Chapter 3

Introduction to Classes, Objects, Methods and Strings
### 3.2 Declaring a Class with a Method and Instantiating an Object of a Class

Each class declaration that begins with keyword public must be stored in a file having the same name as the class and ending with the (.java) extension.

In the next example, we declare class GradeBook. To use it, we have to create an object of this class and call its method.

```java
4 public class GradeBook
5 {
6     // display a welcome message to the GradeBook user
7     public void displayMessage()
8     {
9         System.out.println( "Welcome to the Grade Book!" );
10     } // end method displayMessage
11 } // end class GradeBook
```

- The class declaration begins in line 4, for now we will simply declare every class public, which indicates that the method is available to the public and so, it can be called from methods of other classes.
- Every class declaration contains keyword class followed immediately by the class’s name.
- Recall that main is a special method that’s always called automatically to execute the application. Most methods are not called automatically; you must call method displayMessage explicitly when you want to.
- The method return type specifies the type of data the method returns to its called after performing its task.
- The parentheses after the method name indicates that this is a method, and empty parentheses indicate that this method does not require additional information. Line 7 is commonly referred to as the method header.

Now we will use class GradeBook in an application.

```java
4 public class GradeBookTest
5 {
6     // main method begins program execution
7     public static void main( String[] args )
8     {
9         // create a GradeBook object and assign it to myGradeBook
10        GradeBook myGradeBook = new GradeBook();
11    
12        // call myGradeBook's displayMessage method
13        myGradeBook.displayMessage();
14     } // end main
15 } // end class GradeBookTest
```

- A class that contains method main begins the execution of the application, and so class GradeBook is not an application because it does not contain main.

- If you execute a class without a main method in it, an error will occur.

- In line 7, the word static indicates that main is a static method. A static method is special because you can call it without first creating an object of the class in which the method is declared. This is a key part of enabling the JVM to locate and call method main to begin the execution.
- You cannot call a method that belongs to another class until you create an object of that class, as shown in line 10. Each new class you create becomes a new type that can be used to declare variables and create objects.
- In line 10, variable myGradeBook is initialized with the result of the class instance creation expression new GradeBook(). Keyword new creates a new object of the type specified.
- Any method with the same name of the class in which it was declared in is a called constructor, which is a special method that is called only at the time an object is created to initialize the object’s data.
- Line 13 uses the created object myGradeBook to call the method displayMessage to perform its task and print the message.
- The empty parentheses indicates that no arguments are sent in the method call, after displayMessage perform its task, main continues executing its next statement (line 14).

### 3.3 Declaring a method with a parameter

A method can require one or more parameters that represent additional information it needs to perform its task. Each parameter must specify a type and a variable name. The parameter list may contain any number of parameters, including none at all.

On the other hand, a method call supplies values, called arguments, for each of the method’s parameters.
We now declare class GradeBook with a displayMessage method that displays the course name as part of the welcome message.

```java
4 public class GradeBook
5 {
6     // display a welcome message to the GradeBook user
7     public void displayMessage( String courseName )
8     {
9         System.out.printf( "Welcome to the grade book for\ns\n", courseName );
10    } // end method displayMessage
11 } // end class GradeBook
```

```java
4 import java.util.Scanner; // program uses Scanner
5 public class GradeBookTest
6 {
7     // main method begins program execution
8     public static void main( String[] args )
9     {
10        // create Scanner to obtain input from command window
11        Scanner input = new Scanner( System.in );
12 
13        // create a GradeBook object and assign it to myGradeBook
14        GradeBook myGradeBook = new GradeBook();
15 
16        // prompt for and input course name
17        System.out.println( "Please enter the course name:" );
18        String nameOfCourse = input.nextLine(); // read a line of text
19        System.out.println(); // outputs a blank line
20        
21        // call myGradeBook's displayMessage method
22        // and pass nameOfCourse as an argument
23        myGradeBook.displayMessage( nameOfCourse );
24    } // end main
25 } // end class GradeBookTest
```

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
- In line 7 in class GradeBook, the parameter list declares one parameter indicating that the method requires a String to perform its task. When the method is called, the argument value in the call is assigned to the corresponding parameter (courseName) in the method header.

