BIG JAVA BOOK REVIEW EXERCISE ALL SOLUTION CHAPTER 1-22

WRITTEN BY SUJAN PRODHAN

COMPUTER SCIENCE AND ENGINEERING ,RAJSHAHI UNIVERSITY

BATCH-29

It took me a lot of time to make it. I made this pdf with constant effort. The answers were collected one by one from different websites. One of the reasons for making this pdf is to help all my friends including me. So that the younger ones don't get harassed because I can't find the answer. It takes money to get the answer and there are many other problems. However, I did it for my dear friends. For any need E-mail: ***ru.cse29@gmail.com Website: prodhan2.blogspot.com 01902383808***

Contents

Chapter 1 Review Exercise Solutions

R1.1

Programs are sequences of instructions and decisions that a computer carries out to achieve a task. Programming means designing and implementing the instructions, as opposed to using their results.

R1.2

A household appliance carries out one task (or perhaps several tasks) for which it was designed. A computer can be programmed to carry out arbitrary tasks that the designers of the computer never envisioned.

R1.3

There are several possible answers; here is a typical one.

1. Point my browser to `http://briefcase.yahoo.com`
2. Type in my username and password when prompted
3. Click on the Select file button
4. Select the file `HelloPrinter.java` in the directory `c:\myfiles` from the file dialog box
5. Click Ok in the file dialog box
6. Click on the Upload button
7. Wait for the file upload to complete

R1.4

Answers may vary; the following are typical.

Sample file: `c:\myprograms\HelloPrinter.java`

Java interpreter: `c:\Program Files\Java\jdk1.5.0\bin\java.exe`

Run-time library: `c:\Program Files\Java\jdk1.5.0\jre\lib\rt.jar`

R1.5

Syntax errors are discovered by compiling the program and noting any problems displayed by the compiler. The compiler will not translate the program into machine code until all the syntax errors have been corrected. The compiler will also attempt to report as many syntax errors as it can find.

Logic errors are not readily obvious because the compiler will not detect them as it compiles the program. These types of errors are discovered by carefully examining the output and testing the program.

R1.6

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");
   }
}
```

R1.7

```
3 + 4
7
34
```

R1.8

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.
```

R1.9

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.
```

R1.10

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)
```

R1.11

```
Calculate car cost = (one-way distance x miles per gallon x 4) + (one-way distance x 0.05)
If the car cost > train ticket price
    Ride the train.
Otherwise drive.
```

R1.12

```
work mileage = number of work days x one-way distance x 2
total miles = ending mileage – beginning mileage
fraction of commute = work mileage / total miles
```

R1.13

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.

Chapter 2(Using Objects)

R2.1

Explain the difference between an object and a class.

Object is an instance of a class. Class is a blueprint or template from which objects are created. Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc. Class is a group of similar objects.

OR

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

R2.2

What is the public interface of a class? How does it differ from the implementation of a class?

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.

R2.4

Declare and initialize variables for holding the price and the description of an article that is available for sale.

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

R2.5

What is the value of mystery after this sequence of statements?
int mystery = 1;
mystery = 1 - 2 * mystery;
mystery = mystery + 1;

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.

R2.6

What is wrong with the following sequence of statements? int mystery = 1; mystery = mystery + 1; int mystery = 1 - 2 * mystery;

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.)

R2.7

. Explain the difference between the = symbol in Java and in mathematics..

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.

R2.8

Write Java statements that initialize a string message with "Hello" and then change it to "HELLO". Use the toUpperCase method.

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

R2.10

Write Java statements that initialize a string message with "Hello" and then change it to "hello". Use the replace method

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

R2.12

Explain the difference between an object and an object variable

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

R2.13

Give the Java code for constructing an object of class Rectangle, and for declaring an object variable of class Rectangle.

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

R2.14

Give Java code for objects with the following descriptions:
a. A rectangle with center (100, 100) and all side lengths equal to 50
b. A string with the contents "Hello, Dave"
Create objects, not object variables.new Rectangle(75, 75, 50, 50)

```
"Hello, Dave!"
```

R2.15

Repeat Exercise R2.14, but now declare object variables that are initialized with the required objects.

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

R2.16

Write a Java statement to initialize a variable square with a rectangle object whose top-left corner is (10, 20) and whose sides all have length 40. Then write a statement that replaces square with a rectangle of the same size and top-left corner (20, 20).

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

R2.17

Write Java statements that initialize two variables square1 and square2 to refer to the same square with center (20, 20) and side length 40.

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

R2.18

Find the errors in the following statements:
a. Rectangle r = (5, 10, 15, 20);
b. double width = Rectangle(5, 10, 15, 20).getWidth();
c. Rectangle r; r.translate(15, 25);
d. r = new Rectangle(); r.translate("far, far away!");

a. `new Rectangle` is missing

b. `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();
```

c. `r` has not been initialized; it does not refer to any object.

d. The method `translate` takes two integer arguments, not a string argument.

R2.19

Name two accessor methods and two mutator methods of the Rectangle class.

Possible answers are:

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

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

R2.20

Consult the API documentation to find methods for

• Concatenating two strings, that is, making a string consisting of the first string, followed by the second string.

```
String first = "eyad ";

String secend = "ayman";

String third = first.concat(secend);

System.out.println(third);

/*

Class : String

return type : String

method name : concat

types of the arguments : String

*/
```

• Removing leading and trailing white space of a string.

```
String remove = " This is Trim Method ! ";

System.out.println(remove.trim());

/*

Class : String

return type : String

method name : trim
```

types of the arguments : no arguments

*/

• **Converting a rectangle to a string.**

Rectangle box = new Rectangle( );

System.out.println(box.toString( ));

/*

Class : Rectangle

return type : String

method name : toString

types of the arguments : no arguments

*/

• **Computing the smallest rectangle that contains two given rectangles.**

Rectangle box1 = new Rectangle(5,5,20,30);

Rectangle box2 = new Rectangle(20,30,20,30);

System.out.println(box1.union(box2));

/*

Class : Rectangle

return type : Rectangle

method name : union

types of the arguments : Rectangle

*/

• Returning a random floating-point number.

```
Random rNumber = new Random();

System.out.println(rNumber.nextFloat());

/*

Class : Random

return type : float

method name : nextFloat

types of the arguments : no arguments

*/
```

R2.22

~~**lagbena tai kete dewa hpyese.........(sujan prodhan-ru.cse-29)**~~
~~*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).~~

R2.21

Explain the difference between an object and an object reference.
An object contains state information. An object reference is the location of that object in memory.

R2.23

~~The `paintCompoment` 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.~~

R2.27

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

Chapter 3 Review Exercise Solutions

R3.1

What is the public interface of the Counter class in Section 3.1? How does it differ from the implementation of the class?

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.

R3.2

What is encapsulation? Why is it useful?

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.

R3.3

Instance variables are a part of the hidden implementation of a class, but they aren't actually hidden from programmers who have the source code of the class. Explain to what extent the private reserved word provides information hiding.

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.

R3.4

Consider a class Grade that represents a letter grade, such as A+ or B. Give two choices of instance variables that can be used for implementing the Grade class

```
private String grade;
```

or a numeric value that corresponds to the letter grade to simplify gpa calculations

```
private double grade;
```

R3.5

Consider a class Time that represents a point in time, such as 9 a.m. or 3:30 p.m. Give two different sets ofinstance variables that can be used for implementing the Time class.

Represent the time as one entire string value, example “3:30 a.m.”

```
private String value;
```

or as individual components

```
private int hour;

private int minute;

private String meridian;
```

R3.6

uppose the implementor of the Time class of Exercise R3.5 changes from one imple_mentation strategy to another, keeping the public interface unchanged. What do the programmers who use the Time class need to do?

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.

R3.7

You can read the value instance variable of the Counter class with the getValue accessor method. Should there be a setValue mutator method to change it? Explain why or why not.

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.

R3.8

a. Show that the BankAccount(double initialBalance) constructor is not strictly necessary. That is, if we removed that constructor from the public interface, how could a programmer still obtain BankAccount objects with an arbitrary balance?

(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. Conversely, could we keep only the BankAccount(double initialBalance) con_structor and remove the BankAccount() constructor?`

(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();

R3.9

Why does the BankAccount class not have a reset method?

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.

R3.10

What happens in our implementation of the BankAccount class when more money is withdrawn from the account than the current balance

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

R3.11

What is the this reference? Why would you use it?

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.

R3.12

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);
```

R3.14

Suppose you want to implement a class TimeDepositAccount. A time deposit account has a fixed interest rate that should be set in the constructor, together with the initial balance. Provide a method to get the current balance. Provide a method to add the earned interest to the account. This method should have no arguments because the interest rate is already known. It should have no return value because you already provided a method for obtaining the current balance. It is not possible to deposit additional funds into this account. Provide a withdraw method that removes the entire balance. Partial withdrawals are not allowed.

```
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;
   }
}
```

R3.15

Consider the following implementation of a class Square:
public class Square {
private int sideLength; private int area; // Not a good idea public Square(int length)
{ sideLength = length; }
public int getArea()
{ area = sideLength * sideLength;
return area; } }
Why is it not a good idea to introduce an instance variable for the area? Rewrite the class so that area is a local variable.

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;
   }
}
```

R3.1

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;
   }
}
```

R3.16

```
/**
   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");

   }
}
```

R3.17

```
/**
   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");
   }
}
```

R3.19

Card Front: BankAccount harrysChecking
BankAccount
BankAccount(initialBalance)
deposit(amount)
withdraw(amount)
getBalance

Card Back: balance
~~0~~
~~2000~~
1500

R3.19

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

R3.20

Card Front: Menu mainMenu
Menu
addOption(option)
display

Card Back:

optionCount	menuText
~~0~~	~~“”~~
~~1~~	~~“1) Open new account\n”~~
~~2~~	~~“1) Open new account\n2) Log into existing account\n”~~
~~3~~	~~“1) Open new account\n2) Log into existing account\n3) Help\n”~~
4	“1) Open new account\n2) Log into existing account\n3) Help\n4) Quit\n”

R3.21

First, we need to add a new instance variable to keep track of the number of transactions each month. We will add the following instance variable to the class:

```
int numTransactions;
```

This variable should be incremented by one inside the `withdraw` and `deposit` methods. The following line of code could be added to the end of these methods:

```
numTransactions++;
```

And, of course, we should initialize this variable to zero inside both constructors.

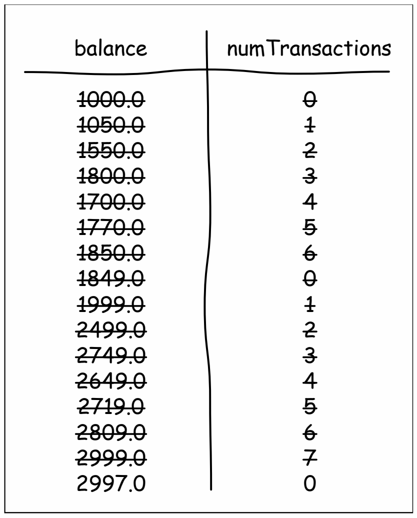
Finally, we need to add a new instance method that will calculate and deduct the fee ($1 for each transaction over 5) and reset the `numTransactions` variable to zero (to start the new month).

```
public void calculateTransFee()
{
   if (numTransactions > 5)
   {
      balance = balance - (numTransactions - 5);
   }

   numTransactions = 0;
}
```

And for the new object trace, let’s assume the following method calls are made over two months

```
BankAccount harrysAccount(1000);
harrysAccount.deposit(50);
harrysAccount.deposit(500);
harrysAccount.deposit(250);
harrysAccount.withdraw(100);
harrysAccount.deposit(70);
harrysAccount.deposit(80);
harrysAccount.calculateTransFee();
harrysAccount.deposit(150);
harrysAccount.deposit(500);
harrysAccount.deposit(250);
harrysAccount.withdraw(100);
harrysAccount.deposit(70);
harrysAccount.deposit(90);
harrysAccount.deposit(190);
harrysAccount.calculateTransFee );
```

Here are the updated object cards showing the results of these method calls:



balance	numTransactions
~~1000.0~~	~~0~~
~~1050.0~~	~~1~~
~~1550.0~~	~~2~~
~~1800.0~~	~~3~~
~~1700.0~~	~~4~~
~~1770.0~~	~~5~~
~~1850.0~~	~~6~~
~~1849.0~~	~~0~~
~~1999.0~~	~~1~~
~~2499.0~~	~~2~~
~~2749.0~~	~~3~~
~~2649.0~~	~~4~~
~~2719.0~~	~~5~~
~~2809.0~~	~~6~~
~~2999.0~~	~~7~~
2997.0	0

You can see that the `numTransactions` variable resets whenever the fee is calculated, and then a fee is subtracted from the balance (one dollar for every transaction over 5).

R3.22

You need four classes: `House`, `Car`, `SceneComponent`, and `SceneViewer`.

R3.23

The methods have the same implicit parameter as the `paintComponent` method. We could have written the calls as `this.getHeight()` and `this.getWidth()`. They are accessor methods so they don't need an explicit parameter; the instance variables of the object are all they need to do their work.

R3.24

Add `width` and `height` fields to the `Car` class, initialize them in the `Car` constructor, and use them in the `draw` method to determine the dimensions of the body, tires, and windshield

Chapter 4: Review Exercise Solutions

R4.1

- This is somewhat ambiguous. Are the variables local variables, or class variables? I assume class variables:
- **A.**`public static final int DAYS_IN_WEEK = 7;`
- **B.**`public int daysToSemesterEnd;`
- **C.**`public static final double CENTIMETERS_IN_INCH = 2.54;`
- **D.**`public double tallestHeight;`

R4.2

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.

R4.3

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.)

R4.4

a) dm = m ((√(1 + v)/c) / (√(1 – v)/c)) – 1
b) volume = π r^2h
c) volume = (4 π r^3h)/ 3
d) z = √(x^2 + y^2)

R4.5
a) `double s = s0 +(v0 * t) + (.5 * g * Math.pow(t,2));`
b) double g = 4 * Math.PI * Math.PI* (Math.pow(a,3)/(p * p * (m1 + m2)));
c) double fv = pv * Math.pow((1 + (intRate / 100)), yrs);
d) double c = Math.sqrt((a * a) + (b * b) - (2.0 * a * b * Math.cos(gamma)));

R4.8

a) `6.25`
b) `6`
c) `12.5`
d) `-3`
e) `1.4142135623730951`

R4.9

a) `8`
b) `1`
c) `17`
d) `17.5`
e) `17`
f) `18`

R4.10

a) `10`
b) `e`
c) `llo`
d) `HelloWorld`
e) `WorldHello`

R4.11

1) There is an extraneous semicolon after `main()`.
2) The message in the first print statement is not surrounded by quotation marks like a string should be.
3) The variables `x` and `y` are not declared.
4) The variable `in` is not declared.
5) The method `readDouble` doesn't exist (`nextDouble` does).
6) If `readDouble` were a method, it should be followed by `()`.
7) In the last line the method `println` is incorrectly spelled `printline`.

R4.12

1) The scanner should have been constructed without quotations as `Scanner in = new Scanner(System.in);`
2) The correct Scanner method for reading integers is nextInt();
3) The second integer should have been read as `y = in.nextInt();`
(The problem requests 3 answers but there are 4 mistakes)

R4.13

The given output is printed in a raw format up to the range of a `double` data type. Users can use *format specifiers* (`printf` with `%`) to format the output as they require.

R4.14

The type of 2 is int. The type of `2.0` is a double; specifically the Java programming language recognizes it as the real number 2.0. The type of `'2'` is a char. On the other hand, Java treats `"2"` and `"2.0"` as strings.

R4.15

a) This statement computes the value of y. Given the initial value of x as 2, the result for y is 4.
b) This statement concatenates the strings `"2"` together. The result for t is `"22"`.

R4.16

Read input into a variable called word.
Print first character in word followed by a space.
Print character at length -1 in word followed by a space.
Print substring between 1st and last character (length -1) in word.

