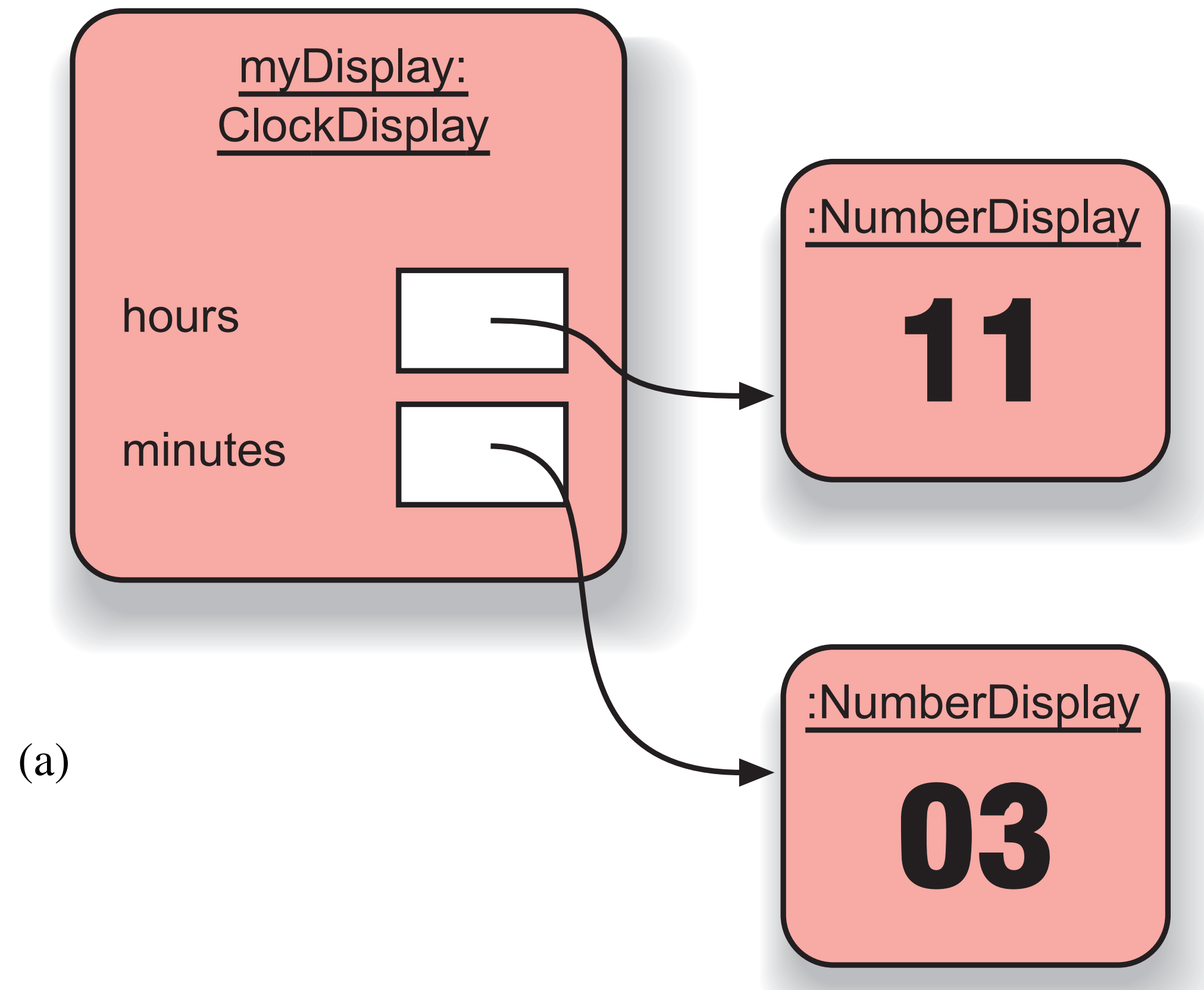


11

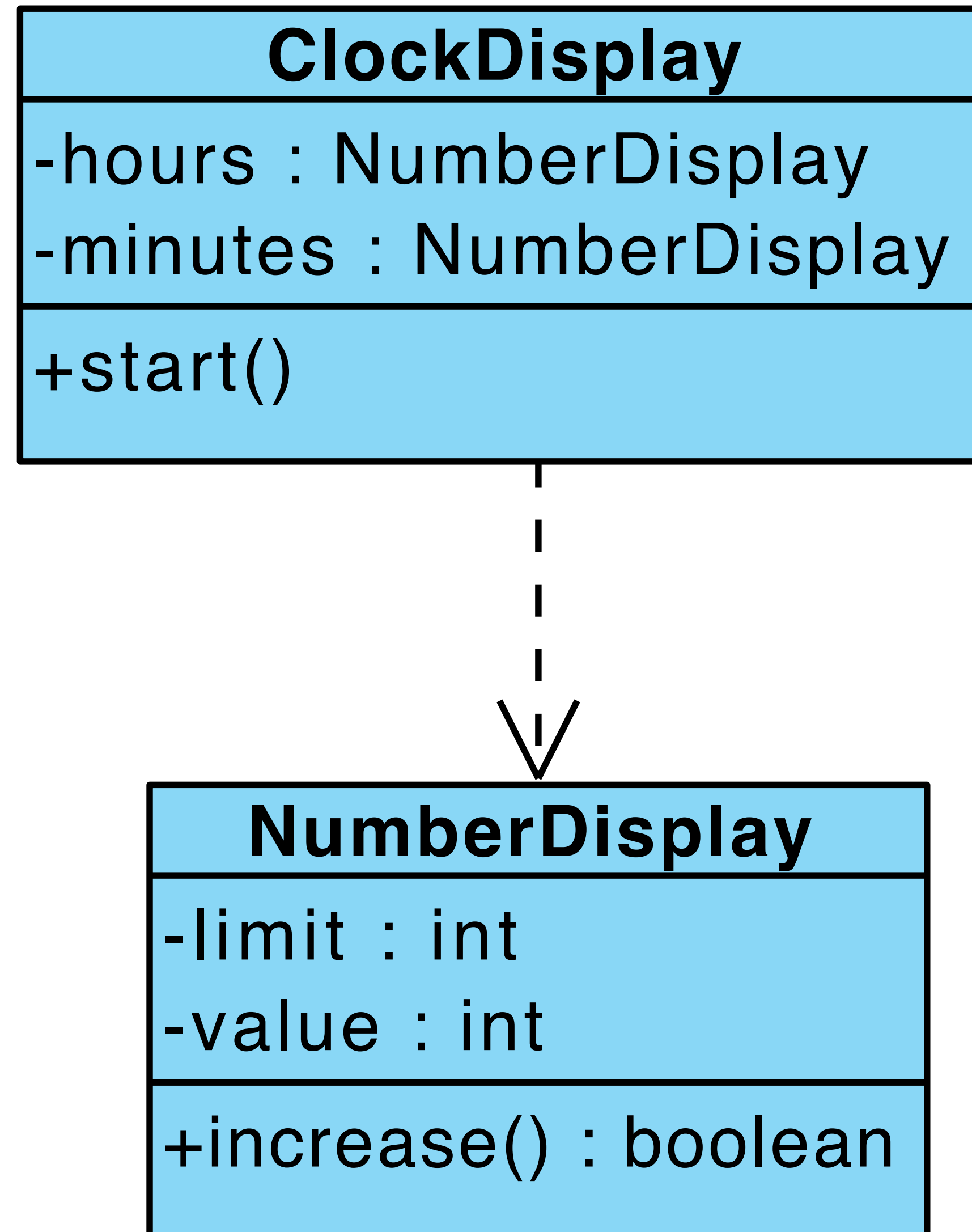


03

# Objects vs Classes



# Class Diagram





# ClockDisplay

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

    Constructor and
    methods omitted.
}
```

# NumberDisplay

```
public class NumberDisplay
{
    private int limit;
    private int value;

    Constructor and
    methods omitted.
}
```