- Line 19 in class GradeBookTest reads the name from the user and assigns it to the nameOfCourse variable, using Scanner method nextLine to perform the input.

- Method nextLine reads characters typed by the user until it encounters the newline character, then returns a String containing the characters up to, but not including, the newline.

- In line 24, the variable nameOfCourse in parentheses is the argument that is passed to method displayMessage.

- The parameter variable’s name (courseName) can be the same or different from the argument variable’s name (nameOfCourse).

- The number of arguments in a method call must match the number of parameters in the parameter list of the method’s declaration. Also, the argument types in the method call must be “consistent with” the types of the corresponding parameters in the method’s declaration.

**Note:** Class String is the same as Class System, they are both in package java.lang, which is implicitly imported into every java program. But Class GradeBook is not in that package, so why didn’t we import it before using it? There’s a special relationship between classes that are compiled on the same directory on disk, like classes GradeBook and GradeBookTest. By default, such classes are considered to be in the same package. Classes in the same package are implicitly imported into the source code files of other classes in the same package.
3.4 Instance Variable, set Methods and get Methods

Variables declared in the body of a particular method are known as local variables and can be used only in that method. When that method terminates, the values of its local variables are lost.

An object of a class can have attributes that are carried with it; these attributes are represented as variables in a class declaration. These variables are called fields and are declared inside a class declaration but outside the bodies of the class’s method declarations.

```java
5 public class GradeBook
6 {
7     private String courseName;  // course name for this GradeBook
8 
9     // method to set the course name
10    public void setCourseName( String name )
11    {
12        courseName = name;  // store the course name
13    } // end method setCourseName
14
15    // method to retrieve the course name
16    public String getCourseName()
17    {
18        return courseName;
19    } // end method getCourseName
20
21    // display a welcome message to the GradeBook user
22    public void displayMessage()
23    {
24        // calls getCourseName to get the name of
25        // the course this GradeBook represents
26        System.out.printf("Welcome to the grade book for\n%s!\n", getCourseName() );
27    } // end method displayMessage
28 } // end class GradeBook
```
- Class GradeBook maintains the course name as an instance variable so that it can be used or modified at any time during an application’s execution.
- Line 7 is a declaration of an instance variable. Every instance of class GradeBook contains one copy of each instance variable. If there are two GradeBook objects, each object would have its own copy of variable courseName.
- A benefit of making courseName an instance variable is that all the methods of the class can manipulate any instance variables that appear in that class.
- Variables or methods declared with access modifier private are accessible only to methods of the class in which they’re declared. Thus, variable courseName can only be used in methods setCourseName, getCourseName and displayMessage of every object of class Gradebook.

- Declaring instance variables with access modifier private is known as data hiding or information hiding. When a program creates (instantiates) an object of class GradeBook, variable courseName is encapsulated (hidden) in the object and can be accessed only by methods of the object’s class. This prevents courseName from being modified accidentally by a class in another part of the program.
- You should always use access modifiers. Generally, instance variables should be declared private and methods should be public.
- Line 12 assigns name to instance variable courseName. And so method setCourseName was used to access or modify the object’s instance variable. Similarly, method getCourseName was used to get the value of the object’s instance variable.
- Note that both previous methods used courseName even though it was not declared in any of them because courseName is an instance variable of the class.