R4.17

```
Read first name into variable called first.
Read middle name into variable called middle.
Read last name into variable called last.

Print first character in first.
Print first character in middle.
Print first character in last.
```

R4.18

```
// Input is stored in variable called num
exponent = integer part of log10 of num
first digit = integer part of num / 10^exponent
last digit = num % 10
Print first digit.
Print last digit.
```

R4.19

```
change due = 100 x bill value – item price in pennies
quarters = change due / 25 (without remainder)
change due = change due % 25
dimes = change due / 10 (without remainder)
amount due = change due % 10
nickels = change due / 5 (without remainder)
// Change is now in quarters, dimes, and nickels
```

R4.20

First, compute the total volume by hand using the formula in Self Check 25. Let's assume the following values for the three sections:

- Conic Section 1 (top of bottle) has first radius ($r1_1$) 0.5 inches, second radius ($r2_1$) 0.75 inches, and height (h_1) 1.5 inches.
- Conic Section 2 (middle) has first radius ($r1_2$) 0.75 inches, second radius ($r2_2$) 1.5 inches, and height (h_2) 0.5 inches.

- Conic Section 3 (bottom of bottle) has first radius ($r1_3$) 1.5 inches, second radius ($r2_3$) 0.75 inches, and height (h_3) 6.0 inches.

So, the total volume of the bottle, V_t, will equal the volume of conic section 1 (V_1) plus the volume of conic section 2 (V_2) plus the volume of the conic section 3 (V_3):

```
Vt = V1 + V2 + V3
```

We calculate the three volumes to be (rounded to two decimal places):

```
V1 = 1.87 cubic inches
V2 = 2.06 cubic inches
V3 = 24.74 cubic inches
```

So, the total volume for the cocktail shaker in our example is:

```
Vt = V1 + V2 + V3 = 28.67 cubic inches
```

The algorithm we use is very similar to the calculation we did above:

volume 1 = $\pi \times (r1_1^2 + r1_1 \times r2_1 + r2_1^2) \times h1 / 3$
volume 2 = $\pi \times (r1_2^2 + r1_2 \times r2_2 + r2_2^2) \times h2 / 3$
volume 3 = $\pi \times (r1_3^2 + r1_3 \times r2_3 + r2_3^2) \times h3 / 3$
total volume = volume 1 + volume 2 + volume 3

R4.21

Because we are given the diameter of the circle, d, and not the height of the circular sector, h, we need to develop an algorithm to find h based on the value of d.

If you look at the figure on the right, you see that a right triangle can be made bounded by the chord, the diameter of the circle, and the radius of the circle drawn through h.

Because we know the length of the radius (half the diameter), we can set up an equation to find the value of h:

$h = 0.5d - x$

where x is one side of the right triangle. We can solve for x by using trigonometry on the right triangle, and then use that value to solve for the value of h, which we need to compute the area of the circular sector (the oddly-shaped piece of pie). In the right triangle, the side labeled x is considered to be the "opposite" side. We know that the hypotenuse of the right triangle is half the diameter, or 0.5d, and the "adjacent" side of the right triangle is half the chord length, or 0.5c. Therefore, we can use the following formulas for a right triangle to determine what we need for the problem:

```
sin θ = opposite/hypotenuse    cos θ = adjacent/hypotenuse
```

Here is the general algorithm (see figure for definition of the variables):

```
Ask the user to enter the diameter of the circle, store the value in d.
Ask the user to enter the length of the chord, store the value in c.
Compute the angle θ with formula $\theta = \cos^{-1}(0.5c / 0.5d)$.
Compute the length of x with the formula $x = \sin\theta * 0.5d$.
Determine h using the formula $h = 0.5d - x$.
Calculate the area of the pie piece, A, using the formula.
A = 2.0 / 3 * c * h + pow(h, 3) / (2 * c).
```

We can use this algorithm to solve for the specific example of a pie with diameter 12 inches and a chord length of 10 inches:

We know that d = 12, c = 10.
Therefore, $\theta = \cos^{-1}(0.5c/0.5d) = \cos^{-1}(5/6) = 33.56$ degrees
$x = \sin\theta * 0.5d = \sin 33.56 * 6 = 3.32$ inches
$h = 0.5d - x = 6 - 3.32 = 2.68$ inches
$A = 2/3 * c * h + h^3/(2*c) = 2/3 * 10 * 2.68 + 2.68^3 / (2 * 10) = 18.83$ square inches

R4.22

Starting position is 3 x 4 = 12 and length of substring is 3.

S	u	n	M	o	n	T	u	e	W	e	d	T	h	u	F	r	i	S	a	T
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

We can see this will extract the substring `Thu`.

R4.23

Character at position `i` (2)

G	a	t	e	w	a	y

0 1 2 3 4 5 6

Character at position `j` (4)

```
G a t e w a y
```
0 1 2 3 4 5 6

first:

```
G a t e w a y
```
0 1 2 3 4 5 6

middle:

```
G a t e w a y
```
0 1 2 3 4 5 6

last:

```
G a t e w a y
```
0 1 2 3 4 5 6

Generate a new string: first + character at j + middle + character at i + last

```
G a w e t a y
```
0 1 2 3 4 5 6

R4.24

You can read characters from a string with the `charAt` method. For the first character, pass a position of 0 to `charAt`. For the last character pass the position that is equal to the length of the string -1 to `charAt`.

Strings in Java are immutable, so they cannot be directly changed. Thus, to "remove" the first character from a string, we can take a substring of the original string that contains the entire thing minus the first character. If our string is called `s`, then this works: `s.substring(1, s.length());`. The last character can be obtained by extracting the substring that contains the entire string minus the last character, like this: `s.substring(0, s.length()-1);`.

R4.23

This program:

```
public class R222
{
   public static void main(String[] args)
   {
      System.out.println(3 * 1000 * 1000 * 1000);
      System.out.println(3.0 * 1000 * 1000 * 1000);
   }
}
```

Prints out the following:

```
-1294967296
3.0E9
```

The reason the first number is negative is because we have exceeded the limited range of an `int` and the number overflowed to a negative value. The second number's result is correct but displayed in scientific notation because it is a floating-point type from the `3.0` in the calculation.

Chapter 5: Review Exercise Solutions

R5.1

a) n = 1, k = 2, r = 1
b) n = 1, k = 2, r = 2
c) n = 1, k = 1, r = 2
d) n = 1, k = 6, r = 3

R5.2

In the first block, the conditions are evaluated sequentially, so `s` could be incremented twice. In the second block, the conditional statements are mutually exclusive, so, `s` cannot be incremented more than once.

R5.3

a) There are two errors in this code. First, there are no parentheses around the `x > 0` part of the `if` statement. Secondly, there is an extraneous `then` which is not part of the Java syntax, instead we should wrap the body of the `if` statement in curly braces. A correct form would be:

```
if (x > 0) { System.out.print(x); }
```

b) In this statement there aren't enough parentheses to balance the condition (there are only two). The following would be correct:

```
if (1 + x > Math.pow(x, Math.sqrt(2))) { y = y + x; }
```

c) In this case there is only a single `=` in the `if` statement. Likely the programmer wanted to check for equality rather than set `x` equal to the value of `1`. A correct form would be:

```
if (x == 1) { y++; }
```

d) The problem in this case is that the programmer tried to validate the input after she read it thus defeating the whole purpose of input validation. Instead we should take the line which reads the integer into the body of the if statement, like this:

```
if (in.hasNextInt())
{
   x = in.nextInt();
   sum = sum + x;
}
else
{
   System.out.println("Bad input for x");
}
```

e) The `if` statements should be an `if/else if/else` sequence. More than one `if` statement will be executed for any grade higher than 60 and the letter grade will be wrongly assigned.

R5.4

a) `-1`
b) `1`
c) `1.0`
d) `2.0`

R5.5

```
if (x > 0.0)
{
   y = x;
}
else
{
   y = 0.0;
}
```

R5.6

```
if (x < 0.0)
{
   y = -x;
}
else
{
   y = x;
}
```

R5.7

Floating-point numbers only have limited precision so it’s very likely that any mathematical operation (like addition, multiplication, etc.) will introduce small errors into the result.

To check to see if an integer equals 10:

```
if (n == 10)
{
   System.out.println("It's 10!");
}
else
{
   System.out.println("It's not 10!");
}
```

To check to see if a floating-point number approximately equals 10:

```
final double EPSILON = 1E-14;
double x = 10.0;
if (Math.abs(x - 10) < EPSILON)
{
   System.out.println("It's approximately 10.0!");
}
else
{
   System.out.println("It's not 10.0!");
}
```

R5.8

In the first case when comparing `if (floor = 13)` you should get the error “Type mismatch: cannot convert from int to boolean".

In the statement `count == 0;` you should get the error “Syntax error on token “==”, invalid assignment operator”.

R5.9

letter	number	color
g	5	“black”

R5.10

The following test cases cover all four branches.

a) “g5”
b) “h5”
c) “g4”
d) “h4”

R5.11

Trace of appointment from 10-12 and one from 11-13:

start1	start2	end1	end2	s	e
10	11	12	13	11	12

Since s < e the program would report “The appointments overlap.” Which is indeed the case.

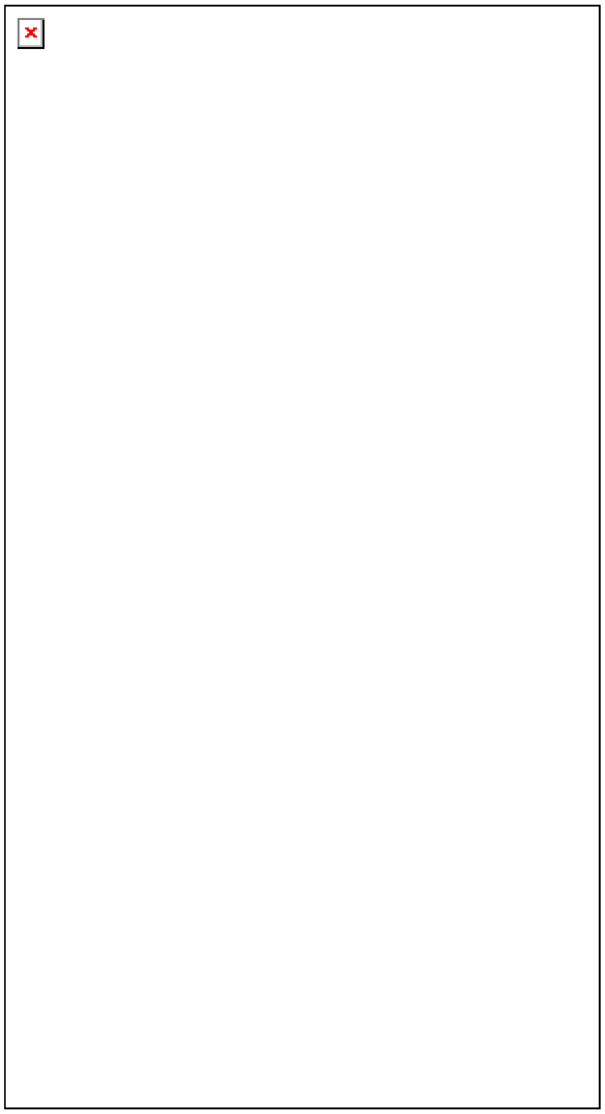
Trace of appointment from 10-11 and one from 12-13

start1	start2	end1	end2	s	e
10	12	11	13	12	11

Since s > e the program would report “The appointments don’t overlap.” Which is correct.

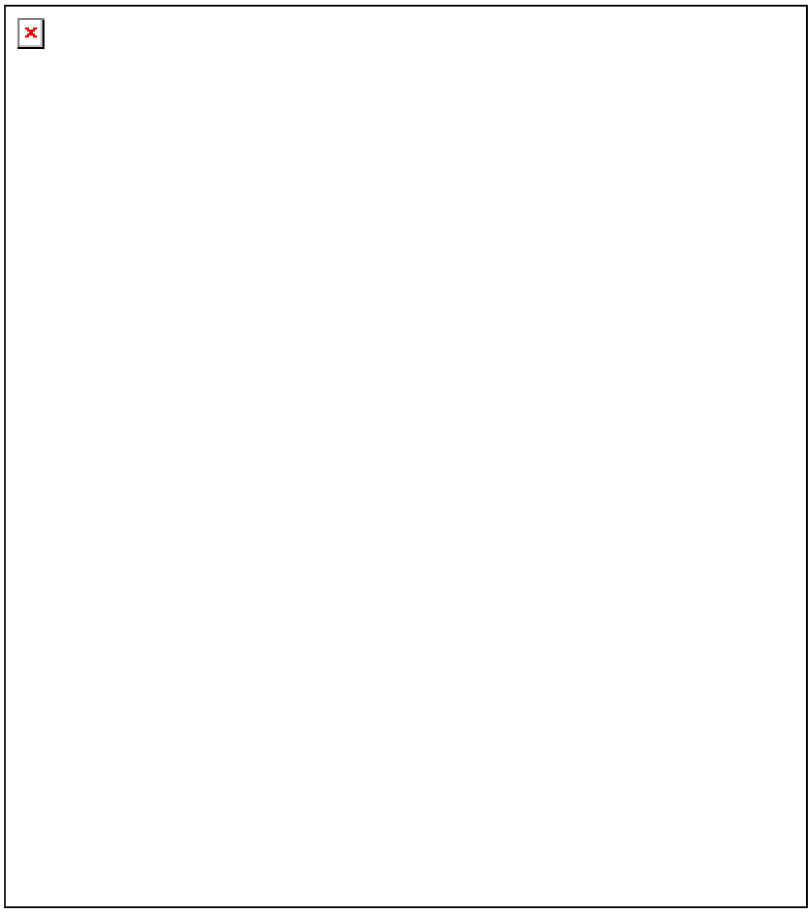
R5.12

The flowchart for Exercise R3.11:



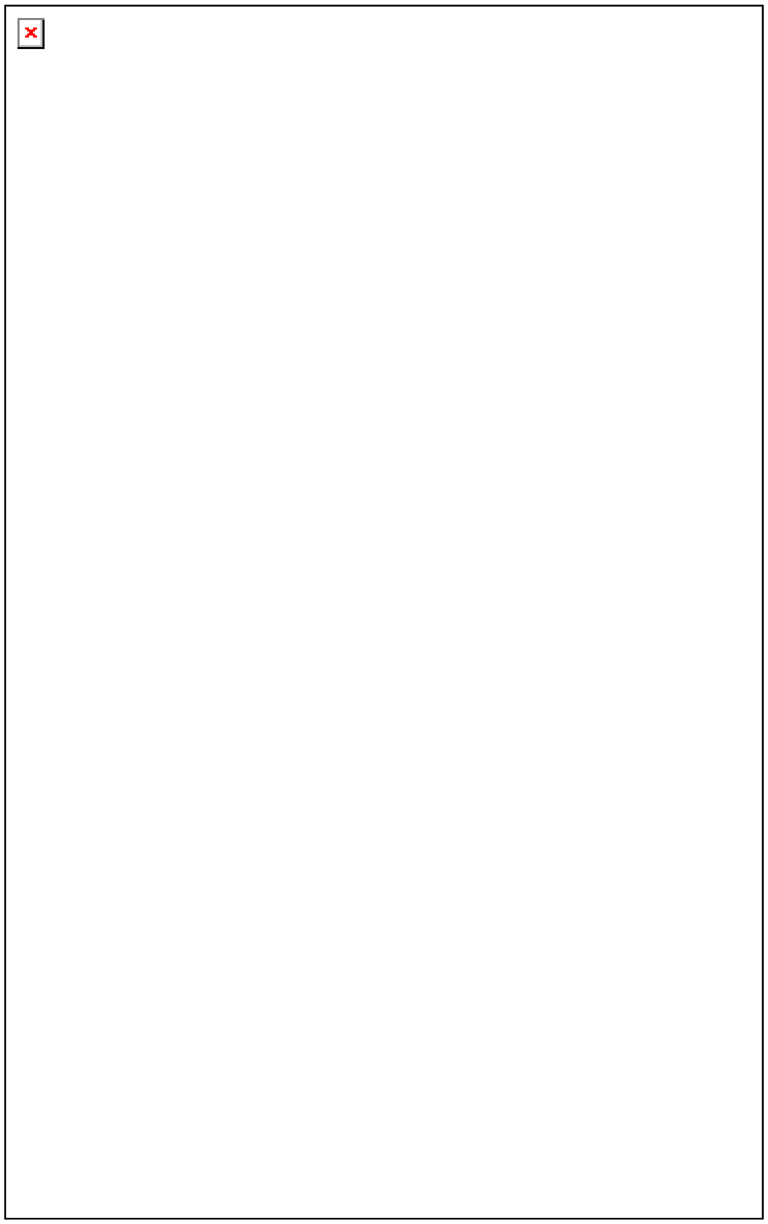
R5.13

The flowchart:



