

- [Chapter 1](#)
- [Chapter 2](#)
- [Chapter 3](#)
- [Chapter 4](#)
- [Chapter 5](#)
- [Chapter 6](#)
- [Chapter 7](#)
- [Chapter 8](#)
- [Chapter 9](#)
- [Chapter 10](#)
- [Chapter 11](#)
- [Chapter 12](#)
- [Chapter 13](#)
- [Chapter 14](#)
- [Chapter 15](#)
- [Chapter 16](#)
- [Chapter 17](#)
- [Chapter 18](#)
- [Chapter 19](#)
- [Chapter 20](#)
- [Chapter 21](#)
- [Chapter 22](#)
- [Chapter 23](#)
- [Chapter 24](#)
- [Chapter 25](#)

Chapter 1

R1.1

A well-designed computer program is easy to use without any special knowledge. For example, most people can learn to navigate webpages with only a few minutes of practice. On the other hand, programming a computer requires special knowledge about what the computer can fundamentally do and how you would communicate with it through a programming language.

R1.2

Typically program code is stored on a hard disk, CD/DVD disc, or in some other central location across a network. User data is often more likely to be stored on a local hard disk, although it can also be stored on a network or even a CD/DVD for backup storage.

R1.3

The monitor, speakers, and printer serve as the primary devices to give information to the user. The keyboard and mouse are the primary devices that take user input.

R1.4

It's very likely your cell phone is a programmable computer. If you can take pictures, send email/text messages, and/or surf the web with your phone, it is a programmable computer. If your cell phone can only send and receive phone calls, it is probably a single-function device.

R1.5

One advantage of Java over machine code is that Java statements are independent of the machine (computer) they are being executed on; machine code statements differ from one type of machine to the next. Another advantage of Java is that it is much more readable and understandable (by humans) than machine code.

R1.6

a) Solutions here will vary based on user and IDE preference. On a UNIX-based system using the Eclipse IDE you may see a path like

```
/home/nancy/JFE/src
```

While on a Microsoft Windows machine you might find a directory like:

```
C:\Users\nancy\Documents\JFE\src
```

b) Again, solutions can vary. On Unix using Eclipse you might see:

```
/home/nancy/JFE/bin
```

A Microsoft Windows machine might be:

```
C:\Users\nancy\Documents\JFE\bin
```

c) The answer to this question is dependent on the type of operating system and version of Java. On a Unix based system using Java 1.6 you might find `rt.jar` here:

```
/usr/lib/jvm/java-6-sun-1.6.0.13/jre/lib/rt.jar
```

While on a Microsoft Windows platform you might find it here:

```
C:\Program Files\Java\jdk1.6.0_10\jre
```

R1.7

The program prints the following:

```
39 + 3
42
```

Java interprets the statement `"39 + 3"` as a string and thus prints out the literal characters `39 + 3`. Java interprets the second statement `39 + 3` as an operation between two numbers, so it first calculates the value `39 + 3 = 42` then prints out the result `42`.

R1.8

```
HelloWorld
```

Because there are no spaces after the `System.out.print("Hello");` the next line prints `World` directly after `Hello` is printed.

R1.9

Java interprets the comma in the `println` method to mean that two strings are passed to `println`. It's likely the programmer meant to do this:

```
System.out.print("Hello, World!");
```

R1.10

Compile-time errors:

```
public class HelloPrinter
{
    public static void main(String[] args)
    {
        System.outprint("Hello, World");
        /*      ^^ missing '.' */
    }
}

public class HelloPrinter
{
    public static void main(String[] args)
    {
        System.out.print("Hello, World);
        /*      ^^ missing '"' */
    }
}

public class HelloPrinter
{
    public static void main(String[] args)
    {
        System.out.print("Hello, World");
        /*      ^^ missing ":" */
    }
}
```

Run-time error:

```
public class HelloPrinter
{
    public static void main(String[] args)
    {
        System.out.print("Hello, Wrold");
    }
}
```

R1.11

Syntax errors are discovered by compiling your Java program; the Java compiler will report them directly. Logic errors are discovered by testing your program by running it and verifying the correct output; if the output is incorrect, you have a logic error.