- One method of a class (displayMessage in this case) can call another method of the same class by using just the method name.

```java
5  public class GradeBookTest
6  {
7      // main method begins program execution
8      public static void main( String[] args )
9      {
10         // create Scanner to obtain input from command window
11         Scanner input = new Scanner( System.in );
12
13         // create a GradeBook object and assign it to myGradeBook
14         GradeBook myGradeBook = new GradeBook();
15
16         // display initial value of courseName
17         System.out.printf( "Initial course name is: %s\n\n", 
18                 myGradeBook.getCourseName() );
19
20         // prompt for and read course name
21         System.out.println( "Please enter the course name:" );
22         String theName = input.nextLine(); // read a line of text
23         myGradeBook.setCourseName( theName ); // set the course name
24         System.out.println(); // outputs a blank line
25
26         // display welcome message after specifying course name
27         myGradeBook.displayMessage();
28      } // end main
29  } // end class GradeBookTest
```

Initial course name is: null
Please enter the course name:
CS101 Introduction to Java Programming
Welcome to the grade book for
CS101 Introduction to Java Programming!
- The first line of the output shows the name “null” because unlike local variables which are not automatically initialized, every field has a default initial value which provided by Java when you do not specify the field’s initial value. Thus, fields are not required to be explicitly initialized before they’re used in a program.

- As we have seen, a class’s private fields can be manipulated only by the class’s methods. So a client of an object – that is, any class that calls the object’s methods – calls the class’s public methods to manipulate the private fields of an object of the class.

### 3.5 Primitive Types vs. Reference Types

The primitive types are boolean, byte, char, short, int, long, float and double. All nonprimitive types are reference types, so classes, which specify the types of objects, are reference types.

A primitive type variable can store exactly one value of its declared type at a time. They are initialized by default, variables of types byte, char, short, int, long, float and double are initialized to 0, and variables of type boolean are initialized to false. Recall that local variables are not initialized by default.

Objects that are referenced may each contain many instance variables. Line 14 of the previous program creates an object of class GradeBook, and the variable myGradeBook contains a reference to that GradeBook object. Instance variables that are of reference types are initialized to null.
3.6 Initializing Objects with Constructors

Each class you declare can provide a special method called a constructor that can be used to initialize an object of a class when the object is created. Java requires a constructor call for every object that’s created. Keyword new requests memory from the system to store an object, then calls the corresponding class’s constructor to initialize the object. The call is indicated by the parentheses after the class name. A constructor must have the same name as the class.

By default, 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 to their default values.

```java
public class GradeBook {
    private String courseName; // course name for this GradeBook

    // constructor initializes courseName with String argument
    public GradeBook( String name ) { // constructor name is class name
        courseName = name; // initializes courseName
    } // end constructor

    // method to set the course name
    public void setCourseName( String name ) {
        courseName = name; // store the course name
    } // end method setCourseName
}
```
May 2014

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

        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

- The class instance creation expression in lines 11-12 of class GradeBook returns a reference to the new object, which is assigned to the variable gradeBook1.
- The output confirms that each GradeBook maintains its own copy of instance variable CourseName.
- An important difference between constructors and methods is that constructors cannot return values, so they cannot specify a return type (not even void), and they are normally declared public.
- If you declare even one constructor for a class, the Java compiler will not create a default one for that class. Thus, we can no longer create a GradeBook object with the default constructor as we did before.

### 3.7 Floating-Point Numbers and Type double

We will declare a class named Account that maintains the balance of a bank account.

```java
public class Account
{
    private double balance; // instance variable that stores the balance

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

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

    // return the account balance
    public double getBalance()
    {
        return balance; // gives the value of balance to the calling method
    } // end method getBalance

} // end class Account
```
import java.util.Scanner;

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

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

        // create Scanner to obtain input from command window
        Scanner input = new Scanner(System.in);
        double depositAmount; // deposit amount read from user
        System.out.printf("Enter deposit amount for account1: "); // prompt
        depositAmount = input.nextDouble(); // obtain user input
        System.out.printf("\nadding %.2f to account1 balance\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", account2.getBalance());

        System.out.printf("Enter deposit amount for account2: "); // prompt
        depositAmount = input.nextDouble(); // obtain user input
        System.out.printf("\nadding %.2f to account2 balance\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", account2.getBalance());
    }
} // end class AccountTest
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

- In lines 14 and 15, if the value is not greater than zero, balance remains at 0.0, its initial value.

