Chapter 7: Introduction to Classes and Objects
Objectives

- Understand classes, objects, instance variables
- Learn to declare a class and use it to create an object
- Learn to declare instance methods to implement class behavior
- Learn to declare instance variables to implement class attributes
- Learn to use a constructor to initialize an object when it is created
Object-Oriented Programming

- Each Java program consists of one or more classes
- Each class represents a type of objects (e.g., those in the physical world such as a car and a printer)
- Objects interact with each other for computation
- Three key concepts: class, object, method
The Car Driving Analogy

- Suppose that our computational task is to drive a car and accelerate it by pressing down on its accelerator pedal (油门)

How to make it happen?
The Car Driving Analogy

- The very first step: Before you can drive a car, someone has to design it (engineering drawings / blueprints).
The Car Driving Analogy

- The car’s blueprints should also include the design for an accelerator pedal.

- A lot of other components should also be designed, e.g., the brake pedal (刹车), the steering wheel (方向盘).

  - We don’t need to know the complex mechanisms behind the design to drive the car.

  - Have you ever wondered how characters are printed on screen by System.out.println() method?
We cannot drive a car’s engineering drawings
The Car Driving Analogy

- We cannot drive a car’s engineering drawings
- Before we drive a car, it must be built from the engineering drawings
- Even building a car is not enough, the driver must press the accelerator pedal to perform the task of driving the car
The Car Driving Analogy

- We cannot drive a car’s engineering drawings [Designing classes]

- Before we drive a car, it must be built from the engineering drawings [Creating concrete instances of a class]

- Even building a car is not enough, the driver must press the accelerator pedal to perform the task of driving the car [Invoking methods for computation]
Detailed Analysis

- When programming in Java, we begin by creating a program unit called `class`, just like we begin with engineering draws in the driving example.

- In a class, we provide one or more `methods` that are designed to perform the class’s tasks.

- Methods hide from users the complex tasks that they perform, just like the accelerator pedal of a car hides from the driver the complex mechanisms that make the car move faster.
Detailed Analysis

- We cannot drive a car’s engineering drawings. Similarly, we cannot “drive” a class to perform a task.
- Just as we have to build a car from its engineering drawings before driving it, we must build an object of a class before getting the program to perform tasks.
When driving a car, pressing the accelerator pedal sends a message to the car to perform a task – make the car go faster.

Similarly, we send a message to an object by a method call to tell the method of the object to perform its task.
A car can have many attributes: its color, the amount of gas in its tank, its current speed, and the total miles driven.

These attributes are represented as part of a car’s design.

As you drive a car, these attributes are always associated with the car (not other cars of the same model).

Every car maintains its own attributes (e.g., knowing how much gas is left in its tank, but do not know about other cars).
Instance Variables

- Similarly, an object has attributes that are carried with the object as it’s used in a program.
  - These attributes are specified as the class’s instance variables.
  - For example, a bank account object has a balance attribute that represents the amount of money in that account.
The Whole Picture

- **Class** – a car’s engineering drawings (a blueprint)
- **Method** – designed to perform tasks (making a car move)
- **Object** – the car we drive
- **Method call** – perform the task (pressing the accelerator pedal)
- **Instance variable** – to specify the attributes (the amount of gas)
Declaring a Class

Every class declaration contains the keyword `public` followed by the class' name.

```
public class GradeBook {
    // every class' body is enclosed in a pair of left and right curly braces

}
```

The access modifier `public` indicates that the declared class is visible to all classes everywhere.
Declaring a Method

A class usually consists of one or more methods.