R1.12

Start with the customer knowing how often they visit the cafeteria and the average price they pay for a meal.

Ask the cafeteria for cost of the discount card, the number of meals to be eaten to achieve the free meal, and the period of time in which all meals must be consumed.

If cost of the card is greater than the average cost of a customer meal
Deal is set to "no good"

Else
If qty of meals to be eaten is greater than qty of customers meals in the time period then
Deal is set to "no good"
Else
Deal is set to "good"

R1.13

Start with a year of value 0, a month of value 0, and a balance of \$10,000
Repeat the following while the balance is greater than 0
Multiply the balance by 1.005.
Subtract 500 from the balance.
Add one to month.
If the month is 12
Set month to 0.
Add one to year.
Year has the answer.

R1.14

If the starting balance is large enough and the interest rate large enough such that the first month's calculation of interest exceeds the monthly expenses, then you will never deplete your account and the algorithm will never terminate.

Modified algorithm:
Ask the user for balance, interest rate, and monthly expenses.
If balance x interest is greater than monthly expenses
Tell the user you're in good financial shape and quit.
Otherwise, start with a year of value 0, a month of value 0 and a balance of \$10,000
Repeat the following while the balance is greater than 0
Multiply the balance by (1 + (interest rate times .01 divided by 12)).
Subtract monthly expenses from the balance.
Add one to month
If the month is 12
Set month to 0.
Add one to year.
Year has the answer.

R1.15

Calculate the overall surface area of the house without windows and doors.

surfaceArea = (2 x width x height) + (2 x length x height)
- (number of windows x windowWidth x windowHeight)
- (number of doors x doorWidth x doorHeight)

R1.16

The computer cannot accurately predict what the price of gas will be on a daily basis (or know what day you will go to the gas station) and how many actual miles you will drive each year.

R1.17

Every day at 5 P.M. please do the following:

- 1) Insert the USB memory stick into the computer.
- 2) Create a new directory on the USB memory stick entitled "BACKUP-DATE" where "DATE" is replaced with the current date.
- 3) Copy the files from the "My Documents" folder on the computer to the folder you just created on the USB stick.
- 4) Double check to make sure the new directory contains the files you copied. If the folder is empty, something is wrong. It's possible you backed up to the wrong directory. Try it again and be careful which directory you copy into.
- 5) Once you verified the copy is complete, remove the USB stick and store it someplace safe.
- 6) Thank you!

R1.18

Get the orange juice from the refrigerator
Get the eggs from the refrigerator
Get the bacon from the refrigerator
Make the pancake batter
For each person eating breakfast:
Crack an egg on the griddle
Pour two pancakes on the griddle
Place two slices of bacon on the griddle
Pour a glass of orange juice
Dish up a plate of cooked food
Call the family to breakfast

R1.19

close enough = 0.001
a = value from user to be square rooted
first guess = a / 2
second guess = (first guess + (a / first guess)) / 2
repeat
first guess = second guess
second guess = (first guess + (a / first guess)) / 2
until (first guess - second guess) <= close enough

second guess holds the square root of a

Chapter 2

R2.1

Objects with the same behavior belong to the same class. A window lets in light while protecting a room from the outside wind and heat or cold. A water heater has completely different behavior. It heats water. They belong to different classes.

R2.2

When one calls a method, one is not concerned with how it does its job. As long as a light bulb illuminates a room, it doesn't matter to the occupant how the photons are produced.

R2.3

An object is an instance (an entity) that defined by a class. A class provides a definition which includes characteristics (data) and behavior (methods).