R5.14

The flowchart:



R5.15

Test Case	Expected Outcome	Comment

Start1=12 End1=14
Start2=15 End2=16
	No overlap	Completely separate appointments

Start1=12 End1=14
Start2=13 End2=15
	Overlap	Overlap by 1 hour

Start1=12 End1=14
Start2=11 End2=15
	Overlap	Appointment 2 starts before and ends after Appointment 1

Start1=12 End1=14 Start2=9 End2=10	No overlap	Appointment 2 starts and ends before Appointment 1
Start1=12 End1=14 Start2=12 End2=14	Overlap	Exact same appointment
Start1=12 End1=14 Start2=14 End2=17	No Overlap	Appointment 2 is right after Appointment 1, boundary condition

R5.16

Test Case	Expected Outcome	Comment
Month=1, Day=10	Season=Winter	First season, not divisible by 3
Month=4, Day=10	Season=Spring	Second season, not divisible by 3
Month=7, Day=10	Season=Summer	Third season, not divisible by 3
Month=10, Day=10	Season=Fall	Fourth season, not divisible by 3
Month=3, Day=21	Season=Spring	Second season, divisible by 3, boundary condition
Month=6, Day=21	Season=Summer	Third season, divisible by 3, boundary condition
Month=9, Day=21	Season=Fall	Fourth season, divisible by 3, boundary condition
Month=12, Day=21	Season=Winter	First season, divisible by 3, boundary condition
Month=3, Day=20	Season=Winter	First season, divisible by 3, beneath boundary
Month=6, Day=20	Season=Spring	Second season, divisible by 3, beneath boundary
Month=9, Day=20	Season=Summer	Third season, divisible by 3, beneath boundary
Month=12, Day=20	Season=Fall	Fourth season, divisible by 3, beneath boundary

R5.17

```
Prompt for and read month from user.
Prompt for and read day from user.
if month equals "January"
    if day equals 1
        print "New Year's Day"
```

```
if month equals "July"
    if day equals 4
        print "Independence Day"
if month equals "November"
    if day equals 11
        print "Veterans Day"
if month equals "December"
    if day equals 25
        print "Christmas Day"
```

R5.18

```
if score >= 90
    assign A
else if score >= 80
    assign B
else if score >= 70
    assign C
else if score >= 60
    assign D
else
    assign F
```

R5.19

When comparing strings lexicographically each letter in the two strings are compared from first to last with the following rules:
• Letters of the same case come in dictionary order.
• All uppercase letters come before lowercase letters.
• Space comes before all printable characters.
• Numbers come before letters.
• Punctuation also has order listed in Appendix A.
When one letter is found that comes before another, the string it belongs to is ordered before the other. Thus the sample strings given in the problem have the following order:

```
"Century 21" < "IBM" < "While-U-Wait" < "wiley.com"
```

R5.20

a) `"Jerry"`
b) `"Tom"`

c) `"Churchill"`
d) `"car manufacturer"`
e) `"Harry"`
f) `"Car"`
g) `"Tom"`
h) `"Car"`
i) `"bar"`

R5.21

In the case of an `if/else if/else` sequence, one clause is guaranteed to be executed, while in the case of nested `if` statements the internal clause is only true if all the `if` statements are true.

As an example one of the following lines of code will be printed:

```
if (n > 10)
{
   System.out.println("Greater than 10!");
}
else if (n >= 0)
{
   System.out.println("Between 0 and 10.");
}
else
{
   System.out.println("Less than 0.")
}
```

Whereas in this case the print will only execute if `n` is both greater than or equal to 0 and less than or equal to 10:

```
if (n >= 0)
{
   if (n <= 10)
   {
      System.out.println("Between 0 and 10.");
   }
}
```

R5.22

When comparing exact values the order of `if/else if/else` statements does not matter:

```
if (n == 1)
{
   System.out.println("1.");
}
else if (n == 2)
{
   System.out.println("2.");
}
else
{
   System.out.println("Something else.");
}
```

When comparing ranges of values it can:

```
if (n > 10)
{
   System.out.println("Greater than 10!");
}
else if (n >= 0)
{
   System.out.println("Between 0 and 10.");
}
else
{
   System.out.println("Less than 0.");
}
```

R5.23

The condition rewritten to use < operators instead of >= operators:

```
if (richter < 4.5)
{
   cout << "No destruction of buildings";
}
else if (richter < 6.0)
{
   cout << "Damage to poorly constructed buildings";
}
else if (richter < 7.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter < 8.0)
{
   cout << "Many buildings destroyed";
}
else
{
   cout << "Most structures fall";
}
```

Changing the operators to < instead of >= forces the order of the options to be reversed, because of the order in which the comparisons are evaluated.

R5.24

status	income	tax
“Single”	$1050.00	$105.00
“Single”	$20000.00	$2600.00
“Single”	$40000.00	$6400.00
“Married”	$1050.00	$105.00
“Married”	$20000.00	$2200.00
“Married	$70000.00	$10300.00
“Single”	$8000.00	$800.00

R5.25

This is an example of the dangling else problem:

```
if (gpa >= 1.5)
   if (gpa < 2)
      status = "probation";
else
   status = "failing";
```

When reading the code it may mistakenly appear that a student fails only when the GPA is less than 1.5, when in reality the code executes like this:

```
if (gpa >= 1.5)
   if (gpa < 2)
      status = "probation";
   else
      status = "failing";
```

Now we see all students whose GPAs is over 2.0 are also listed as failing. To correct for this, we can add curly braces:

```
if (gpa >= 1.5)
{
   if (gpa < 2)
   {
      status = "probation";
   }
}
else
{
   status = "failing";
}
```

R5.26

```
  p     q     r    (p && q) || !r  !(p && (q || !r))
```

false	false	false	true	true
false	false	true	false	true
false	true	false	true	true
false	true	true	false	true
true	false	false	true	false
true	false	true	false	true
true	true	false	true	false
true	true	true	true	false

R5.27

This is true, the Boolean operation `&&` is symmetric. One can test this by filling in all values of a truth table and noting both operations *A && B* and *B && A* have the same truth values:

A	B	A && B	B && A
false	false	false	false
false	true	false	false
true	false	false	false
true	true	true	true

R5.28

Boolean operations in search engines are spelled out in English while in Java they have symbolic replacements. For example OR is equivalent to "`||`" in Java while AND NOT would be equivalent to a combination of "&&" and "`!=`".

R5.29

a) `false`
b) `true`
c) `true`
d) `true`
e) `false`
f) `false`
g) `false`
h) `true`

R5.30

a) `b`
b) `!b`
c) `!b`
d) `b`

R5.31

a) `b = (n == 0);`
b) `b = !(n == 0);`
c) `b = ((n > 1) && (n < 2));`
d) `b = (n < 1) || (n > 2);`

R5.32

The program attempts to read the integer before it actually tests to see if there's an integer to read. To correct this problem the `int quarters = in.nextInt();` line should be moved inside the body of the `if` statement.

Chapter -6

R6.1

. Given the variables

```
String stars = "*****";
String stripes = "=====";
```

what do these loops print?

a.

```
int i = 0;
while (i < 5)
```

```
{
   System.out.println(stars.substring(0, i));
   i++;
}
```

Answer:

```
*

* *

* * *

* * * *
```

Explantion: The given loop runs from i = 0 to 4, each time a substring of stars is printed.

When i is 0, stars.substring(0,i) will be (empty string) as the start and end index is zero hence the first line is empty space

When i is 1, stars.substring(0,i) will be * , the substring of stars from index 0 to 1

hence second line is *

When i is 2, stars.substring(0,i) ** , the substring of stars from index 0 to 2

hence third line is **

When i is 3, stars.substring(0,i) ***, the substring of stars from index 0 to 1

hence fourth line is ***

When i is 4, stars.substring(0,i) **** , the substring of stars from index 0 to 1

hence fifth line is ****

b.

```
int i = 0;
while (i < 5)
{
   System.out.print(stars.substring(0, i));
   System.out.println(stripes.substring(i, 5));
   i++;
}
```

Answer:

```
=====

*====

**===

***==

****=
```

Explanation: Here in this loop as we are using print in the first statement, the cursor stays on the current line print ='s and then moves on to the next line

The given loop runs from i = 0 to 4, each time a substring of stars is printed, followed by substring of ='s.

When i is 0, stars.substring(0,i) will be (empty string) as the start and end index is zero, the next line stripes.substring(i,5) gets the entire string as start index is zero and end index is 5 the entire string ===== is returned and printed

When i is 1, stars.substring(0,i) will be * , the substring of stars from index 0 to 1

next line stripes.substring(i,5) gets the entire string as start index is 1 and end index is 5 the string ==== is returned and *==== is printed

When i is 2, stars.substring(0,i) will be ** , the substring of stars from index 0 to 2

next line stripes.substring(i,5) gets the entire string as start index is 2 and end index is 5 the string === is returned and **=== is printed

When i is 3, stars.substring(0,i) will be *** , the substring of stars from index 0 to 3

next line stripes.substring(i,5) gets the entire string as start index is 3 and end index is 5 the string == is returned and ***==is printed

When i is 4, stars.substring(0,i) will be * , the substring of stars from index 0 to 1

next line stripes.substring(i,5) gets the entire string as start index is 1 and end index is 5 the string = is returned and ****=printed

c.

```
int i = 0;
while (i < 10)
{
   if (i % 2 == 0) { System.out.println(stars); }
   else { System.out.println(stripes); }
}
```

The above loop becomes an infinite loop as i is 0 and there not statement to increment the value of i and 0 is always less than 10 and hence leads to infinite loop.

R6.2

What do these loops print?

a.

```
int i = 0; int j = 10;
while (i < j) { System.out.println(i + " " + j); i++; j--; }
```

Answer:

```
0 10

1 9

2 8

3 7

4 6
```

The above loop prints the values of i and j separated by space and then split the line.

The loop runs along as i is less then j, for every i < j print i and j separated by j, each time increment the value of i by 1 and decrement the value of j by 1.

b.

```
int i = 0; int j = 10;
while (i < j) { System.out.println(i + j); i++; j++; }
```

Answer: The above loop leads to an infinite loop, the while header states to continue the loop as long as i < j. Initially we have i = 0 and j = 0, the condition in the while loop is satified and prints 0+10 = 10, next i and j are incremented by 1, now i is 1 and j is 11 and prints 12, and the loop continues. If we observer the value of i will always be less than 10 and hence it leads to infinite loop.

R6.3

A. 1 3 8 10 15 21 28 36 45 55

B. 0 2 1 3 2 4 3 5 4 6

C. 5 x 1= 5 | 4 x 5 = 20 | 3 x 20 = 60 | 2 x 60 = 120 | 1 x 120 = 120

D. 1 x 1 = 1 | 2 x 1 = 2 | 4 x 2 = 8 | 8 x 8 = 64

R6.4

a)
```
int sumOfEvenNumbers = 0;
for (int n = 2; n <= 100; n = n + 2)
{
   sumOfEvenNumbers = sumOfEvenNumbers + n;
}
// sumOfEvenNumbers now has the sum of all even numbers from 2 to 100
```

b)
```
int sumOfSquares = 0;
int currentN = 1;
while (Math.pow(currentN, 2) <= 100)
{
   sumOfSquares = sumOfSquares + Math.pow(currentN, 2);
   currentN++;
}
// sumOfSquares now has the sum of all squares from 1 to 100
```

c)
```
int sumOfOddNumbers = 0;
for (int n = a; n <= b; n++)
{
   if (n % 2 == 1)
   {
      sumOfOddNumbers = sumOfOddNumbers + n;
   }
}
//sumOfOddNumbers now has the value of all odd numbers between a and b
```

d)

```
int nLeft = n;
int sumOfOddDigits = 0;
while (nLeft > 0)
{
   int digit = nLeft % 10;
   if (digit % 2 == 1)
   {
      sumOfOddDigits = sumOfOddDigits + digit;
   }
   nLeft = nLeft / 10;
}
// sumOfOddNumbers now has sum of all odd digits in n
```

R6.6

a)

i	j	n
0	10	0
1	9	1
2	8	2
3	7	3
4	6	4
5	5	5

b)

i	j	n
0	0	0
1	1	1
2	2	4
3	3	9
4	4	16
5	5	25
6	6	36
7	7	49
8	8	64

9	9	81
10	10	100

c

i	j	n
10	0	0
9	1	8
8	2	14
7	3	18
6	4	20
5	5	20
4	6	18
3	7	14
2	8	8
1	9	0
0	10	-10

d)
As the trace table shows, `i` will never equal `j` since they pass each other at the 4-6 to 6-4 mark. The trace table stops right below this point to indicate it is an infinite loop.

i	j	n
0	10	0
2	8	1
4	6	2
6	4	3
8	2	4
…	…	…

R6.7

a) `1 2 3 4 5 6 7 8 9`

b) `1 3 5 7 9`

`c) 10 9 8 7 6 5 4 3 2`

d) `0 1 2 3 4 5 6 7 8 9`

e) `1 2 4 8`

f) `2 4 6 8`

R6.8

An infinite loop occurs when the terminating condition in the loop never evaluates to `false`. How you stop an infinite loop on your computer depends on your operating system and your IDE. For example, in the Eclipse IDE there is a red “TERMINATE” button that will stop any Java program at any time.

R6.9

first	value	minimum	output
~~true~~			
false	4	4	
	~~7~~		
	~~-2~~	~~-2~~	
	~~-5~~	~~-5~~	
	0	-5	-5

R6.10

An off-by-one error occurs when you incorrectly initialize or terminate a loop by one unit. A common example occurs when looping through a Java `String`. The programmer may forget that character positions start at 0 rather than 1 and start his loop one character beyond the actual start of the `String`.

R6.11

A sentinel value is a particular value that is used to indicate the end of a sequence. By necessity the sentinel cannot be a valid member in the sequence. For example, an employer may be interested in the average age of her employees. She could write a program that reads a series of ages and terminates when it reads a negative number. Since there are no employees with a negative age it makes an appropriate sentinel value.

R6.2

1
Java supports three types of loop statements: `for`, `do`, and `while`. Use a `for` loop if you know in advance how many times the loop should be repeated. Use a `do` loop if the loop must be executed at least once. Otherwise, use a `while` loop.