Method = **Method header** + **Method body** (enclosed by `{}`).

```java
public class GradeBook {
    // display welcome message to the user
    public void displayMessage() {
        System.out.println("Welcome to the Grade Book!");
    }
}
```
Declaring a Method

The **return type** specifies the type of data the method returns after performing its task, **void** means returning nothing to its calling method.

```java
public class GradeBook {
   // display welcome message to the user
   public void displayMessage() {
      System.out.println("Welcome to the Grade Book!");
   }
}
```

The **access modifier** `public` indicates that the method is “available to public”, that is, can be called from the methods of other classes.
Declaring a Method

By convention, **method names** are in **Lower Camel Case**: the initial letter is in lower case, subsequent words begin with a capital letter.

```java
public class GradeBook {
    // display welcome message to the user
    public void displayMessage() {
        System.out.println("Welcome to the Grade Book!");
    }
}
```

The parentheses enclose the information that the method requires to perform its task. Empty parentheses indicate no information needs.
Declaring a Method

Like class, the method body is also enclosed in `{}`. The method body contains **statements** that perform the method’s task.

```java
public class GradeBook {
    // display welcome message to the user
    public void displayMessage() {
        System.out.println("Welcome to the Grade Book!");
    }
}
```

**Tips:** (1) Don’t forget the ; after a statement; (2) try to use meaningful names when declaring a method to make your programs understandable.
Can We Run the Program?

```java
public class GradeBook {
    // display welcome message to the user
    public void displayMessage() {
        System.out.println("Welcome to the Grade Book!");
    }
}
```

```
Yepangs-MacBook:Desktop yepang$ java GradeBook
Error: Main method not found in class GradeBook, please define the main method as:
    public static void main(String[] args)
```
Define a variable of the type GradeBook. Note that each new class you create becomes a new data type. Java is an extensible language.
public class GradeBookTest {
    public static void main(String[] args) {
        // create a GradeBook object
        // assign it to myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // call myGradeBook's displayMessage method
        myGradeBook.displayMessage();
    }
}

Class instance creation expression. The keyword `new` is used to create a new object of the specified class. Class name + () represent a call to a constructor (a special method used to initialize the object’s data).
public class GradeBookTest {
  public static void main(String[] args) {
    // create a GradeBook object
    // assign it to myGradeBook
    GradeBook myGradeBook = new GradeBook();

    // call myGradeBook's displayMessage method
    myGradeBook.displayMessage();
  }
}

We can use the variable myGradeBook to refer to the created object and call the method displayMessage() using the member operator “.”. Empty parentheses indicate that we provide no data to the called method.
Method Parameters

- Sometimes a method needs **additional information** to perform its task. **Parameters** are for this purpose.

```java
public class GradeBook {
    // display welcome message to the user
    public void displayMessage(String courseName) {
        System.out.printf("Welcome to the Grade Book for the course%s!\n", courseName);
    }
}
```
public class GradeBookTest {
    public static void main(String[] args) {
        GradeBook myGradeBook = new GradeBook();
        myGradeBook.displayMessage("Java Programming");
    }
}

• Here when calling the method displayMessage, we supply a value for the parameter courseName. We call such values arguments.

• (Parameter vs. Argument, 形式参数与实际参数) A parameter is the variable that is part of the method’s declaration. An argument is an expression used when calling the method.
More on Instance Variables

- An object has attributes (e.g., the amount of gas of a car) that are carried with the object as it is used in a program.

- Such attributes exist before a method is called on an object and after the method completes execution.

- A class typically consists of one or more methods that manipulate the attributes of a particular object of the class.
More on Instance Variables

Object attributes are represented as variables (called fields) in a class declaration.

```java
public class GradeBook {
    private String courseName;
    public void displayMessage( String courseName ) {
        System.out.printf("Welcome to the Grade Book for the course%s!\n", courseName);
    }
}
```
More on Instance Variables

Each object (instance) of the class has its own copy of an attribute in memory, the **field** that represents the attribute is also know as an **instance variable**.

```java
public class GradeBook {
    private String courseName;
    public void displayMessage(String courseName) {
        System.out.printf("Welcome to the Grade Book for the course%s!\n", courseName);
    }
}
```
Don’t Confuse with Local Variables

Variables declared in the body of a particular method are known as **local variables** and can be only used in that method.