R2.4

Examples of String objects:
"Hello World"
"Schrödinger is the name of my cat."
"I LOVE programming"

Example of PrintStream object:
System.out

Sample Methods in the String class that are not in PrintStream class:

```
length()
toUpperCase()
```

Sample Method in the `PrintStream` class that is not in `String` class:

```
println()
```

R2.5

The public interface consists of all the methods we can apply to any of its objects. Essentially, it is the set of methods to interact with the class. The implementation is how the methods accomplish their tasks. The public interface is accessible to all; the implementation should be private.

R2.6

```
double price = 5.50;
String description = "Dress Socks";
```

R2.7

The value of `mystery` is equal to 0 after the statements are executed. In the first statement (line 1), `mystery` is initialized to a value of 1. In the assignment statement on line 2, `mystery` is set to -1. Finally, `mystery` is set to 0 in line 3.

R2.8

The variable `mystery` is being declared twice, first in line 1 and then again in line 2. A variable can only be initialized once. (If you remove the reserved word `int` on line 3, the statements will work just fine, and will set `mystery` to -3.)

R2.9

In the Java programming language, the `=` operator denotes an action, to replace the value of a variable. This usage differs from the traditional use of the `=` symbol as a statement about equality.

R2.10

```
title = "Big Java";
char letter = title.charAt(0); // letter would be 'B'
int titleLength = title.length(); // titleLength would be 8
```

R2.11

```
String message = "Hello";
message = message.toUpperCase();
```

R2.12

```
String message = "Hello";
message = message.replace("H", "h");
```

R2.13

```
String message = "Hello, World!";
message = message.replace(" ", "");
message = message.replace("!", "");
```

R2.14

An object contains state information. An object variable contains an object reference, that is, the location of an object.

R2.15

```
new Rectangle(5, 10, 20, 30); // Object
Rectangle box; // Object variable
```

R2.16

```
new Rectangle(75, 75, 50, 50)
"Hello, Dave!"
```

R2.17

```
Rectangle square = new Rectangle(75, 75, 50, 50);
String greeting = "Hello, Dave!";
```

R2.18

```
Rectangle square = new Rectangle(10, 20, 40, 40);
square = new Rectangle(10, 20, 40, 40);
```

R2.19

```
Rectangle square1 = new Rectangle(20, 20, 40, 40);
Rectangle square2 = square1; // the same object
```

R2.20

- `newRectangle` is missing
- `Rectangle(5, 10, 15, 20)` does not refer to an object. The corrected version should be:

```
double width = (new Rectangle(5, 10, 15, 20)).getWidth();
```
- `v` has not been initialized; it does not refer to any object.
- The method `translate` takes two integer arguments, not a string argument.

R2.21

Possible answers are:

Accessor: `getWidth()`, `getHeight()`

Mutator: `translate(int, int)`, `add(int, int)`

R2.22

| Class | Return type | Method name | Types of arguments |
|-----------|-------------|-------------|--------------------|
| String | String | concat | String |
| String | String | trim | none |
| Rectangle | String | toString | none |
| Rectangle | Rectangle | union | Rectangle |
| Random | float | nextFloat | none |

R2.23

An object contains state information. An object reference is the location of that object in memory.

R2.24

Console applications provide text-only interfaces and cannot display any drawings/figures (except ASCII art). Graphical applications are more user friendly and can display drawings inside frames (windows).

R2.25

The Swing toolkit calls the `paintComponent` method whenever the component needs to be repainted. For example, it is called when the window is shown for the first time, when it is resized, or when it is shown again after it was hidden.

R2.26

The designers of Java did not want to inconvenience those programmers who had produced programs that used simple graphics from the `Graphics` class after they developed the newer, more powerful `Graphics2D` class, so they did not change the parameter of the `paintComponent` method to `Graphics2D`.

R2.27

The purpose of a graphics context is to store the programmer choices for colors, fonts, and so on, and to draw and fill shapes.