R6.13

1
a) 10 times

b) 10 times

c) 10 times

d) 21 times

e) This is an infinite loop

f) 11 times

g) 7 times

R6.14

```
Print out calendar header with days of week: Su M T W Th F Sa

// The -2 indicates that the first week doesn't start until Wednesday
current day = -2
last day of month = 31

// Let 0 represent Su and 6 represent Sa
weekday = 0
while (current day <= last day of month)
   if (current day > 0)
      Print current day followed by a space
   else
      Print three spaces
   Add one to current day
   Add one to weekday

   // Check to see if we've printed out a week, print a newline if we have
   if (weekday == 7)
      Print a newline to go to next week
      weekday = 0
```

R6.15

1

```
Print out table header
for (celsius = 0; celsius <= 100; celsius = celsius + 10)
   Print right aligned celsius
   Print a | character
   fahrenheit = (9/5) * celsius + 32
   Print right aligned Fahrenheit
   Print newline
```

R6.16

```
// Initialize current name to something other than -1
current name = ""
read name from input into current name
counter = 0
total score = 0
average score = 0
read next input into current score
while (current score != -1)
   counter = counter + 1
   add current score to total score
   read next input into current score
average score = total score / counter
print average score
print newline
```

Trace Table:

name	current	counter	total	average

	score		score	score
Harry Morgan	94	1	94	0
	71	2	165	0
	86	3	251	0
	95	4	346	0
	-1			0
				86.5

R6.17

```
// Initialize current name to something other than END
current name = “”
read name from input into current name
while (current name != “END”)
   current score = 0
   total score = 0
   read next input into current score
   while (current score != -1)
      add current score to total score
      read next input into current score
   print current name and total score
   print newline
```

Trace Table:

name	current score	total score
Harry Morgan	94	94
	71	165
	86	251
	95	346
	-1	
Sally Lin	99	99
	98	197
	100	297
	95	392
	90	482

	-1	

R6.18

```
int s = 0;
int i = 1;
while (i <= 10)
{
   s = s + i;
   i++;
}
```

R6.19

```
int n = in.nextInt();
double x = 0;
double s;
s = 1.0 / (1 + n * n);
n++;
x = x + s;
while (s > 0.01)
{
   s = 1.0 / (1 + n * n);
   n++;
   x = x + s;
}
```

R6.20

a)

s	n
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	2

b)

s	n
1	1
2	2
4	3
7	4
11	

c)

s	n
1	1
2	2
4	3
7	4
11	5
16	6
22	7
29	8
37	9
46	10
56	11
67	12
79	13
92	14
106	15
121	16
137	17
154	18
172	19
191	20
211	21

R6.21

a) `2 4 7 11 16`

b) `4 9 16`

c) `10 7`

R6.22

a) `10`

b) `6`

c) `3`

d) `0`

R6.23

If a programmer needed to print a list of numbers starting from `a` to `b` it is easier to read if the `for` loop has `a` and `b` as symmetric bounds, like this:

```
for (int i = a; i <= b; i++)
```

If a programmer needed to loop through all the characters in a string, it is easier to read if the loop has asymmetric bounds based on the length of the string, like this:

```
for (int i = 0; i < myString.length(); i++)
```

R6.24

Storyboard for incompatible units.

```
What unit do you want to convert from? cm
What unit do you want to convert to? gal
Sorry, cannot convert from one unit to the other.
What unit do you want to convert from? cm
What unit do you want to convert to? in
Enter values, terminated by zero
10
10 cm = 3.94 in
0
What unit do you want to convert from?
```

R6.25

Storyboard showing how valid unit types can be displayed.

```
What unit do you want to convert from? leagues
Sorry, invalid units.  Valid unit types are:
          in, ft, mi, mm, cm, m, km
          oz, lb, g, kg
          tsp, tbsp, pint, gal, ltr
What unit do you want to convert from? cm
What unit do you want to convert to? in
Enter values, terminated by zero
10
10 cm = 3.94 in
0
What unit do you want to convert from?
```

R6.26

Storyboards from Section 6.6 with a new menu.

```
Converting a Sequence of Values
What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
1
What unit do you want to convert from? cm
What unit do you want to convert to? in
Enter values, terminated by zero
30
30 cm = 11.81 in
100
100 cm = 39.37 in
0

What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
```

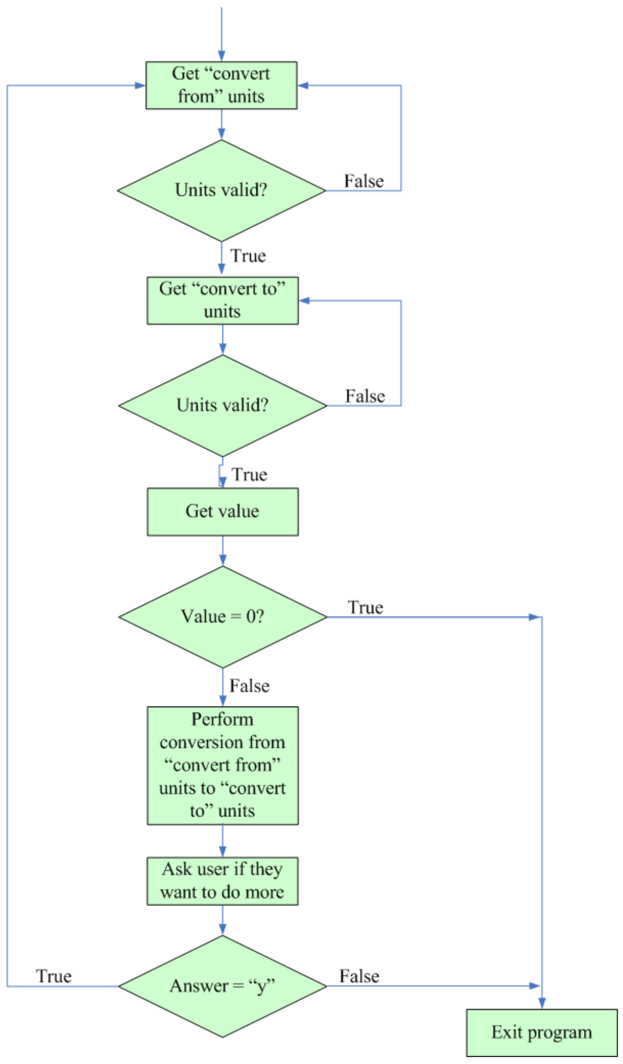
Handling Unknown Units

```
What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
1
What unit do you want to convert from? cm
What unit do you want to convert to? inches
Sorry, unknown unit.

What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
1
What unit do you want to convert from? cm
What unit do you want to convert to? inch
Sorry, unknown unit.

What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
1
What unit do you want to convert from? cm
What unit do you want to convert to? in
30
30 cm = 11.81 in
100
100 cm = 39.37 in
0

What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
```

```
Exiting the Program
What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
1
What unit do you want to convert from? cm
What unit do you want to convert to? in
Enter values, terminated by zero
30
30 cm = 11.81 in
100
100 cm = 39.37 in
0

What would you like to do?
   Enter 1 to convert units
   Enter 2 for program help
   Enter 3 to quit
3
```

R6.27

Flowchart for the units conversion program in Section 6.6.



R6.28

You cannot initialize the `largest` and `smallest` variables to zero because it is possible that all of the values entered may be either all larger than zero or all smaller than zero.

For example, if we are looking for the maximum and set `largest` to zero, and the values entered are -10.2, -8.1, and -7.6, the value of maximum never changes from zero, because none of the values entered is greater than zero. The maximum number of those entered should be -7.6, but the program would think it was 0.

R6.29

Nested loops are loops contained in other loops. This sort of structure is common when having to process a table. One loop would process the rows while the other would process the columns in each row.

R6.30

```
for (int i = 1; i <= width * height; i++)
{
   System.out.print("*");
   if (i % width == 0)
   {
      System.out.println();
   }
}
```

R6.31

Java has a random number generator (`Math.random()`) which generates random numbers from 0 up to 1 (exclusive). To generate a random hour, get a random number between 0 up to 1, multiply it by 12 and cast it to an `int`; this gives you a random number between 0 and 11. Next, add 1 to that number to create a number between 1 and 12 which is appropriate for the hour.

The minutes are easier, simply generate a random number between 0 up to 1, multiply it by 60 and cast it to an `int`.

R6.32

This problem is easier to think about if you generate a random number to select which friend to visit. First generate a random number between 1 and 15 (Harry has 15 total friends). Next convert the friend numbers to state numbers. If the number is between 1 and 10, generate a 1 indicating California; if the number is between 11 and 13, generate a 2 indicating Nevada; otherwise generate a 3 indicating Utah.

R6.33

- **Step into:** steps inside method calls. You should step into a method to check whether it carries out its job correctly.

- **Step over:** skips over method calls. You should step over a method if you know it works correctly.

R6.34

The procedure depends on the debugger. Most debuggers know about the `String` class and display its contents immediately when you ask to inspect or watch a string. However, you can also display the instance variables and inspect the `value` instance variable.

R6.35

3This varies according to the debugger used. Typically, you inspect a variable of type `Rectangle` and carry out some action (such as clicking on a tree node) to open up the display of the instance variables. The `x`, `y`, `width`, and `height` instance variables should then be visible.

R6.36

This varies according to the debugger used. Typically, you inspect a variable of type `BankAccount` and carry out some action (such as clicking on a tree node) to open up the display of the instance variables. The `balance` variable should then be visible.

R6.37

The “divide-and-conquer” strategy involves stepping over the methods in `main` to pinpoint the location of the failure. When a failure occurs just after a particular method (say `f`), then restart `main`, set a breakpoint before `f`, step into `f`, and now apply the same strategy to `f`. Keep going until the error is found.

Chapter-7

R7.1

a)
```
for (int i = 0; i < 10; i++)
{
   values[i] = i + 1;
}
```

b)
```
for (int i = 0; i <= 10; i++)
{
   values[i] = i * 2;
}
```

c)
```
for (int i = 0; i < 10; i++)
{
   values[i] = (i + 1) * (i + 1);
}
```

d)
```
for (int i = 0; i < 10; i++)
{
   values[i] = 0;
}
```

e)
```
int[] values =  {1, 4, 9, 16, 9, 7, 4, 9, 11};
```

f)
```
for (int i = 0; i < 10; i++)
{
   values[i] = i % 2;
}
```

g)
```
for (int i = 0; i < 10; i++)
{
   values[i] = i % 5;
}
```

R7.2

a) 25

b) 13

c) 12

d) This loop condition causes an out-of-bounds index error. If the condition were `i < 10`, `total` would be 22.

e) 11

f) 25

g) 12

h) -1

R7.3

a) `{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }`

b) `{ 1, 1, 2, 3, 4, 5, 4, 3, 2, 1 }`

c) `{ 2, 3, 4, 5, 4, 3, 2, 1, 0, 0 }`

d) `{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }`

e) `{ 1, 3, 6, 10, 15, 19, 22, 24, 25, 25 }`

f) `{ 1, 0, 3, 0, 5, 0, 3, 0, 1, 0 }`

g) `{ 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 }`

h) `{ 1, 1, 2, 3, 4, 4, 3, 2, 1, 0 }`

R7.4

To generate an array with ten random numbers between 1 and 100:

```
for (int i = 0; i < 10; i++)
{
   values[i] = (int) (Math.random() * 100 + 1);
}
```

To generate an array with ten different random numbers between 1 and 100:

```
for (int i = 0; i < 10; i++)
{
   values[i] = (int) (Math.random() * 100 + 1);

   // Check to see if number has been generated before
   int count = 0;
   for (int j = 0; j < i; j++)
   {
      if (values[j] == values[i])
      {
         count++;
      }
   }
   if (count > 0)
   {
      i--; // if it has, back up and try again
   }
}
```

R7.5

Below, `data` is an array.

```
int min = data[0];
int max = data[0];
for (Integer val : data)
{
   if (val > max)
   {
      max = val;
   }
   if (val < min)
   {
      min = val;
   }
}
```

R7.6

a) The loop iterates over the values 1 to 10 which in turn are used as array indices. Yet, the length of the array is 10, thus its indices range from 0 to 9.

b) The programmer forgot to initialize the array values. One would expect a statement like `int[] values = new int[10];` or however many elements the programmer desired.

R7.7

a)
```
for (Object object : array)
{
   System.out.print(object.toString());
}
```

b)
```
int max = array[0];
for (Object object : array)
{
   if (object > max)
   {
      max = object;
   }
}
```

c)
```
int count = 0;
for (Object object : array)
{
   if (object < 0)
   {
      count++;
   }
}
```

R7.8

a)
```
for (int i = 0; i < values.length; i++)
{
   total = total + values[i];
}
```

b)
```
for (int i = 0; i < values.length; i++)
{
   if (values[i] == target)
   {
      return true;
   }
}
```

c)
```
for (int i = 0; i < values.length; i++)
{
   values[i] = 2 * values[i];
}
```

R7.9

a)
```
for (double x : values)
{
   total = total + x;
}
```

b)
```
int pos = 0;
for (double x : values)
{
   if (pos >= 1)
   {
      total = total + x;
   }
   pos++;
}
```

c)
```
int pos = 0;
for (double x : values)
{
   if (x == target)
   {
      return pos;
   }
   pos++;
}
```

R7.10

a) `ArrayList` can't take a Java primitive as a type, rather it needs a wrapper class, in this case `Integer`.

b) In this case the new `ArrayList` did not define its generic type. One would expect to see new `ArrayList<Integer>();` as the initial value. In Java 7, however, the syntax shown is acceptable.

c) Parentheses () are required after `new ArrayList<Integer>`.

d) The initial size of the `ArrayList` already is 0 elements, thus the `set` method cannot be used to set them.

e) The `ArrayList` reference `values` is not initialized. The code `= new ArrayList<Integer>();` should be to the right of the declaration.

R7.11

The index of an array specifies the array position in which the element can be read or set.

Legal index values start at 0 and go to the length of the array minus one.

A bounds error occurs when an array is referenced with an index that is outside the range of legal array index values.

R7.12

The following is a program that contains a bounds error.

```
public class R6_12
{
   public static void main(String[] args)
   {
      int[] array = new int[10];
      array[10] = 10;
   }
}
```

When run it outputs the following error:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
 at R6_12.main(R6_12.java:6)
```

R7.13

```
Scanner in = new Scanner(System.in);
int myArray[10];
for (int i = 0; i < 10; i++)
{
   System.out.println("Enter a number: ");
   myArray[i] = in.nextInt();
}

for (int j = 9; j >= 0; j--)
{
   System.out.print(myArray[j] + " ");
}
```

R7.14

The headers of the described methods.

a) `public void sortDecreasing(int[] arr, int currentSize)`

b) `public void printWithString(int[] arr, int currentSize, string sep)`

c) `public int numItemsBelow(int[] arr, int currentSize, int value)`

d) `public void removeItemsLessThan(int[] arr, int currentSize, int value)`

e) `public void copyLessThan(int[] source, int firstSize, int[] target, int secondSize, int value)`

R7.15

i	output
0	32
1	| 54
2	| 67.5
3	| 29
4	| 35

R7.16

element	matches
110	{110}
90	{110}
100	{110}
120	{110, 120}
80	{110, 120}

R7.17

pos	found
0	false
1	false
2	false
3	true

pos	found
0	false
1	false
2	false
3	false
4	false
5	true

R7.18

currentSize	i	values
5		{110, 90, 100,120,80}
5	3	{110, 90, 120,120,80}
5	4	{110, 90, 120,80,80}
4	5	{110, 90, 120,80,80}

R7.19

```
Copy the first element into a variable, x.
Loop through the first length – 2 positions of the array using pos as a counter
Copy the element at pos + 1 to the element at pos.
Copy x into the last position of the array.
```

R7.20

```
pos = 0
while pos < length of array
if the element at pos is negative
```

```
            remove the element at pos.
        else
            increment pos.
```

R7.21

2

```
// x is the element to insert
pos = 0
inserted = false
while pos < length of array and not inserted
    if the element at pos is less than x
        insert x at pos.
        inserted = true.
    else
        increment pos.
if not inserted
    insert x at the end of the array.
```

R7.22

```
longestRun = 0
pos = 0
while pos < length of array - 1
    if element at pos is equal to element at pos + 1
        currentRun = 0
        currentElement = element at pos
        while pos < length of array - 1 and currentElement equals element at pos
            increment pos
            increment currentRun
        if currentRun > longestRun
            longestRun = currentRun
    else
        increment pos
print longestRun
```

R7.23

The `values` variable is a reference to another array. Setting it equal to `numbers` causes the `values` reference to point to the `numbers` array, but it does not change the value of the object originally referred to by `values`.

R7.24

To find the smallest rectangle containing the points defined by the two arrays, compute the max and min of the x-values and the max and min of the y-values. With those four values, you can define the locations of the x- and y-coordinates of the rectangle as follows:

Upper left corner: (min x, max y)
Lower left corner: (min x, min y)
Upper right corner: (max x, max y)
Lower right corner: (max x, min y)

R7.25

If the array is already sorted, we no longer need to find the position of the lowest quiz score. If the array is sorted from lowest to highest, then the lowest quiz score is in the first position in the array. Technically, we can even compute the sum without actually removing the lowest element, because it is always in the first location. We can find the sum simply by starting the loop at position 1 instead of position 0:

```
public double sum(double[] values)
{
   double total = 0.0;
   // Start loop count at 1 to skip lowest score
   for (int n = 1; n < values.length; n++)
   {
      total = total + values[n];
   }
   return total;
}
```

R7.26

Pseudocode for an algorithm using “removes” and “inserts” instead of swaps.