**Note:** So far, we have used three methods of class Scanner to read an input. 
nextInt, which reads an integer. nextLine which reads a string and 
nextDouble which can read a float or a double. Scanner also provides a 
similar method – next – that reads individual words.
Exercise 3.13 Modify class Account to provide a method called debit that withdraws money from an account. Ensure that the debit amount does not exceed the accounts balance. If it does, the balance should be left unchanged and the method should print a message indicating “debit amount exceeded amount balance”. Modify class AccountTest to test method debit.

```java
public class Account {
    private double balance;

    public Account(double initialBalance) {
        if (initialBalance > 0.0)
            balance = initialBalance;
    }

    public void credit(double amount) {
        balance = balance + amount;
    }

    public double getBalance() {
        return balance;
    }

    public void debit(double amount) {
        if (balance >= amount)
            balance = balance - amount;
        if (balance < amount)
            System.out.println("Debit amount exceeded amount balance");
    }
}
```
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 );

        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, debitAmount;
        System.out.println( "Enter deposit amount for account1: " );
        depositAmount = input.nextDouble();
        System.out.printf( "adding %.2f to account1 balance\n\n", depositAmount );
        account1.credit(depositAmount );

        System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
        System.out.printf( "account2 balance: $%.2f\n\n", account2.getBalance() );

        System.out.println( "Enter deposit amount for account2: " );
        depositAmount = input.nextDouble();
        System.out.printf( "adding %.2f to account2 balance\n\n", depositAmount );
        account2.credit(depositAmount );

        System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
        System.out.printf( "account2 balance: $%.2f\n", account2.getBalance() );
    }
}

May 2014
```java
debitAmount = input.nextDouble();
account1.debit(debitAmount);

System.out.printf("\naccount1 balance: $%.2f\n", account1.getBalance() );
System.out.printf("account2 balance: $%.2f\n\n", account2.getBalance() );

```

**Sample run #1**

account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 10

adding 10.00 to account1 balance

account1 balance: $60.00
account2 balance: $0.00

Enter deposit amount for account2: 20

adding 20.00 to account2 balance

account1 balance: $60.00
account2 balance: $20.00

Enter deposit amount for account1: 30

account1 balance: $30.00
account2 balance: $20.00
Sample run #2

account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1:
50

adding 50.00 to account1 balance

account1 balance: $100.00
account2 balance: $0.00

Enter deposit amount for account2: 10

adding 10.00 to account2 balance

account1 balance: $100.00
account2 balance: $10.00

Enter deposit amount for account1: 120
Debit amount exceeded amount balance

account1 balance: $100.00
account2 balance: $10.00
Exercise 3.13 Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables—a part number (type String), a part description (type String), a quantity of the item being purchased (type int) and a price per item (double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named InvoiceTest that demonstrates class Invoice’s capabilities.

```java
public class Invoice {
    private String number, description;
    private int quantity;
    private double price;

    public Invoice(String n, String d, int q, double p) {
        number = n;
        description = d;
        if (q > 0) quantity = q;
        if (p > 0) price = p;
    }

    public void setNumber(String n) {
        number = n;
    }

    public double getInvoiceAmount() {
        return quantity * price;
    }
}
```
```java
public void setDescription ( String d) {
    description = d;
}
public void setQuantity ( int q) {
    if ( q > 0 )
        quantity = q;
    if ( q <= 0 )
        quantity = 0;
}
public void setPrice ( double p) {
    if ( p > 0 )
        price = p;
    if ( p <= 0 )
        price = 0;
}
public String getNumber() {
    return number;
}
public String getDescription() {
    return description;
}
public int getQuantity() {
    return quantity;
}
public double getPrice() {
    return price;
}
public double getInvoiceAmount(){
    return (quantity * price);
}
```
import java.util.Scanner;

public class InvoiceTest
{

public static void main( String[] args )
{

Scanner input = new Scanner(System.in);

