Writing Classes

Chapter 4

Objectives

- You will be able to write and use your own class definitions in your Java programs.

Writing Classes

- In a Java program, everything that we write is in a *class*.

Key Concept

  - A class is a container for data definitions and code that acts on that data.

- So far all our programs have consisted of only one class
  - in addition to predefined classes in the Java class library.

- As we write larger programs, putting all of our code into a single class would make it hard to understand and maintain.

Classes

- A real world program might consist of *many* classes.
  - Written and maintained by *many* programmers.
- A programmer who works on a class must understand that class in detail.
  - It is not necessary to understand the inner workings of other classes.
- Consider an automobile.
  - A mechanic can work on the brakes without having to understand the inner details of the engine or the transmission.

Why Classes?

- The purpose of classes is to divide a program into more of less independent parts.
  - Each with well defined functionality.
  - Each makes sense without having to understand the internal details of the others.
  - Each has an interface that specifies what the rest of the program can do with that class.
    - *Services* that the class provides.

- We say the the class *encapsulates* its internal details.

Key Concept

Why Classes?

- Classes can potentially be reusable components.
  - Consider the predefined classes that we have used in the Java class library.
    - String, Scanner, File, PrintWriter, etc.

- We can create our own classes to be used in multiple programs.
  - Specific to an application area.

Classes and Objects

What is a class?

- A class definition is a blueprint for a chunk of executable code and data.
  - In general, not directly exectable.
- Contains *declarations* for variables.
  - Called member variables, or *fields*.
  - In general, not *actual* variables, just declarations
- Named chunks of code, called *methods*.
  - Example: The length method of class String.
  - Invoked by writing the method name followed by parentheses.

Classes and Objects

- To use a class, we typically have to create an *object* based on the class.
  - An *instance* of the class.
  - We *instantiate* a class to create an object.
  - Example:
    - `String my_string = new String("The quick brown fox");`

Object Class

Objects

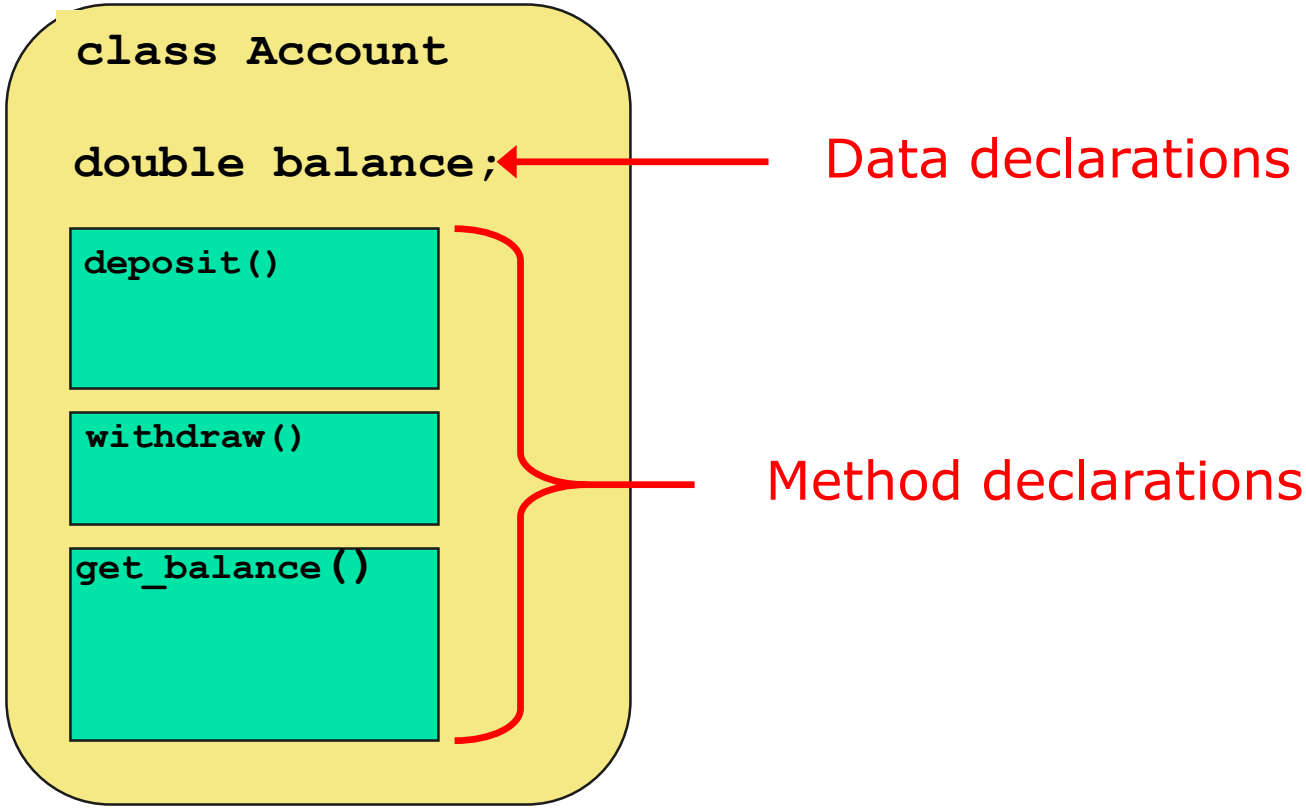
- An object has *state* and *behavior*.
- The values of its data define the state of an object created from the class.
  - Each object has its own data.
  - Can potentially change over time.
- The the methods define the behavior of the object.
  - Set when the class is compiled.
  - Same for all objects of a class.