R2.28

The `paintComponent` method of the component class produces a drawing, while the `main` method of the viewer class constructs a frame and a component, adds the component to the frame, and makes the frame visible.

R2.29

You specify a text color simply by calling the `setColor` method of the graphics context before calling the `drawString` method.

Chapter 3

R3.1

A digitally controlled thermostat provides an interface that allows it to be manipulated through a smart phone. The interface specifies operations that can be accessed wirelessly via the smart phone.

One way of doing this is to have the thermometer run a web server that can be accessed via the internet, thereby enabling the smartphone to access it. The operations supplied via the digital interface would include the ability to connect to the thermostat, to turn the heating to verify the identity of the connecting agent, to turn the heat and A/C on and off, to modify the temperature, to set a schedule, to report the current temperature, etc.

The implementation of the interface encapsulates all the details for controlling the temperature by hiding them from the user, who only needs to know to interact with the user interface. In addition, from the perspective of the heating and cooling units, nothing has changed, because all the signals they received before from manual thermostats are still generated by the digital thermostats.

R3.2

The public interface of the Counter in Section 3.1 consists of the click, getValue, and reset methods. The public interface specifies the functionality supported by the class but does not disclose any details of how the functionality is implemented. In contrast, the implementation of a class is the code that accomplishes the tasks to support the class's functionality.

R3.3

Encapsulation is the process of hiding implementation details while publishing an interface for programmers to use. If you hide the implementation details, you can diagnose errors and make changes and improvements to the implementation of a class without disrupting the work of programmers who use the class.

R3.4

The `private` reserved word prevents a programmer using the class from manipulating instance variables except through the methods of that class. A programmer must use the public interface—the public methods that access or modify the instance variables—to change them. As such, the instance variables are hidden from methods outside of the class.

R3.5

```
private String grade;
// or a numeric value that corresponds to the letter grade to simplify gpa calculations
private double grade;
```

R3.6

Represent the time as one entire string value, example "8:35 a.m."

```
private String value;
// or as individual components
private int hour;
private int minute;
private String meridian;
```

R3.7

The programmers using the `Time` class do not have to do anything. The code that uses the `Time` class is written based on the public interface and, as such, it need not change unless the public interface changes. That is the purpose of encapsulation and using a public interface.

R3.8

No, there should not be a `setValue` method. The Counter class, as the original example indicates, models a device that can be used to count people. The tally increases by one for each person counted. The required functionality does not dictate the need to begin a tally at a value other than zero.

If there is a need to begin a count at a value other than zero, then a constructor should be added to support such functionality. A mutator such as `setValue` should only be added if there is a need to support functionality to reset an existing counter to a value other than zero.

R3.9

(a) If `BankAccount(double initialBalance)` did not exist, we could use the default constructor to create a `BankAccount` and then use the `deposit` method to increase the balance.

```
BankAccount account = new BankAccount();
account.deposit(1500);
```

is equivalent to

```
BankAccount account = new BankAccount(1500);
```

(b) (The previously provided answer does not address the question asked.)

If the `BankAccount()` constructor is removed from the `BankAccount` class, then creating an object of the `BankAccount` class would require using the `BankAccount(double initialBalance)` constructor. This constructor allows the specification of an `initialBalance` which can be 0.

```
BankAccount account = new BankAccount(0);
```

is equivalent to (with the no-argument constructor available)

```
BankAccount account = new BankAccount();
```

R3.10

A `reset` method effectively empties the account without any accountability. It would be best to create a new object if you want to delete an account and start with a new one. Otherwise, use the `withdraw` method to reduce the balance down to zero, which is how a real bank account should work.

R3.11

The account will have a negative balance. There are no checks to detect or stop this from happening.

R3.12

The `this` reference is used to refer to the implicit parameter of a method. It can be used to clarify the code, to explicitly access instance variables and methods in the object referenced by the implicit parameter, and to call another constructor of the same class.