```
i = 0
j = size / 2
while (i < size / 2)
```

```
Remove item at position j.
Insert removed item at position i.
i++
j++
```

I will try this algorithm out with four coins, a penny (P), a nickel (N), a dime (D), and a quarter (Q).

```
Position    0  1  2  3
            P  N  D  Q
i = 0, j = 2, Remove coin at position j, insert at position 0
Increment i and j.


Position    0  1  2  3
            D  P  N  Q
i = 1, j = 3, Remove coin at position j, insert at position 1
Increment i and j.


Position    0  1  2  3
            D  Q  P  N
i = 2, j = 4.
```

The reason this is always less efficient that using the swapping algorithm is that our new remove/insert algorithm above requires two method calls (one for the remove, and one for the insert) each time a coin is removed. In the swapping algorithm, one method call (the swap) moves both coins at the same time.

Another way to look at it is that each time you remove a coin and each time you insert one, you have to move elements in the array to new locations. (If you remove an element, all the elements above it must move down one. If you insert, all the elements above it must move up one.) When you do this as a single swap operation, only the two elements affected are moved. The other elements do not have to move during a swap.

R7.27

When I laid the coins and paper clips out as described in this problem (a series of coins each with a number of paperclips beneath it), I envisioned a two-dimensional array with N rows and 2 columns. The first column held the value of the coin (whether it was a penny, nickel, dime, or quarter) and the second column held a cumulative count corresponding to the number of that type of coin in column 1. For instance, if I had a total of 10 coins consisting of 3 pennies (P), 2 nickels (N), 4 dimes (D), and 1 quarter (Q), my initial two-dimensional array might look like this (with all of the “counts” in the second column set to zero):

D	0
D	0
P	0
Q	0
P	0
N	0
D	0
N	0
D	0
P	0

Then my algorithm for counting the coins would use a nested loop like this:

```
Loop (int i = 0; i < 10; i++)
  Loop (int j = 0; j < 10; j++)
    if coins[j][0] equals coins[i][0]
      Increment the count at location coins[j][i].
  End inner loop.
End outer loop.
```

In other words, you look at the coin in the first location (a dime), and then walk through the two-dimensional array in the inner for loop, incrementing the count of any dime you find in the array. Then you look at the second coin (also a dime), and loop through the inner for loop again, incrementing the count for any dimes you find, and so on.

When you have walked through the entire array in this manner, your final two-dimensional array should look like this:

D	4
D	4
P	3
Q	1
P	3
N	2
D	4
N	2
D	4
P	3

Then it is a simple matter of looping through the array, using the algorithms given in this chapter, to find the position of the element with the maximum count in the second column.

R7.28

```
for (int i = 0; i < ROWS; i++)
{
   for (int j = 0; j < COLUMNS; j++)
   {
      values[i][j] = 0;
   }
}

for (int i = 0; i < ROWS; i++)
{
   for (int j = 0; j < COLUMNS; j++)
   {
      values[i][j] = (i + j) % 2;
   }
}

for (int j = 0; j < COLUMNS; j++)
{
   values[0][j] = 0;
   values[ROWS - 1][j] = 0;
}

int sum = 0;
for (int i = 0; i < ROWS; i++)
{
   for (int j = 0; j < COLUMNS; j++)
   {
      sum = sum + values[i][j];
   }
}

for (int i = 0; i < ROWS; i++)
{
   for (int j = 0; j < COLUMNS; j++)
   {
      System.out.print(values[i][j] + " ");
   }
   System.out.println();
}
```

R7.29

Assume the two-dimensional array, `data`, has M rows and N columns:

```
Loop from row=0 to row=M
   Loop from column=0 to column=N
      If row is 0 or M
```

```
Set data[row][column] equal to -1.
      Else if column is 0 or N
Set data[row][column] equal to -1.
          End inner loop.
            End loop.
```

R7.30

When traversing backward, the indices of the array list have to be decremented (instead of incremented) in the for loop. This process continues while the index `i` is greater than zero (0). When an object is removed from the array only the indices of the objects greater than the index removed are effected. The loop is only working with the indices less than the index of the object removed, so there is no issue.

R7.31

a) True.

b) False.

c) False.

d) False.

e) False.

f) True.

g) True.

R7.32

a) Verify that the lengths are equal, and then set a Boolean false if any mismatch occurs.

```
boolean equal = true;
if (data1.size() != data2.size())
{
   equal = false;
}
for (int i = 0; i < data1.size() && equal; i++)
```

```
{
   if (data1.get(i) != data2.get(i))
   {
      equal = false;
   }
}
```

b) Create a second array list that is empty, then loop over the first:

```
ArrayList<Integer> data2 = new ArrayList<Integer>();
for (int i = 0; i < data1.size(); i++)
{
   data2.add(data1.get(i));
}
```

c) Loop over all positions in the array list and set each one to 0:

```
for (int i = 0; i < data.size(); i++)
{
   data.set(i, 0);
}
```

d) Loop while the array list still has elements in it and remove the element at position 0:

```
while (data.size() > 0)
{
   data.remove(0);
}
```

More simply, one can call `ArrayList`'s `clear` method:

```
data.clear();
```

R7.33

a) True

b) True

c) False

d) True

e) False

f) False.

R7.34

Regression testing is testing done to ensure changes to code haven't broken any previously existing functionality. A test suite is a set of tests for the code that check if existing functionality works.

R7.35

Debugging cycling is a phenomenon where an issue is resolved that is later seen again after another issue's fix breaks the original fix for the resurfaced issue.

Chapter -8

R8.1

VendingMachine would be an appropriate name for the class because it a) simulates a vending machine, and b) has functionality that matches vending machine functionality exactly, without more or less functions.

PriceChecker would not be an appropriate name for the class because while it does check the price, it also performs a transaction.

ChangeGiver would not be an appropriate name for the class because while it does give change when too much money is given, it will return the money paid if there is not enough.

R8.2

a) Invoice would be a good class to use if the goal was to keep track of the invoice as a digital record. The application only takes the information in to print the invoice, so this could be misleading.

b) InvoicePrinter is a good class to use for this application, because it describes exactly the functionality of the application.

c) PrintInvoice works as a description of the application, but might not be the best choice because it sounds more like a function or an action than an entire class.

d) InvoiceProgram would be a good class to use if there were more than one action the application could perform. Using the word "program" implies that there are multiple possible functions of the application, whereas one could more easily use the single action provided by the application in its name.

R8.3

Actor Class – PaycheckComputer

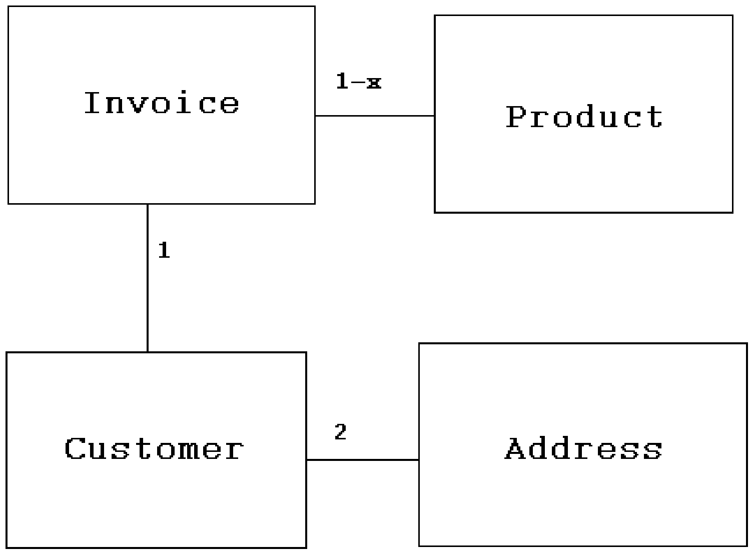
Non-Actor Class – Payroll

The difference between the two classes if the implied functionality. The actor class defines its functional capacity of the class within its name, whereas the non-actor class only provides the general description of the functionality.

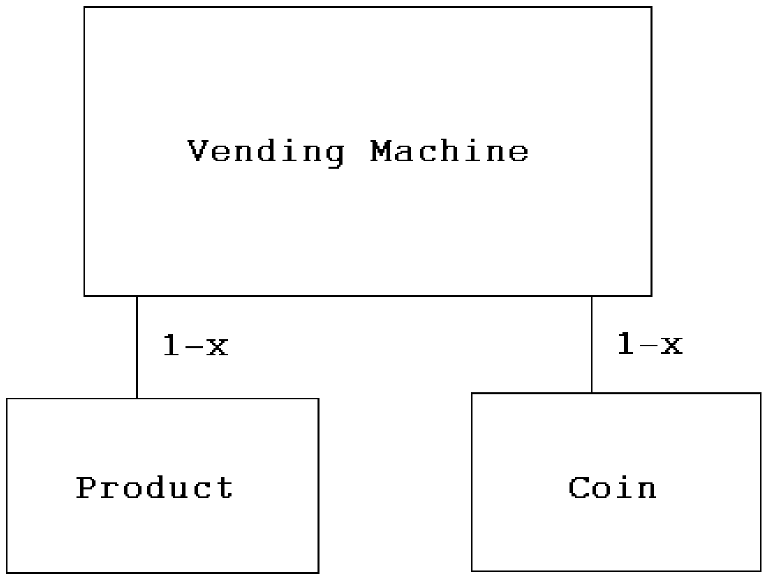
R8.4

The System class is not cohesive because it includes many features that are unrelated, such as console I/O, garbage collection, and the time in milliseconds.

R8.5


Invoice
1-x
Product
1
Customer
2
Address

R8.6


Vending Machine
1-x
1-x
Product
Coin

R8.7

The Integer class depends on the String class.

R8.8

The class Rectangle depends on the Rectangle2D, Point, Dimension, and String classes.

R8.9

boolean hasNext() – Accessor
boolean hasNextDouble() – Accessor
boolean hasNextInt() – Accessor
boolean hasNextLine() – Accessor
String next() – Mutator
double nextDouble() – Mutator
int nextInt() – Mutator
String nextLine() – Mutator

R8.10

Accessor methods:

contains
createIntersection
createUnion
equals
getBounds
getBounds2D
getHeight
getLocation
getSize
getWidth
getX
getY
intersection
intersects

isEmpty
outcode
toString
union

Mutator methods:

add
grow
setBounds
setLocation
setRect
setSize
translate

R8.11

The Resistor class in problem P8.8 is immutable because all of its functions are accessors.

R8.12

Of the three class, Rectangle, String, and Random, only class String is immutable.

R8.13

Of the three classes PrintStream, Date, and Integer, the Integer class is immutable.

R8.14

The side effect of the read() method is that it modifies the Scanner parameter that is passed in. A redesign would be to use the Scanner object within a main() method that passes in strings one at a time to the DataSet object.

R8.15

public void print() The side effect is modifying System.out to print on the console.
public void print(PrintStream stream)The side effect is modifying the stream parameter.
public String toString()There is no side effect.

R8.16

If no function (including main) has a side effect, then you could not observe the program doing anything. Such a program would be useless. (Note that producing output on the screen is a side effect.)

R8.17

When you call falseSwap, then a is initialized with 3 and b is initialized with 4. At the end of the falseSwap method, a is 4 and b is 3. Then the method exits and its local variables (including the parameter variables) are forgotten. The contents of x and y are not affected.

R8.18

```
public static void swap(Point2D.Double p)
{
   p.setLocation(p.getY(), p.getX());
}

public static void main(String[] args)
{
   double x = 3;
   double y = 4;
   Point2D.Double p = new Point2D.Double(x, y);
   swap(p);
   x = p.getX();
   y = p.getY();
   System.out.println(x + " " + y);
}
```

R8.19

TODO

R8.20

TODO

R8.21

We get the error message "non-static method print(int) cannot be referenced from a static context" because the print method is not declared as static. If you change the header of the method to public static void print(int x), then the program will work. The reason the method needs to be declared as static is because we are calling it without an object reference (implicit parameter), but only static methods can be called like that.

R8.22

decode
getInteger
highestOneBit
lowestOneBit
numberOfLeadingZeros
numberOfTrailingZeros
parseInt
reverse
reverseBytes
rotateLeft
rotateRight
signum
toBinaryString
toHexString
toOctalString
toString // all of the toString variations, except toString()
valueOf

They are static methods because these methods do not need an implicit parameter.

R8.23

All of the valueOf methods in the String class are static methods. Like the methods in the Integer class, these methods are static because these methods do not operate on an object and have only explicit parameters. They create a new String instead of modifying an existing String (implicit parameter, which these methods do not need). The format methods are also static, for the same reason.

R8.24

It is not a good design because using public static variables is not a good idea; they can accidentally get overwritten in large programs. A better way to do this is to have static methods System.getIn() and System.getOut() that return these streams.

R8.25

To write a Java program without import statements, the user needs to specify the path names of the classes that are used in the program.

```
/**
  A component that draws two rectangles.
*/
public class RectangleComponent extends javax.swing.JComponent
{
  public void paintComponent(java.awt.Graphics g)
  {
    // Recover Graphics2D
    java.awt.Graphics2D g2 = (java.awt.Graphics2D) g;

    // Construct a rectangle and draw it
    java.awt.Rectangle box = new java.awt.Rectangle(5, 10, 20, 30);
    g2.draw(box);

    // Move rectangle 15 units to the right and 25 units down
    box.translate(15, 25);