Example: Bank Account

- We can represent a bank account in software by designing a class called **`Account`** that models the required data and operations.
  - The class serves as the *blueprint* for Account objects.
- We can instantiate as many Account objects as we need for any particular program.

Example: Bank Account

A class definition contains data declarations and method declarations.



A Mistake in the Book

- Unfortunately, the Account example in the textbook is seriously flawed.
  - Uses the double class for bank account balances.

- This is bad programming.
- The double class is subject to representation error, and round off in calculation results.
  - Not appropriate for dollars and cents.

What Oracle Says

float: The `float` data type is a single-precision 32-bit IEEE 754 floating point. … This data type should never be used for precise values, such as currency. For that, you will need to use the java.math.BigDecimal class instead.

double: The `double` data type is a double-precision 64-bit IEEE 754 floating point … As mentioned above, this data type should never be used for precise values, such as currency.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Unfortunately, use of the BigDecimal class is complicated. Rather than try to repair the Account example, we will simply skip it for now.

Constructor

Key Concept

- A *constructor* is a special method that is used to set up an object when it is initially created.
  - Initializes member variables.
- A constructor has the same name as the class.
- The programmer does not have to define a constructor for a class: Each class has a *default constructor* that accepts no parameters.
  - All member variables will have default values.

Example: Class Dog

- Information about a dog.
  - Perhaps for a vetenarian's office.

  - Name
  - Breed
  - Age

New Yorker, July 5, 1993

Nobody knows you're a dog.

- Unfortunately no longer true.

Class Dog

```
//*********************************************
//  Dog.java
//
//  Represents a dog.
//
//*********************************************

public class Dog
{
    // Instance variables
    private String name;
    private String breed;
    private int age;
```

Class Dog: Constructor

```
//-----------------------------------------------------
// Constructor - sets up a dog object by initializing
// the name, the breed, and the age.
//-----------------------------------------------------
public Dog(String newName, String newBreed, int newAge)
{
    name = newName;
    breed = newBreed;
    age = newAge;
}
```