**Instance variables** are declared inside a class declaration, but outside the bodies of the class’ method declarations.
Manipulating Instance Variables with Methods

```java
public class GradeBook {
    private String courseName;

    // method to set the course name
    public void setCourseName(String name) {
        courseName = name;
    }

    // method to retrieve the course name
    public String getCourseName() {
        return courseName;
    }
}
```
Access Modifiers

Most instance variables are declared to be **private for data hiding**. Variables (or methods) declared to be private are accessible only to methods of the class in which they are declared.

```java
public class GradeBook {
    private String courseName;

    public void setCourseName(String name) {
        courseName = name;
    }

    public String getCourseName() {
        return courseName;
    }
}
```
Using Getter and Setter

public class GradeBook {
    private String courseName;

    public void setCourseName(String name) {
        courseName = name;
    }

    public String getCourseName() {
        return courseName;
    }

    public void displayMessage() {
        System.out.printf("Welcome to the grade book for \n%s!\n", getCourseName());
    }
}

Manipulate the value of fields
Retrieve the value of fields
public class GradeBook {
    ...
    public void displayMessage() {
        System.out.printf("Welcome to the grade book for\n%s!\n", getCourseName());
    }
}

import java.util.Scanner;
public class GradeBookTest {
    public static void main(String[] args) {
        GradeBook myGradeBook = new GradeBook();
        Scanner input = new Scanner(System.in);
        System.out.printf("Enter course name: ");
        String theName = input.nextLine();
        myGradeBook.setCourseName(theName);
        System.out.println();
        myGradeBook.displayMessage();
    }
}
Initializing Objects with Constructors

- Each class can provide a special method called a **constructor** to be used to *initialize* an object of a class when the object is created.

- Java requires a constructor call for *every* object that is created.

- Keyword **new** requests memory from the system to store an object, then calls the corresponding class’s constructor to initialize the object.

  ```java
  GradeBook myGradeBook = new GradeBook();
  ```
Initializing Objects with Constructors

GradeBook myGradeBook = new GradeBook();

- The empty parentheses after "new GradeBook" indicate a call to the class’s constructor without arguments
- The compiler provides a default constructor with no parameters in any class that does not explicitly include a constructor
  - When a class has only the default constructor, its instance variables are initialized with default values (e.g., an int variable gets the value 0)
- When you declare a class, you can provide your own constructor to specify custom initialization for objects of your class
Example

```java
public class GradeBook {
    public String courseName; // course name for this grade book
    // constructor initializes courseName with String argument
    public GradeBook(String name) {
        courseName = name; // initializes courseName
    }
    // method to set the course name
    public void setCourseName(String name) {
        courseName = name;
    }
    // ...
}
```
Like a method, a constructor’s parameter list specifies the data it requires to perform its task.

- When creating a new object, the data is placed in the parentheses after the class name: `GradeBook book = new GradeBook("CS102A");`

- A class instance creation expression returns a reference to the new object (the address to its variables and methods in memory).
Initializing Objects with Constructors

```java
public class GradeBookTest {
    public static void main(String[] args) {
        // create GradeBook objects
        GradeBook gradeBook1 = new GradeBook("CS101 Introduction to Java Programming");
        GradeBook gradeBook2 = new GradeBook("CS102 Data Structures in Java");

        // display initial value of CourseName for each GradeBook
        System.out.printf("gradeBook1 course name is: %s\n", gradeBook1.getCourseName());
        System.out.printf("gradeBook2 course name is: %s\n", gradeBook2.getCourseName());
    }
}
```

gradeBook1 course name is: CS101 Introduction to Java Programming
gradeBook2 course name is: CS102 Data Structures in Java
Initializing Objects with Constructors

- An important difference between constructors and methods is that constructors cannot return values, so they cannot specify a return type (not even void).

- Normally, constructors are declared public.