    // Draw moved rectangle
    g2.draw(box);
  }
}
```

R8.26

The default package is the package that contains the classes with no package specifier. All classes that we have programmed up to this point were in the default package.

R8.27

The exception is reported, and the remaining methods continue to be executed. This is an advantage over a simple tester class whose main method would terminate when an exception occurs, skipping all remaining tests.

Chapter 9: Review Exercise Solutions

R9.1

a.
The subclasses of the class Employee can be found from the class diagram shown in the step 2. The classes HourlyEmployee and SalariedEmployee are derived from the class Employee and the class Manager is derived from the class SalariedEmployee.
Hence, the subclasses of the class Employee are HourlyEmployee, SalariedEmployee and Manager.
b.
The superclass is a class from which one or more classes are inherited. This class can also be called as parent class. The class Manager is inherited from the class SalariedEmployee and SalariedEmployee is inherited from the class Employee as can be seen in the figure in step 2.
Hence, the super classes of the class Manager are SalariedEmployee and Employee.
c.
The super class is the class from which many subclasses can be inherited and a subclass is a class which is derived from some other class. The subclass can also be called a child class.
The super class of the class SalariedEmployee is Employee and subclass of the class SalariedEmployee is Manager as can be seen in the figure given in step 2.

d.
The subclasses or child class inherits the properties or methods of the parent class. A method with the same name in the child class as of the parent class is called the overridden method.
The method weeklyPay is in the parent class Employee and can be seen in the classes HourlyEmployee, SalariedEmployee, and Manager which can be seen from the class diagram given in step 6.

Hence, the method weeklyPay of the class Employee is overridden in the classes HourlyEmployee, SalariedEmployee, and Manager.

e.
The method setName in the class Employee is a setter or mutator method which sets the value of the member variable name of the class Employee. Since it is a member function of the class Employee and don't have any need to be overridden in the subclasses of the class Employee.

Hence, the method setName is not overridden in any of the subclasses of the class Employee.
f.
The instance variable of a class is the variable declared in the class outside of any constructors, methods, or blocks. These variables can be accessed within any constructor, methods, or blocks of the same class. The instance variables will be created when an object of the class is instantiated or created.
The instance variable of the class HourlyEmployee is hourlyWage which can be seen in the class diagram given in the step 6.
Hence, the instance variable of an object of the class HourlyEmployee is hourlyWage.

R9.2

a)
Superclass: `Employee`
Subclass: `Manager`
b)
Superclass: `Student`
Subclass: `GraduateStudent`
c)
Superclass: `Person`
Subclass: `Student`
d)
Superclass: `Employee`
Subclass: `Professor`
e)
Superclass: `BankAccount`
Subclass: `CheckingAccount`
f)
Superclass: `Vehicle`
Subclass: `Car`
g)
Superclass: `Vehicle`
Subclass: `Minivan`
h)
Superclass: `Car`
Subclass: `Minivan`
i)
Superclass: `Vehicle`
Subclass: `Truck`

R9.3

Because in terms of inventory, toasters, car vacuums, and travel irons all behave the same. There are a certain number of them and they cost a certain amount.

R9.4

`ChoiceQuestion` inherits the following methods from its superclass:

- `setText`
- `setAnswer`
- `checkAnswer`

It overrides the following method:

- `display`

And it adds the following methods:

- `addChoice`

R9.5

`SavingsAccount` inherits the following methods from its superclass:

- `deposit`
- `getBalance`

It overrides the following methods:

- `withdraw`
- `monthEnd`

It adds the following method:

- `setInterestRate`

R9.6

`CheckingAccount` has a single instance variable:

- `private int withdrawals;`

R9.7

a) legal
b) not legal
c) not legal
d) legal

R9.8Answer
Consider the following inheritance relationship classes:
Person
Employee
Student
Instructor
Classroom
Object

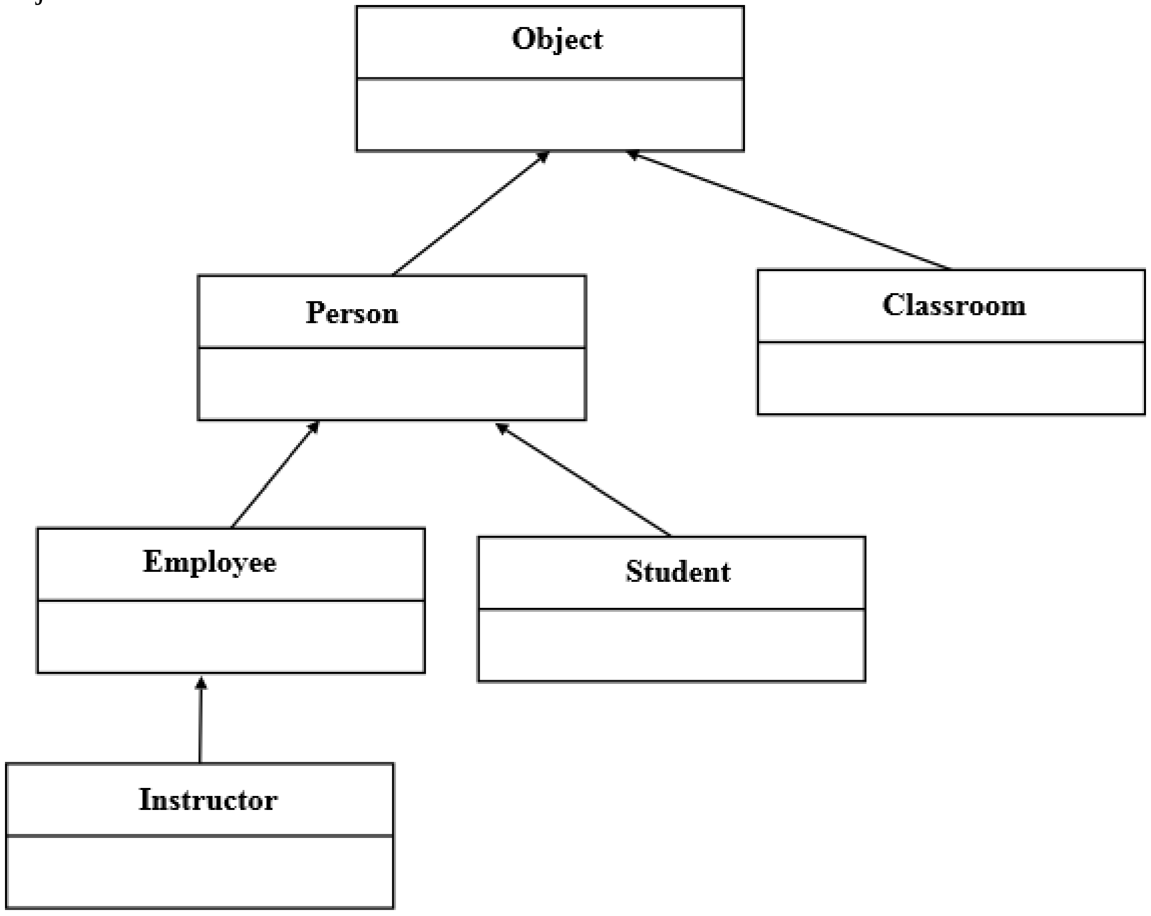


Figure 1: Inheritance diagram that shows the relationship between the classes.

R9.9 Consider the following relationship classes between in an object-oriented traffic simulation system:

Vehicle
Car
Truck
Sedan
Coupe
PickupTruck
SportUtilityVehicle

Minivan
Bicycle
Motorcycle

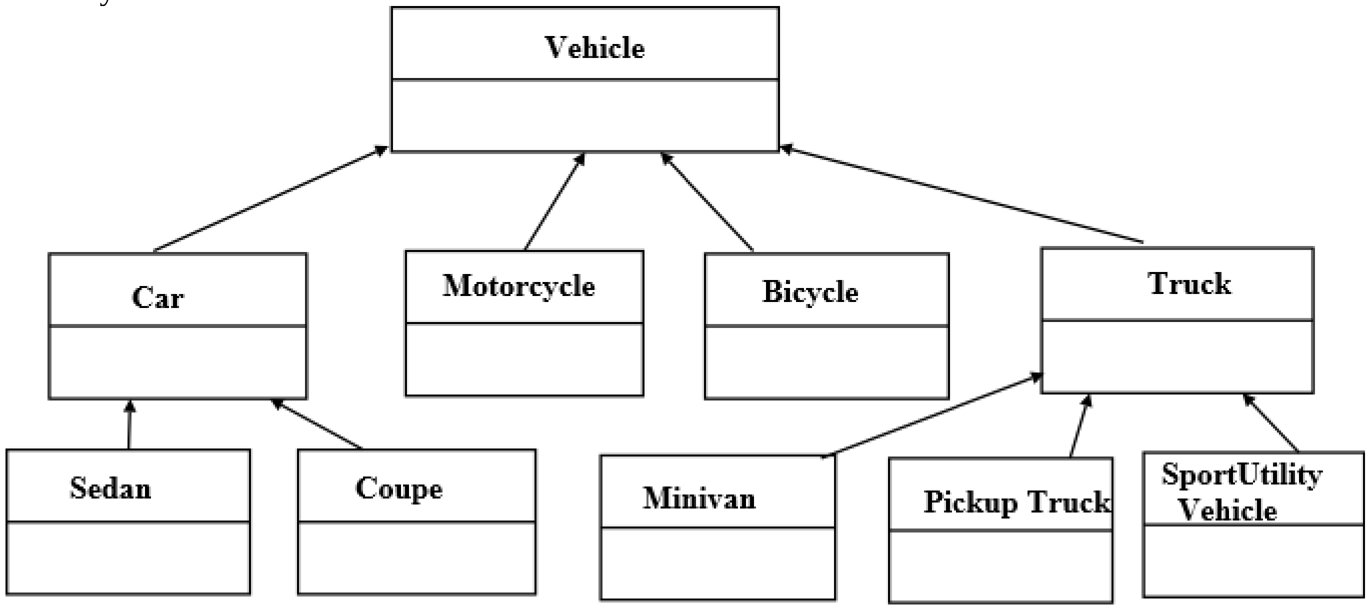


Figure 2: Inheritance diagram that shows the relationship between the classes.

R9.10. Consider the following inheritance relationship classes:

Student
Professor
TeachingAssistant
Employee
Secretary
DepartmentChair
Janitor
Lecture
ComputerLab

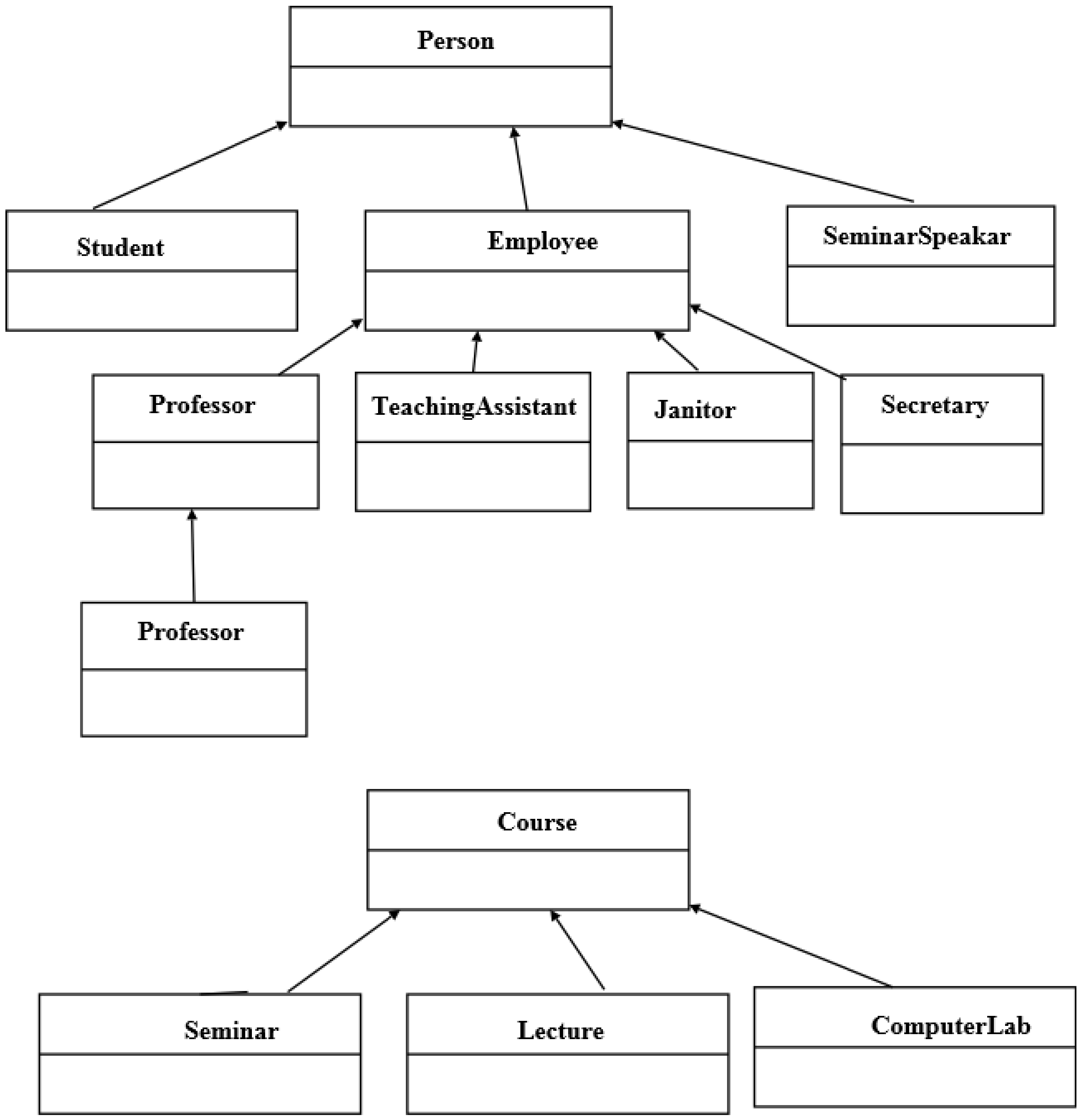


Figure 3: Inheritance diagram that shows the relationship between the classes.

R9.11

The `(BankAccount) x` cast is casting the variable type of the object without changing its value whereas the `(int) x` cast can actually change the underlying primitive value.

R9.12

a) true
b) true
c) false
d) true
e) false
f) false

Chapter 10:

R10.1

The following require a cast:

```
c = i; // c = (C) i;
i = j; // i = (I) j;
```

R10.2

None of them will throw an exception.

R10.3

The following are legal:

a. `e = sub;`
c. `sub = (Sandwich) e;`

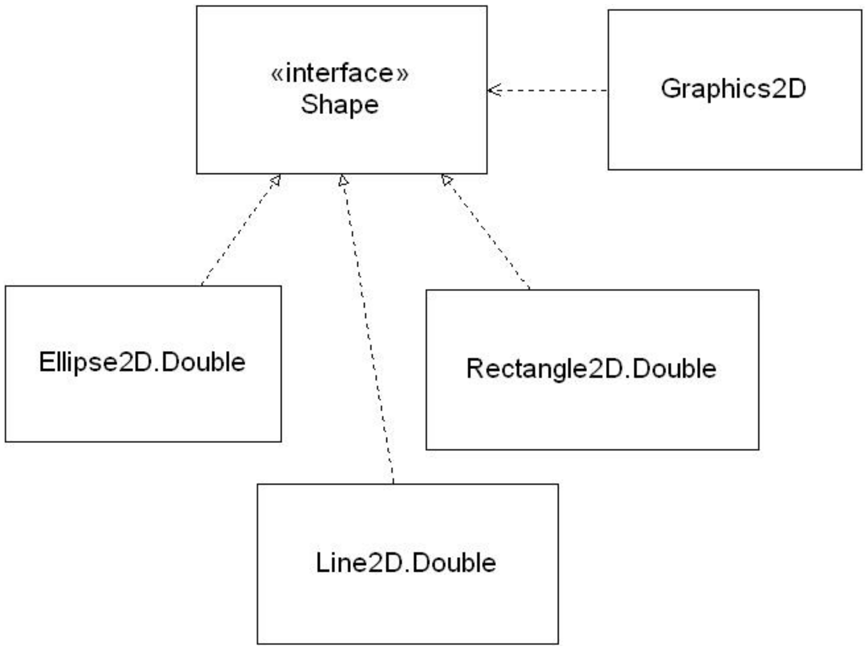
The following statement will compile but throw an exception at run time.

f. `e = (Edible) cerealBox;`

R10.4

When casting a primitive type, the user acknowledges to the compiler that information may be lost. When casting objects, the user acknowledges to the compiler that the cast may cause an exception.

R10.5



R10.6

a. `Rectangle a = r;`
b. `Shape b = r;`
f. `Serializable f = r;`
g. `Object g = r;`

R10.7

The call `Rectangle r = s.getBounds();` is an example of polymorphism because the `Shape` interface declares the method, and each class that implements the interface (`Rectangle2D.Double`, `Ellipse2D.Double`, `Line2D.Double`, etc.) provides its own implementation of the method. The correct implementation is picked at run time.

R10.8

The Employee class must
1) implement Measurable and
2) include a method
    public double getMeasure()

To use the second solution in 10.4, you
1) create an interface Measurer that defines a callback method (measure) and
2) you create a small helper class with the method that tells the average method how to measure the objects
    public double measure(Object anObject)

The callback method decouples your class with other classes. It is a more generic and flexible solution . It is no more difficult to implement than a pure interface solution.

R10.9

You will get a compiler error. The String class does not implement the Measurable interface.

R10.10

A callback is objtained by implementing the Measurer interface. Provide a class StringMeasurer that implements Measurer.

```
public class StringMeasurer implements Measurer
{
   public double measure(Object anObject)
   {
      string aString = (String) anObject;
      double length = aString.length();
      return length
   }
}
```

Now you need to construct an object of the StringMeasurer class and pass it to the average method.