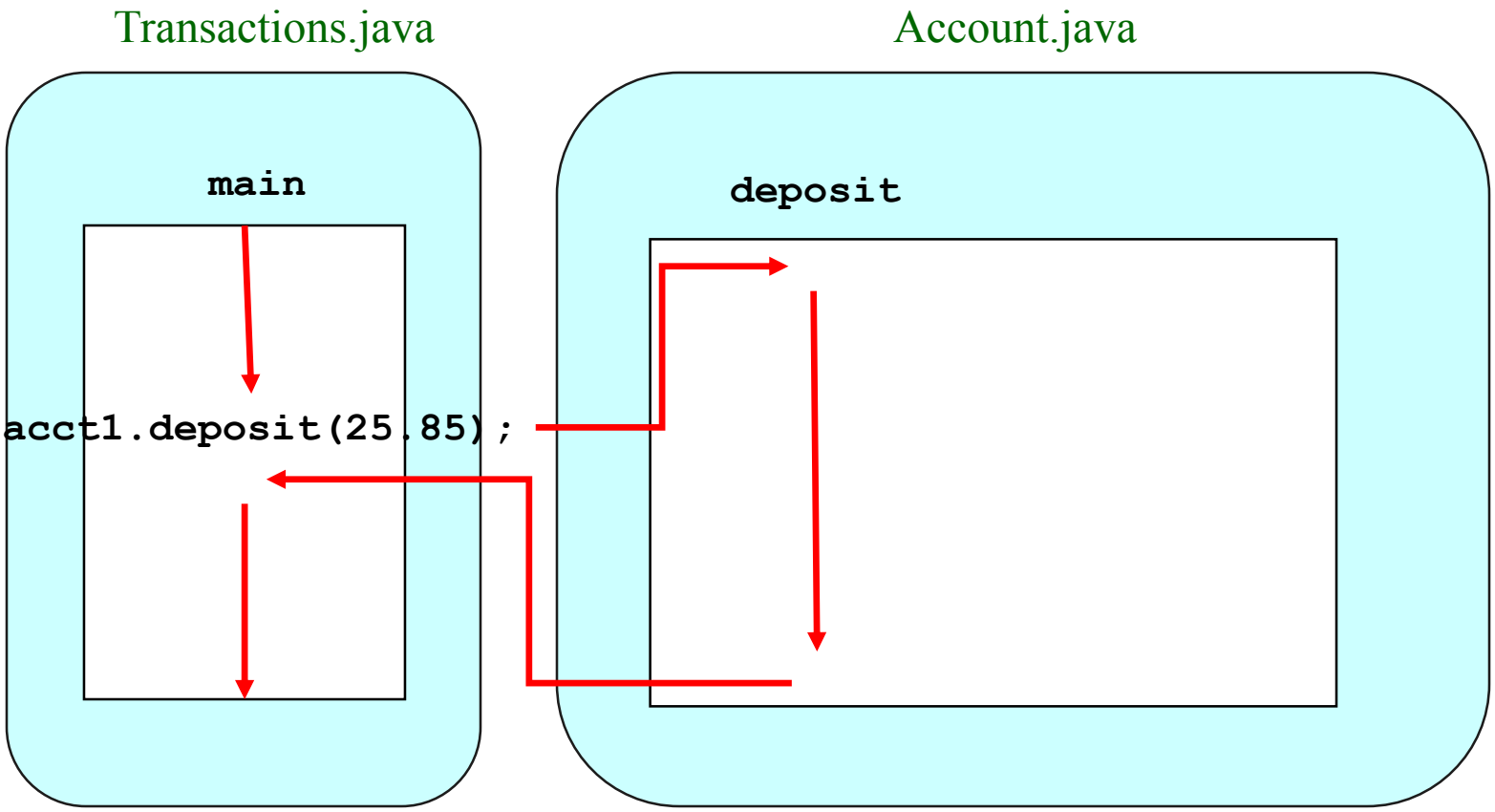
Methods

Method Declarations

- A *method definition* specifies the code that will be executed when the method is invoked (called).
- When a method is invoked, the flow of control jumps to the first executable statement of the method and continues through its code.
- When complete, the flow returns to the place where the method was called and continues.
- The invocation may or may not return a value, depending on how the method is defined.

Method Control Flow

The called method is often part of another class or object.