- **Tip:** If you declare any constructors for a class, the Java compiler will not create a default constructor for the class.
More on Default Constructors

```java
public class GradeBook { // no constructor provided by the programmer
    private String courseName;
    public void setCourseName(String name) {
        courseName = name;
    }
    public String getCourseName() {
        return courseName;
    }
    public void displayMessage() {
        System.out.printf("Welcome to the grade book for\n%s!\n", getCourseName());
    }
}
```

Can we write the following statement to create a GradeBook object?

```java
GradeBook myGradeBook = new GradeBook();
```

Yes. Compiler will provide a default constructor with no parameters.
Can we write the following statement to create a GradeBook object?

```
GradeBook myGradeBook = new GradeBook();
```

**No.** Compiler will not provide a default constructor this time. The statement will cause a **compilation error**.
Case Study I: Pet Show

- A happy family has two pets: a poodle (贵宾犬) named “Fluffy”, a hound (猎犬) named “Alfred”.
- Suppose we want to write a Java program for a pet show: each dog makes a self introduction.

"Hello, my name is Fluffy. I am a poodle."

"Hello, my name is Alfred. I am a hound."
Program Design

- **Observation 1:** The two pets are both dogs. So we can design a `Dog` class to represent them.

```java
class Dog {
}
```
Observation 2: The two pets have their own names and belong to different breeds (品种). We can define two instance variables to represent such information.

```java
public class Dog {
    private String name;
    private String breed;
}
```
In order to create instances of the Dog class, we need to define a constructor. Since each dog has two attributes, we can define a constructor that takes two arguments.

```java
public class Dog {
    private String name;
    private String breed;
    public Dog(String name, String breed) {
        this.name = name;
        this.breed = breed;
    }
}
```
In order to create instances of the `Dog` class, we need to define a constructor. Since each dog has two attributes, we can define a constructor that takes two arguments.

```java
public class Dog {
    private String name;
    private String breed;
    public Dog(String name, String breed) {
        this.name = name;
        this.breed = breed;
    }
}
```

The passed arguments will be used to initialize the attributes.
In order to create instances of the Dog class, we need to define a constructor. Since each dog has two attributes, we can define a constructor that takes two arguments.

```java
public class Dog {
    private String name;
    private String breed;
    public Dog(String name, String breed) {
        this.name = name;
        this.breed = breed;
    }
}
```

The keyword “this” points to the current object. Helps differentiate the method parameters (local variables) and the instance variables.
In order to create instances of the Dog class, we need to define a constructor. Since each dog has two attributes, we can define a constructor that takes two arguments.

```java
public class Dog {
    private String name;
    private String breed;
    public Dog(String dogName, String dogBreed) {
        name = dogName;
        breed = dogBreed;
    }
}
```

“this” is not needed if the parameters and instance variables have different names (no ambiguity).
The dogs have the ability of making self introductions.

```java
public class Dog {
    private String name;
    private String breed;
    public Dog(String name, String breed) {
        this.name = name;
        this.breed = breed;
    }
    public void selfIntro() {
        System.out.printf("My name is %s. I am a %s.\n", name, breed);
    }
}
```
Finally, we implement the PetShow program with a main method.

```java
public class PetShow {
    public static void main(String[] args) {
        Dog dog1 = new Dog("Fluffy", "poodle");
        Dog dog2 = new Dog("Alfred", "hound");
        dog1.selfIntro();
        dog2.selfIntro();
    }
}
```

Invoke the constructor to create two dog objects.
Finally, we implement the `PetShow` program with a `main` method.

```java
public class PetShow {
    public static void main(String[] args) {
        Dog dog1 = new Dog("Fluffy", "poodle");
        Dog dog2 = new Dog("Alfred", "hound");
        dog1.selfIntro();
        dog2.selfIntro();
    }
}
```

Invoke methods on the two objects.