```
Measurer stringMeas = new StringMeasurer();
String[] strings = { … };

double averageLength = Data.average(strings, stringMeas);
```

R10.11

You will get an error. When the callback method is used, a String object will be passed to a method that id designed for calculating the area of a Rectangle object. String objects do not have getWidth() and getHeight() methods.

R10.12

The `f` method can access the variables `b`, `t`, and `x`.

R10.13

The compiler gives an error saying that the local variable is accessed from within inner class and needs to be declared `final`:

```
local variable a is accessed from within inner class; needs to be declared final
```

If we change the variable so that it is declared as `final`, then the inner class can access it, provided it does not try to assign a new value.

R10.14

We would need to put each class (`InvestmentViewer1` and `AddInterestListener`) in its own file. With this change, the class `AddInterestListener` is no longer able to access the `account` variable. Thus, we need to add a `"private BankAccount account"` variable to the class `AddInterestListener`, and have it receive the bank account in the constructor.

R10.15

An event is an external activity to which a program may want to react. For example, "user clicked on button X" is an event.

An event source is the user-interface component that generates a particular event. Event sources report on events. When an event occurs, the event source notifies all event listeners.

An event listener belongs to a class that is provided by the application programmer. Its methods describe the actions to be taken when an event occurs.

R10.16

With a console application, the programmer is in control and it can ask the user for input, in the order that is most convenient for the program.

With a graphical user interface application, the programmer lays out the various elements for user input. Then the user is in control. The user can click and type into the components in any order, and the programmer must be able to cope with that variety.

R10.17

An `ActionEvent` is generated when the user-interface library has detected a particular user intent, such as clicking on a button or hitting `ENTER` in a text field.

A `MouseEvent` is generated whenever the user moves or clicks the mouse.

R10.18

An `ActionListener` is only interested in one notification: when the action happens.

A `MouseListener` has multiple methods because there are several potentially interesting mouse events, such as pressing, releasing, or clicking the mouse button.

R10.19

Yes–a class can be the event source for multiple event types. For example, a `JButton` is the event source for both mouse events and action events.

You can listen to the mouse events of a button, simply by installing a mouse listener:

```
button.addMouseListener(mouseListener);
```

You might want to do that for example to make the button glow in a different color whenever the mouse hovers over it.

R10.20

Every event carries the *source* object. You can retrieve it with the `getSource` method.

A mouse event has the following additional information:

- the x- and y- component of the mouse position
- the click count

An action event has the following additional information:

- the command string associated with the action
- the modifier keys held down during the action event

Hint: This information can be found in the API documentation.

R10.21

Event listeners often need to access instance variables from another class. An inner class can access the instance variables of the outer class object that created it. Thus, it is convenient to use inner classes when implementing event listeners.

If Java had no inner classes, we could pass the values of instance variables needed to be accessed to the event listener constructor, so that local references/copies can be stored in the event listener object.

R10.22

The `paintComponent` method is called by the GUI mechanism whenever *it* feels that a component needs to be repainted. That might happen, for example, because the user just closed another window that obscured the component.

A programmer calls the `repaint` method when the *programmer* feels that the component needs to be repainted (for example, if the state of the object has changed). That call is a request to the GUI mechanism to call `paintComponent` at an appropriate moment.

R10.23

A frame is a window to which you can add a component to be displayed on the screen. A frame can be displayed on its own, even if it is empty. However, we cannot simply add multiple components directly to a frame–they would be placed on top of each other. A panel is a container to group multiple user-interface components together. A panel needs to be added to a frame to be displayed.

Chapter -11

Review Excercise:

R11.1

Whenever the user tries to access a file for reading and if the the file doesn't exist then an exception is thrown.And, whenever a user tries to access a file, which doesn't exist, to write on it, then a new file is created with length 0.

R11.2.

An exception is thrown whenever a user tries to write on a file which is write protected since the file cannot be modified.

R11.3

Backslash has a special meaning in programs hence it cannot be used directly in a string. To solve this problem, double backslash is used. it will indicate that the next character is a backslash.

R11.

4File file=new File("c:temp\\output.dat");
FileInputStream acces=new FileInputStream(file);
About two line can be used in your code.
This is basic syntax to open a new connection to file.

So, file name should be c:temp\\output.dat

double backslash is required here.
Because, double backslash makes the next character (means second backslash) read as a string by java.

R11.6

Try-catch block is used to handle the exception. **In a try block, we write the code which may throw an exception and in catch block we write code to handle that exception**.

Throw keyword is used to explicitly throw an exception. Generally, throw keyword is used to throw user defined exceptions

R11.7

1.) Checked Exceptions:

Checked exceptions are those which are checked at the comiple time by the compiler. There are some of the exceptions which are very likely to occur at some condition so complier it self forces to handle those exceptions and report at the compile time. These exceptions can be handled by try catch block or by throws keyword.
For Example: While reading a file it is very likely that file is not present or unable to open the file can happen. So for this situation compiler forces you to handle these exceptions.

```
public void read(){
FileReader file = new FileReader("C:\\test\\a.txt");
BufferedReader fileInput = new BufferedReader(file);
}
```

Here compiler will report to for FileNotFoundException()

Unchecked Exceptions:

The exceptions which are checked at the run time not at the compile time. Unchecked Exception doesn't likely to occur in the program regularly. Unchecked excpetions are the result of bad programming or system failure. For example NullPointerException , ArithmeticException
for ex.

```
public static void main(String args[]){
    System.out.println(10/0);   // Causes Exceptions
}
```

throws keyword is required only for Checked exceptions and usage of throws keyword is meaningless in case of unchecked Exceptions. Only if checked exceptions have been using throws keyword then the caller function can catch that exception or throw that exception.

R11.8

yes there is no need of declaring that your method might throw an IndexOutOfBound exception beacuse it is a run time exception   as a good programmer you can avoid it . but coming to the checked exceptions we cannot ,because they rellated to environmental problems

R11.9

Whenever throw statement is executed first it **find the type of exception** throw then execute the **first line in the first catch block** that processes that type of exception
.

R11.10

If an exception does not have a matching catch clause, **the current method terminates and throws the exception to the next higher level**. If there is no matching catch clause at any higher level, then the program terminates with an error. Your program can do anything with the exception object that it chooses to.

R11.11

It can **handle the error by printing out a specific message and ask the user to input the correct data**.

R11.2

1It is not necessary that type of the exception object always same as the type declared in the catch block. In the catch block the exception which we are using is always has to be same or super class of the exception thrown.
if the exception thrown is FileNotFoundException() then in catch block we can use either FileNotFoundException() or Exception() which is super class of FileNotFoundException() but we can not use any subclass of ?FileNotFoundException().

R11.15

Exceptions:
These are the conditions which disturbs the normal execution of programs and these are handled by using exception handling mechanism in java using try/catch methods.
Checked exceptions:
These are the exceptions which are compile time errors so we can handle these by using try/catch methods.
Unchecked exceptions:

These are the exceptions which can't be handled by try/catch because these occur by bad programming and these occur during runtime.
The following exceptions can be thrown by next() or nextInt() methods of Scanner object..

Exceptions thrown by **next()** method are:

1. NoSuchElementException - if no more tokens are available for scanning and it is a unchecked exception.
2. IllegalStateException - if this scanner is closed unexpectedly and it is a unchecked exception
Exceptions thrown by **nextInt()** method:
1. InputMismatchException - if the next token does not match the Integer regular expression, or is out of range and it is a unchecked exception.
2. NoSuchElementException - if input is more than expected and it is a unchecked exception.
3. IllegalStateException - if this scanner is closed unexpectedly and it is a unchecked exception.
4. NumberFormatException - if the input is not an integer and it is also a unchecked exception.

R11.18

In that case, if suppose the statement

PrintWriter out = new PrintWriter(filename);

throws an IOException, then since this statement is not put into a try block hence this will cause the program to crash. So to avoid this malfunctioning we need to rearrange the statements like below.

```
try
{
PrintWriter out = new PrintWriter(filename);
 // Write output. out.close(); }

catch (IOException exception)
{                    }
```

In the above code, the close is automatically called.

Chapter 21

R21.1 What is the difference between an input stream and a reader?
Streams handle binary data, accessing sequences of bytes. Readers and writers handle data in text form, accessing sequences of characters.

• R21.1

Write a few lines of text to a new PrintWriter("output1.txt", "UTF-8") and the same text to a new PrintWriter("output2.txt", "UTF-16"). How do the output files differ?
The exercise should say PrintWriter, not FileWriter. Most text editors will automatically convert the text using the proper encoding. In Microsoft Word, the software detects the encoding and asks the user to confirm it before opening the file. If a different encoding is specified, the values in the file are interpreted as different characters.

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;

public class EncodingTester
{
   public static void main(String[] args)
      throws FileNotFoundException, UnsupportedEncodingException
   {
      PrintWriter out = new PrintWriter("output1.txt", "UTF-8");
      out.println("If you write to a file reader, there will be a "
      + "compile-time error because input files don't support output "
      + "operations such as print.\n If you write to a random access "
      + "file that was opened for reading only, there will be an "
      + "exception.");

      PrintWriter out2 = new PrintWriter("output2.txt", "UTF-16");
      out2.println("If you write to a file reader, there will be a "
      + "compile-time error because input files don't support output "
      + "operations such as print.\n If you write to a random access "
      + "file that was opened for reading only, there will be an "
      + "exception.");

      out.close();
      out2.close();
   }
}
```

• R21.3

How can you open a file for both reading and writing in Java?
You need to open a RandomAccessFile to open a file for both reading and writing. For example,
RandomAccessFile f = new RandomAccessFile("bank.dat", "rw");

•• • R21.4

What happens if you try to write to a file reader?
If you write to a file reader, there will be a compile-time error because input files don't support output operations such as print.

•• R21.5

5 What happens if you try to write to a random access file that you opened only for reading? Try it out if you don't know.
If you write to a random access file that was opened for reading only, there will be an exception.

• R21.6

How can you break the Caesar cipher? That is, how can you read a document that was encrypted with the Caesar cipher, even though you don't know the key?
To break the Caesar cipher, you can try all 25 keys (b...z) to decrypt the message. Look at the decryptions and see which one of them corresponds to English text.

• R21.7

What happens if you try to save an object that is not serializable in an object output stream? Try it out and report your results.
If you try to save an object to a object stream and the object is not serializable, an exception is thrown.

• R21.8

Of the classes in the java.lang and java.io packages that you have encountered in this book, which implement the Serializable interface?
java.lang.Boolean
java.lang.Character
java.lang.Double
java.lang.Integer
java.lang.String
java.lang.Throwable and its subclasses, i.e., all exceptions
java.io.File

• R21.9

Why is it better to save an entire ArrayList to an object output stream instead of programming a loop that writes each element?
If you simply save the entire ArrayList, you do not have to save the number of elements. If you saved a collection manually, you'd first have to write out the length, then all entries. When reading the collection back in, you'd have to read in the length, then allocate sufficient storage, and then read and store all entries. It is much simpler to just write and read the ArrayList and let the serialization mechanism worry about saving and restoring the length and the entries.

• R21.10

What is the difference between sequential access and random access?
Sequential access forces you to read all bytes in a file in order, starting from the first byte and progressing through the last byte. Once a byte has been read, it cannot be read again, except by closing and reopening the file. With random access, you can repeatedly select any position in the file for reading or writing.

• R21.11

What is the file pointer in a file? How do you move it? How do you tell the current position? Why is it a long integer?
The file pointer denotes the current position in a random access file for reading and writing. You move it with the seek method of the RandomAccessFile class. You get the current position with the getFilePointer method. The position is the number of bytes from the beginning of the file, as a long integer, because files can be longer than 2GB (the longest length representable with an int).

• R21.12

How do you move the file pointer to the first byte of a file? To the last byte? To the exact middle of the file?
```
f.seek(0);
f.seek(f.length() - 1);
f.seek(f.length() / 2);
```

• R21.13

What happens if you try to move the file pointer past the end of a file? Try it out and report your result
It is legal to move the file pointer past the end of the file. If you write to that location, the file will be enlarged. If you read from that location, an exception is thrown.

• R21.14

Can you move the file pointer of System.in?
You can't move the file pointer on System.in. It is not a RandomAccessFile, therefore a call to the seek method is a compile-time error.

• R21.15

Paths can be absolute or relative. An absolute path name begins with a root element (/ on Unix-like systems, a drive letter on Windows). Look at the Path API to see how one can create and recognize absolute and relative paths. What does the resolve method do when its argument is an absolute path?
Passing an absolute path to the resolve method returns the passed-in path.

• R21.16

Look up the relativize method in the Path API and explain in which sense it is the opposite of resolve. Give two examples when it is useful; one for files and one for directories
.The following example shows usage of relativize with both files and directories:

```
Path path01 = Paths.get("Foo.txt");
Path path02 = Paths.get("Bar.txt");
Path path03 = Paths.get("/Java/JavaTemp/Foo.txt");
Path path04 = Paths.get("/Java/2015");

Path path01_to_path02 = path01.relativize(path02);
System.out.println(path01_to_path02);

Path path02_to_path01 = path02.relativize(path01);
System.out.println(path02_to_path01);

Path path03_to_path04 = path03.relativize(path04);
System.out.println(path03_to_path04);

Path path04_to_path03 = path04.relativize(path03);
System.out.println(path04_to_path03);
```

Would result in the following:

```
..\Bar.txt
..\Foo.txt
..\..\2015
..\JavaTemp\Foo.txt
```

• R21.17

What exactly does it mean that Files.walk yields the results in depth-first order? Are directory children listed before or after parents? Are files listed before directories? Are either listed alphabetically? Run some experiments to find out
Depth first order means that it will traverse downward through the directories until it reaches the bottom of the tree. From there it will handle all siblings at each level as it moves through the tree in a recursive nature. Directory children are listed after their parents. Files will be listed before directories. The listing of each directory will be displayed alphabetically.

Chapter 22 Review Exercise Solutions

R22.1

Run a program with the following instructions:
GreetingRunnable r1 = new GreetingRunnable("Hello");
GreetingRunnable r2 = new GreetingRunnable("Goodbye");
r1.run();
r2.run(); Note that the threads don't run in parallel. Explain.
The reason is that the code calls run on the GreetingRunnable objects so they run in sequence in the same thread. It should create a thread for each and call start on the threads.

R22.2

In the program of Section 22.1, is it possible that both threads are sleeping at the same time? Is it possible that neither of the two threads is sleeping at a particular time? Explain.
Yes, the thread scheduler may just switch to thread B while thread A is sleeping. Then thread B may also go to sleep, and the scheduler may find both thread A and B sleeping at the same time.
Yes, for example, thread A can be printing to the console while the thread scheduler decides to give the time slice to thread B. Thread B gets its turn and resumes working on whatever it was doing.

R22.3

In Java, a program with a graphical user interface has more than one thread. Explain how you can prove that

There is one MAIN thread in the program plus other threads for the GUI. For example, when one does not implement the EXIT_ON_CLOSE feature, the program will not end after a user closes the GUI window.

••• R22.4

Why is the stop method for stopping a thread deprecated? How do you terminate a thread?
The stop method for stopping a thread is deprecated because a thread may have occupied some resources, such as database connections. If we just stop the thread without releasing the resources, the database will eventually run out of connections for others that have work to do.
A thread can be terminated by calling threadName.interrupt().

R22.5

Give an example of why you would want to terminate a thread
Suppose a long computation is being carried out in a thread, the computation is not yet completed, and the user is no longer interested in the result and wishes to terminate the computation. Then it is time to terminate the thread.

R22.6

Suppose you surround each call to the sleep method with a try/catch block to catch an InterruptedException and ignore it. What problem do you create?
The problem is that the threads cannot be terminated by interrupting them.

R22.7

What is a race condition? How can you avoid it?
A race condition occurs if the effect of multiple threads on shared data depends on the order in which the threads are scheduled. Race conditions can be avoided using lock objects to control the threads that attempt to manipulate a shared resource.