Method Header

A method definition begins with a *method header.*

```
int add(int num1, int num2)
```

return type

method name

parameter list

A method can have inputs, called parameters.

The parameter list specifies the type and name of each parameter.

The items in the parameter list are called *formal parameters*.

The Return Type

- The *return type* of a method indicates the *type of value* that the method sends back to the calling location.
- A method that does not return a value has **`void`** as its return type.
- Example:

```
public static void main(String[] args)
```

Method Body

The method header is followed by the *method body.*

```
int add(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

Note alignment and indentation.

The return Statement

- A *return statement* determines the value that is returned to the caller:

```
return expression;
```

- Its expression must conform to the return type

```
int add(int num1, int num2)
{
    int sum = num1 + num2;

    return sum;
}
```

The return expression must be consistent with the return type

Parameters

- Formal Parameters:
  - The names of the parameters in the header of the method definition.
- Actual Parameters:
  - The values passed into a method by a call.

Parameters

- When a method is called, the values of the *actual parameters* in the invocation are copied into the *formal parameters* in the method header

```
int num = obj.add (24,  192);
```

actual parameters

```
int add(int num1, int num2)
{
    int sum = num1 + num2;

    return sum;
}
```

formal parameters

Example

- Dog program continued

- Add a method:
  - A method *ageInPersonYears* that takes no parameter.
  - The method should compute and return the age of the dog in “person years”.
    - Seven times the dog’s age.

Method ageInPersonYears

```
//-------------------------------------------------------------
// Method ageInPersonYears that takes no parameter. The method
// should compute and return the age of the dog in person years
// (seven times the dog's age).
//-------------------------------------------------------------

public int ageInPersonYears()
{
    int personAge = age*7;
    return personAge;
}
```

The *toString* Method

- All classes that can be instantiated should define a **`toString`** method.
- The **`toString`** method returns a character string that represents the object.
- It is called automatically when an object is concatenated to a string or when it is passed to the **`print/println`** method.

Method toString

```
//-----------------------------------------------------
// Returns a string representation of a dog.
//-----------------------------------------------------
public String toString()
{
    return name + "\t"+  breed + "\t" + age +
        "  age in person years:" + ageInPersonYears();
}
```

Driver Programs

- A *driver program* is a class with a main method that can be used to exercise the methods of a class.
  - A test program for a class definition.
- Let's write a test driver for class Dog.

Class Kennel

```
//*************************************************
//  Kennel.java
//     A test driver for class Dog.
//*************************************************
import java.util.Scanner;

public class Kennel
{
//-----------------------------------------------
// Creates a dog object and then calls the
// ageInPersonyears method.
//-----------------------------------------------
public static void main (String[] args)
{
    String dogName;
    String dogBreed;
    int dogAge;
```

Class Kennel

```
    Scanner scan = new Scanner(System.in);
    System.out.print ("Enter the name of the dog: ");
    dogName = scan.nextLine();

    System.out.print ("Enter the breed of the dog: ");
    dogBreed = scan.nextLine();

    System.out.print ("Enter the age of the dog: ");
    dogAge = scan.nextInt();

    // Create a dog object
    Dog dog1 = new Dog(dogName, dogBreed, dogAge);

    System.out.println("Its age in person years is " +
        dog1.ageInPersonYears() );
    System.out.println();

    // Print summary for the dog
    System.out.println(dog1);
}
```

Compile and Run

```
Command Prompt
C:\test>
C:\test>javac Dog.java

C:\test>javac Kennel.java

C:\test>java Kennel
Enter the name of the dog: Snoopy
Enter the breed of the dog: Beagle
Enter the age of the dog: 8
Its age in person years is 56

Snoopy  Beagle  8  age in person years:56

C:\test>
```

Readings and Assignments

- Reading: Chapter 4.1, 4.2
- Lab Assignment: Project 11

- Exercises and Programming Projects:
  - After Chapter Exercises
    - EX 4.1, 4.6, 4.10
  - Programming Projects
    - PP 4.1, 4.7