System.out.print("Enter the part number: ");
String myNumber = input.nextLine();
System.out.print("Enter the description of the invoice: ");
String myDescription = input.nextLine();
System.out.print("Enter the quantity of the item: ");
int myQuantity = input.nextInt();
System.out.print("Enter the price of the item: ");
double myPrice = input.nextDouble();

Invoice myInvoice = new Invoice(myNumber, myDescription, myQuantity, myPrice);

System.out.printf("\n
The number of the invoice is: %s\n", myInvoice.getNumber());
System.out.printf("The description of the invoice is: %s\n", myInvoice.getDescription());
System.out.printf("The quantity of the items is: %d\n", myInvoice.getQuantity());
System.out.printf("The price of the item is: %s\n", myInvoice.getPrice());

System.out.print("\nEnter the new number of the part: ");
myNumber = input.nextLine();
System.out.print("Enter the new description of the invoice: ");
myDescription = input.nextLine();
System.out.print("Enter the new quantity of the item: ");
myQuantity = input.nextInt();
System.out.print("Enter the new price of the item: ");
myPrice = input.nextDouble();

System.out.print("\nEnter the new number of the invoice: ");
myInvoice = new Invoice(myNumber, myDescription, myQuantity, myPrice);

System.out.printf("\nThe number of the invoice is: %s\n", myInvoice.getNumber());
System.out.printf("The description of the invoice is: %s\n", myInvoice.getDescription());
System.out.printf("The quantity of the items is: %d\n", myInvoice.getQuantity());
System.out.printf("The price of the item is: %s\n", myInvoice.getPrice());

System.out.println("\nEnter the new number of the part: ");
myNumber = input.nextLine();
System.out.println("Enter the new description of the invoice: ");
myDescription = input.nextLine();
System.out.println("Enter the new quantity of the item: ");
myQuantity = input.nextInt();
System.out.println("Enter the new price of the item: ");
myPrice = input.nextDouble();
myQuantity = input.nextInt();
System.out.print("Enter the new price of the item: ");
myPrice = input.nextDouble();

myInvoice.setNumber(myNumber);
myInvoice.setDescription(myDescription);
myInvoice.setQuantity(myQuantity);
myInvoice.setPrice(myPrice);

System.out.printf("\nThe new number of the part is: %s\n",myInvoice.getNumber());
System.out.printf("The new description of the invoice is: %s\n",myInvoice.getDescription());
System.out.printf("The new quantity of the items is: %d\n",myInvoice.getQuantity());
System.out.printf("The new price of the item is: %s\n",myInvoice.getPrice());
System.out.printf("The total price of the invoice is: %f\n",myInvoice.getInvoiceAmount());

}
Exercise 3.14 Create a class called Employee that includes three instance variables, a first name (type String), a last name (type String) and a monthly salary (double). Provide a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, do not set its value. Write a test application named EmployeeTest that demonstrates class Employee’s capabilities. Create two Employee objects and display each object’s yearly salary. Then give each Employee a 10% raise and display each Employee’s yearly salary again.
النوتات مجانية للنفع العام فيرجى المساهمة بالإبلاغ عن أي خطاً أو ملاحظات تراها ضرورية برسالة نصية 06244499 أو بالبريد الإلكتروني

أو


eng-hs.com, eng-hs.net

شرح ومسائل محلولة مجانية بالموقعين

info@eng-hs.com 94444

م. حمادة شعبان 260
Exercise 3.15 Create a class called Date that includes three instance variables, a month (type int), a day (type int) and a year (type int). Provide a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date’s capabilities.
النوتات مجانية للنفع العام فيرجى المساهمة بالإبلاغ عن أي خطأ أو ملاحظة ضرورية برسالة نصية 062 4444 9 أو بالبريد الإلكتروني


eng-hs.com, eng-hs.net  شرح ومسائل محلولة مجانية بالموقعين  info@eng-hs.com 9 4444

م. حمادة شعبان 260 م