R3.13

Mutator Methods

```
recordPurchase
receivePayment
```

Accessor Method

```
giveChange
```

R3.14

The method transfers an amount from the account passed as the implicit parameter to the account that is given as an explicit parameter.

Here is a sample call:

```
BankAccount harrysChecking = new BankAccount();
BankAccount momsSavings = new BankAccount(10000);
momsSavings.mystery(harrysChecking, 1000);
```

R3.15

```
public class TimeDepositAccount
{
    private double balance;
    private double interestRate;
    // Constructor
    public TimeDepositAccount(
        double initialBalance, double interestRate)
    {
        this.balance = initialBalance;
        this.interestRate = interestRate;
    }
    // Methods
    public void withdrawAll()
    {
        balance = 0.0;
    }
    public double getBalance()
    {
        return balance;
    }
    public void addInterest()
    {
        balance = balance + balance * interestRate;
    }
}
```

R3.16

Instance variables are initialized when the object is created. In this case, `area` would be initialized to zero and wouldn't get calculated with the correct value until `getArea` is called. If another method were added to this class that used `area`, it would incorrectly use the value zero if `getArea` had not been called first to calculate the correct value. As is, `area` is used in only a single method, `getArea`, and does not hold a value that must exist across method calls. This suggests that `area` is a candidate for being turned into a variable local to `getArea`.

```
public class Square
```

```

{
    private int sideLength;

    public Square(int length)
    {
        sideLength = length;
    }

    public int getArea ()
    {
        int area; // local variable

        area = sideLength * sideLength;
        return area;
    }
}

```

R3.17

If the method `grow` is called, the value of the instance variable `area` will no longer be correct. You should make `area` a local variable in the `getArea` method and remove the calculation from the constructor.

```

public class Square
{
    private int sideLength;

    public Square(int length)
    {
        sideLength = length;
    }

    public int getArea ()
    {
        int area; // local variable

        area = sideLength * sideLength;
        return area;
    }

    public void grow ()
    {
        sideLength = 2 * sideLength;
    }
}

```

R3.18

```

/**
 * A class to test the Counter class.
 */
public class CounterTester
{
    /**
     * Tests the methods of the Counter class.
     * @param args not used
     */
    public static void main(String[] args)
    {
        Counter tally = new Counter();
        int result = tally.getValue();
        System.out.println(result);
        System.out.println("Expected 0");

        tally.count();
        tally.count();
        tally.count();
        result = tally.getValue();
        System.out.println(result);
        System.out.println("Expected 3");

        tally.reset();
        result = tally.getValue();
        System.out.println(result);
        System.out.println("Expected 0");
    }
}

```

R3.19

```

/**
 * A class to test the Car class.
 */
public class CarTester
{
    /**
     * Tests the methods of the Car class.
     * @param args not used
     */
    public static void main(String[] args)
    {
        Car myHybrid = new Car(50); // 50 miles per gallon
        myHybrid.addGas(20); // Fill tank with 20 gallons
        myHybrid.drive(100); // Drive 100 miles, use 2 gallons
        myHybrid.addGas(10); // Add 10 gallons to tank
        myHybrid.drive(200); // Drive 200 miles, use 4 gallons
        double gasLeft = myHybrid.getGasInTank(); // Gas remaining in tank
        System.out.println(gasLeft);
        System.out.println("Expected 24");
    }
}

```

R3.20

Card Front:

```

BankAccount harrysChecking
BankAccount
BankAccount(initialBalance)
deposit(amount)
withdraw(amount)
getBalance

```

Card Back:

```

balance

0
2000
1500

```

R3.21

Card Front:

```

CashRegister register

```

```

CashRegister
recordPurchase(amount)
receivePayment(amount)
giveChange

```

Card Back:

```

purchase payment
0 0
29.50 0
38.75 0
38.75 50
0 0

```

R3.22

Card Front:

```

Menu mainMenu

```

```

addOption(option)
display

```