Object references (or names) are needed to invoke instance methods.
Case Study II: Account Balances

- Suppose we are asked to design a Java program for managing bank accounts.
- For simplicity, we assume that the bank only provides two types of services: (1) adding money to an account, (2) checking the balance of an account.
- The key task is to define an Account class.
// Account class with a constructor to validate and
// initialize instance variable balance of type double

public class Account {

    // instance variable that stores the balance
    private double balance;

    // constructor
    public Account(double initialBalance) {
        // if initialBalance is not greater than 0.0
        // balance is initialized to the default value 0.0
        if(initialBalance > 0.0) balance = initialBalance;
    }

    // add an amount to the account
    public void credit(double amount) {
        balance += amount;
    }

    // return the account balance
    public double getBalance() {
        return balance;
    }
}

Validating Constructor Arguments

- It’s common for users to open an account to deposit money immediately, so the constructor receives a parameter `initialBalance` of type `double` that represents the initial balance.
  - The constructor ensures that `initialBalance` is greater than 0.0
  - If so, `initialBalance`’s value is assigned to instance variable `balance`.
  - Otherwise, `balance` remains to be 0.0 (its default initial value).

```java
// constructor
public Account(double initialBalance) {
    if (initialBalance > 0.0) balance = initialBalance;
}
```
Case Study II: Account Balances

- We further define a class `AccountTest` that creates and manipulates two `Account` objects.
import java.util.Scanner;

public class AccountTest {
    public static void main(String[] args) {
        Account account1 = new Account(50.00);
        Account account2 = new Account(-7.53);

        // display initial balance of each object
        System.out.printf("account1 balance: $%.2f\n", account1.getBalance());
        System.out.printf("account2 balance: $%.2f\n\n", account2.getBalance());

        Scanner input = new Scanner(System.in);
        double depositAmount; // deposit amount read from user
System.out.print("Enter deposit amount for account1: ");

depositAmount = input.nextDouble();

System.out.printf("\nadding %.2f to account1 balance\n\n", depositAmount);

account1.credit(depositAmount); // add to account1 balance

// display balances
System.out.printf("account1 balance: $%.2f\n", account1.getBalance());

System.out.printf("account2 balance: $%.2f\n\n", account2.getBalance());
System.out.print("Enter deposit amount for account2: ");

    depositAmount = input.nextDouble();
System.out.printf("\nadding \%.2f to account2 balance\n\n",
    depositAmount);
account2.credit(depositAmount); // add to account2 balance

//display balances
System.out.printf("account1 balance: $\%.2f\n", 
    account1.getBalance());
System.out.printf("account2 balance: $\%.2f\n\n", 
    account2.getBalance());
input.close();
}
account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 25.53

adding 25.53 to account1 balance

account1 balance: $75.53
account2 balance: $0.00

Enter deposit amount for account2: 123.45

adding 123.45 to account2 balance

account1 balance: $75.53
account2 balance: $123.45
Java types are divided into two categories: **primitive types** and **reference types**.

- Primitive types are the basic types of data
  - byte, short, int, long, float, double, boolean, char
  - A primitive-type variable can store one value of its declared type

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Default value</th>
<th>Size</th>
<th>Example code</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Truth value</td>
<td>false</td>
<td>1 bit</td>
<td>boolean b = false;</td>
</tr>
<tr>
<td>char</td>
<td>Unicode character</td>
<td>\u0000</td>
<td>16 bits</td>
<td>char c = 'z';</td>
</tr>
</tbody>
</table>
All non-primitive types are reference types, including instantiable classes and arrays (an array is a container object that holds a fixed number of values of a single type)

- Scanner, String, String[], int[]

Reference-type variables store the memory locations of objects

- Dog dog1 = new Dog("Fluffy", "Poodle");
- Such a variable is said to refer to an object in the program. Objects that are referenced may each contain instance variables of primitive or reference types.
Reference-type variables, if not explicitly initialized, are initialized by default to the value \texttt{null} (reference to nothing).

To call methods of an object, you need to use the reference (must be non-null) to the object: \texttt{dog1.selfIntro();}

Primitive-type variables (e.g., \texttt{int} variables) do not refer to objects, so such variables cannot be used to call methods.