R22.8

Consider the ArrayList implementation from Section 16.2. Describe two different scenarios in which race conditions can corrupt the data structure.
If the remove method goes to sleep after it removes an element but before it updates currentSize, the add method will use the wrong value when it increments currentSize.
Similarly, if the remove method goes to sleep after it removes an element but before it updates currentSize, the growBufferIfNecessary method may increase the buffer size and copy the removed element to the new buffer.

R22.9

Consider a stack that is implemented as a linked list, as in Section 16.3.1. Describe two different scenarios in which race conditions can corrupt the data structure.
If push goes to sleep before setting first to the new node, an attempt to pop the stack may return an error (if first is null) or the wrong value.
If pop goes to sleep before setting first to first.next, push could set the new node to link to the element that was to be removed by pop. When pop wakes, it will not remove the desired element, but rather set it to first, thereby eliminating the new first created by push.

R22.10

Consider a queue that is implemented as a circular array, as in Section 16.3.4. Describe two different scenarios in which race conditions can corrupt the data structure.
If the remove method goes to sleep after it removes an element but before it updates head, the growBufferIfNecessary method will set head to the wrong value.
Similarly, if the remove method goes to sleep after it removes an element but before it updates currentSize, the growBufferIfNecessary method may increase the buffer size, copy the removed element to the new buffer, and set tail to the wrong value.
21.11
A deadlock occurs if no thread can proceed because each thread is waiting for another thread to do some work first.
One important way to avoid deadlocks is to call signalAll/notifyAll after a state change that may allow waiting threads to proceed. But there are other potental causes for deadlocks that are not so easily remedied.

•• R22.11

What is a deadlock? How can you avoid it?
The sleep method puts the current thread to sleep for a given number of milliseconds, after which it wakes up. The await method blocks the thread and allows another thread to acquire the object lock. In other words, sleep will sleep for a specified period of time; a waiting thread is blocked until another thread calls signalAll or signal on the condition object for which the thread is waiting.

R22.12

What is the difference between a thread that sleeps by calling sleep and a thread that waits by calling await?
It will wait forever.

R22.13

What happens when a thread calls await and no other thread calls signalAll or signal?
There are two threads: the animation thread and the GUI thread. They share a resource, namely the array of values. But the animation thread is almost 100% certain to be sleeping when the GUI thread paints the graph. If for some reason, the display was repainted while the animation thread was modifying it, then it would display the wrong image. Thus, no synchronization is needed.

R22.14

In the algorithm animation program of Section 22.6, we do not use any conditions. Why not?
The array to be sorted is filled automatically with random numbers. There is no need for user input in the animation.

Chapter -18

R18.1

What is a type parameter?

Type parameters are variables that can be instantiated with class or interface types.

R18.2

What is the difference between a generic class and an ordinary class?

A generic class has one or more type parameters. In order to use a generic class, you need to instantiate the type parameters, that is, supply actual types. Ordinary classes do not use type parameters.

R18.3

What is the difference between a generic class and a generic method?

A generic class is a class with one or more type parameters. A generic method is a method with one or more type parameters.

R18.4

Why is it necessary to provide type arguments when instantiating a generic class, but not when invoking an instantiation of a generic method?

You must provide a type argument for instantiating a generic class so that the compiler knows the memory size for each each element within the gereic class. There is no need to supply type arguments when invoking an instantiation of a generic class since the type class is already known.

R18.5

Find an example of a non-static generic method in the standard Java library.

The toArray method in the java.util.LinkedList class:

```
public <T> T[] toArray(T[] a)
```

R18.6

Find four examples of a generic class with two type parameters in the standard Java library.

```
AbstractMap<K,V>
Dictionary<K,V>
EnumMap<K extends Enum<E>,V>
HashMap<K,V>
Hashtable<K,V>
IdentityHashMap<K,V>
LinkedHashMap<K,V>
TreeMap<K,V>
WeakHashMap<K,V>
```

R18.7

Find an example of a generic class in the standard library that is not a collection class.

```
java.lang.Class<T>
java.lang.Enum<T>
java.util.concurrent.Exchanger<V>
java.util.concurrent.ExecutorCompletionService<V>
java.util.concurrent.FutureTask<T>
```

R18.8

Why is a bound required for the type parameter T in the following method? int binarySearch(T[] a, T key)

A binary search locates a value in a sorted array by determining whether the value occurs in the first or second half, then repeating the search in one of the halves. To know if the key occurs in the first half or second half, the binary search algorithm needs to be able to determine whether the key is smaller or greater than the middle element. The type bound allows it to use the `compareTo` method of the `Comparable` interface to compare values because any value in the array must implement `Comparable`.

R18.9

Why is a bound not required for the type parameter E in the HashSet class?

All classes are descendants of the class `Object`. The class `Object` has a method to calculate the hash code of an object. Thus, objects of any class can be stored in a hash set.

R18.10

What is an ArrayList<Pair>?

It is an array list of `Pair`s of objects of the same type (e.g. an array list of `String` pairs).

R18.11

Explain the type bounds of the following method of the Collections class. public static > void sort(List a) Why doesn't T extends Comparable or T extends Comparable suffice?

`public static <T extends Comparable<? super T>> void sort(List<T> a)` means that the type variable `T` must extend `Comparable<U>` for some type `U` where `U` is any supertype of `T` (or possibly `T` itself).

The bound `<T extends Comparable>` would not have been sufficient. Suppose we want to sort an `ArrayList<BankAccount>`, where the class `BankAccount` implements `Comparable<BankAccount>`. Then we would get a warning. A `Comparable` object is willing to compare itself against any object, but a `BankAccount` is only willing to compare against another `BankAccount`.

`public static <T extends Comparable<T>> void sort(List<T> a)` does not suffice either because it is still too restrictive. Suppose the `BankAccount` class implements `Comparable<BankAccount>`. Then the subclass `SavingsAccount` also implements `Comparable<BankAccount>` and not `Comparable<SavingsAccount>`.

R18.12

What happens when you pass an ArrayList to a method with an ArrayList parameter variable? Try it out and explain.

We will call the method with the raw `ArrayList` parameter the "legacy method". You can pass a parameterized type, such as `ArrayList<String>` or `ArrayList<BankAccount>` to the legacy method. This is not completely safe - after all, the legacy method might insert an object of the wrong type. Nevertheless, your program will compile without a warning.

For example, compiling and running the following program does not produce a warning or error:

```
import java.util.ArrayList;

public class Test
{
   public static void print(ArrayList list)
   {
      for (int i = 0; i < list.size(); i++)
      {
         System.out.println(list.get(i));
      }
   }

   public static void main(String args[])
   {
      ArrayList<String> strList = new ArrayList<String>();
      strList.add("A");
      strList.add("B");
      strList.add("C");
      print(strList);
   }
}
```

R18.13

What happens when you pass an ArrayList to a method with an ArrayList parameter variable, and the method stores an object of type BankAccount into the array list? Try it out and explain.

We will call the method with the raw `ArrayList` parameter the "legacy method".

The code calling the legacy method will compile without a warning. A warning is issued for the legacy method. When the wrong element is inserted, no error happens at that time. When the element with the wrong type is later retrieved, then a `ClassCastException` occurs.

Consider this example:

```
import java.util.ArrayList;

public class Test
{
   public static void main(String args[])
   {
      ArrayList<String> strList = new ArrayList<String>();
      strList.add("A");
      strList.add("B");
      strList.add("C");
      doSomethingBad(strList);
      String first = strList.get(0);
   }

   public static void doSomethingBad(ArrayList list)
   {
      list.add(0, new BankAccount(10000));
   }
}
```

Compiling with the `-Xlint:unchecked` option yields this warning:

```
javac -Xlint:unchecked Test.java
Test.java:17: warning: [unchecked] unchecked call to add(int,E)
as a
member of the raw type java.util.ArrayList
      list.add(0, new BankAccount(10000));
              ^
```

We get the following run-time error:

```
java Test
Exception in thread "main" java.lang.ClassCastException:
BankAccount
cannot be cast to java.lang.String
 at Test.main(Test.java:12)
```

R18.14

What is the result of the following test? ArrayList accounts = new ArrayList<>(); if (accounts instanceof ArrayList) . . . Try it out and explain

Compiling

```
ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
if (accounts instanceof ArrayList<String>)
{
   System.out.println("true");
}
else
{
   System.out.println("false");
}
```

produces the following error:

```
Test.java [15:1] illegal generic type for instanceof
      if (accounts instanceof ArrayList<String>)
System.out.println("true");
                                       ^
```

The virtual machine that executes Java programs does not work with generic classes or methods. Instead, it uses raw types, in which the type parameters are replaced with ordinary Java types. Each type parameter is replaced with its bound, or with `Object` if it is not bounded. The compiler erases the type parameters when it compiles generic classes and methods.

At run time, Java uses raw types (e.g. `Object` or `Comparable`). Since "`accounts instanceof ArrayList<String>`" cannot be translated to a run-time raw type, its use is not allowed.

R18.15

The ArrayList class in the standard Java library must manage an array of objects of type E, yet it is not legal to construct a generic array of type E[] in Java. Locate the implementation of the ArrayList class in the library source code that is a part of the JDK. Explain how this problem is overcome.

You should look for the source code in a directory similar to

```
C:\Program Files\Java\jdk1.6.0_03\src.zip
```

and inside the zipped file, under `java\util` you should find `ArrayList.java`. Copy this file and look through it. You see immediately that the class contains an array of `Object` elements.

```
private transient Object[] elementData;
```

There is a constructor:

```
public ArrayList(Collection<? extends E> c)
```

The argument is a `Collection` that inherits from a class of type parameter `E`. The `Collection` elements are placed into the `Object` array by using the following:

```
elementData = c.toArray();
```

It is expected that the `toArray` method will return an `Object` array. If the conversion doesn't happen correctly, it compensates by coercing it with a copy and specifying the original class.

```
if (elementData.getClass() != Object[].class)
{
   elementData = Arrays.copyOf(elementData, size,
Object[].class);
}
```

Chapter-19

R19.01

Provide expressions that compute the following information about a Stream. a. How many elements start with the letter a? b. How many elements of length greater than ten start with the letter a? c. Are there at least 100 elements that start with the letter a? (Don't count them all if there are more.)

1. words.filter(w -> w.startsWith("a")).count()
2. words.filter(w -> w.startsWith("a")).filter(w -> w.length() > 10).count()
3. words.filter(w -> w.startsWith("a")).limit(100).count()

R19.02

How can you collect five long words (that is, with more than ten letters) from an ArrayList without using streams? Compare your solution with the code frag_ment in Section 19.1. Which is easier to understand?

Traditional Code to collect 5 large words from an ArrayList<String>

```
int count = 0;
ArrayList<String> resultList = new ArrayList<>();
Iterator<String> oldList = words.iterator();
while (oldList.hasNext() && count < 5)
{
   String str = oldList.next();
   if (str.length() > 10)
   {
      resultList.add(str);
      count++;
```

```
    }
}
```

New code with Streams

```
resultStream = wordList.stream();
   .filter(w -> w.length() > 10)
   .limit(5);
```

The new stream syntax is easier to understand.

R19.03

What is the difference between these two expressions?

words.filter(w -> w.length() > 10).limit(100).count()

words.limit(100).filter(w -> w.length() > 10).count()

a) This statement finds the first 100 elements that have a length greater than 10.

b) This statement finds the elements with length greater than 10 within the first 100 elements.

R19.04

Give three ways of making a Stream (or five, including the ones described in Special Topic 19.1

1) Using the `Stream.of` method by listing the elements:

```
  Stream<String> wordStream = Stream.of("Mary", "had", "a",
"little", "lamb");
```

2) Using the `Steam.of` method and an Array of objects:

```
  String[] words = {"Mary", "had", "a", "little", "lamb"};
  Stream<String> wordStream = Stream.of(words);
```

3) Using the `stream` method of a collection:

```
  List<String> wordList = new ArrayList<>();
  Stream<String> wordStream = wordList.stream();
```

R19.05

How can you place all elements from a Stream into

a . a List<Integer>?

b. an Integer[] array?

c. an int[] array

a) `List<Integer> result = stream.collect(Collectors.toList());`

b) `Integer[] result = stream.toArray(Integer[]::new);`

c) Use b) above and then process the wrapper array converting it to an array of primitives.

R19.06

How do you turn a Stream into a Stream, with each number turned into the equivalent string? How do you turn it back into a Stream?

a) Stream<String> to Stream<Double>

```
Stream<Double> dblValues = strValues.stream()
  .mapToDouble(s -> Double.parseToDouble(s));
```

a) Stream<Double> to Stream<String>

```
Stream<String> strValues = dblValues.stream()
  .map(s -> Double.toString(s));
```

R19.07

Give three ways of making a lambda expression that applies the length method to a String (or four if you read Special Topic 19.2).

1. lambda body is a single expression

```
// returns names larger than 10 characters
name -> name.length() < 10
```

2. lambda body uses a defined method call

```
// returns names larger than 10 characters
name -> bigNames(name)
```

3. lambda body uses a functional block

```
// returns names larger than 10 characters
name ->
{
   int len = name.length();
   if (len > 10)  return name;
}
```

R19.08

Given an Optional, what are three different ways of printing it when it is pres_ent and not printing anything when it isn't? Which of these can be adapted to print the string "None" if no string is present?

Three different ways of printing when the optional value is present and not printing when it is not are as follows.

```
1. System.out.print(optResults.orElse(""));
2. optResults.ifPresent(v -> System.out.print(v));
3. if (optResults.isPresent())
   {
      System.out.print(optResults.get()):
   }
```

Options 1 and 3 above can be configured to print "None" if the optional value is not present.

R19.09

Describe five different ways of producing an IntStream. Which of them can be adapted to producing a DoubleStream?

1. Create an IntStream from individual integers or from an array.

```
IntStream stream = IntStream.of(3, 1, 4, 1, 5, 9);
int[] values = . . .;
stream = IntStream.of(values);
```

2. Create an IntStream from a range of consecutive integers.

```
IntStream stream = IntStream.range(a, b);
```

3. Create an IntStream from a random range of integers: inclusive of loBound and exclusive of hiBound.

```
  Random generator = new Random();
  IntStream dieTosses = generator.ints(loBound, hiBound);
```

4. Create an IntStream from a String objects Unicode points.

```
  IntStream codePoints = str.codePoints();
```

5. Create an IntStream from another stream by applying some function.

```
  IntStream lengths = words.mapToInt(w -> w.length());
```

Options 1, 3, and 5 can be used to create a DoubleStream with the corresponding methods used.

R19.10

suppose you want to find the length of the longest string in a stream. Describe two approaches: using mapToInt followed by the max method of the IntStream class, and calling the max method of Stream. What are the advantages and disadvantages of each approach?

a) Using the mapToInt approach:

```
int maxOfLengths = words
  .mapToInt(w -> w.length())
  .max();
```

b) Using the stream.max approach on a non-primitive stream:

```
final Comparator<String> comp = (s1, s2) -> Integer.compare(
s1.length(), s2.length());
int maxOfLengths = words
  .max(comp);
```

The advantage of the first approach is that it does not need a comparator for primitive type streams.

R19.11

List all terminal operations on object streams and primitive-type streams that have been discussed in this chapter.

```
findFirst
findAny
min
max
count
toArray
collect
forEach
```

```
allMatch
anyMatch
noneMatch
```

R19.12

List all collectors that were introduced in this chapter.

```
averagingDouble
averagingInt
averagingLong
counting
groupingBy
joining
maxBy
minBy
summingDouble
summingInt
summingLong
toList
toSet
```

R19.13

xplain the values used in the orElse clauses in Section 19.10.2.

The intent of the `.orElse` values is to provide a meaningful value when the compution cannot be completed. For example,
the `average().orElse(0)` provides the value 0 when the average cannot be computed beacause there are no values to average and the result would require dividing by zero.
Both `min().orElse(Double.MAX_VALUE)` and `max().orElse(Double.MIN_VALUE)` provide a default value when calculating the minimum and maximum of a set of values.