Java DeiTel exercise solutions

Table of Contents

[H¥pƐrlinkƐd] Clicking on above chapters will redirect to the location!

Beginning Note

All chapters are not available here and may contain tons of mistakes. First thing is that for most cases text generative predictive models such as GPT3.5, bard, claude or PaLM are capable enough to solve those. And codes are explicitly available on GitHub. Check,

1) https://github.com/ssarrayya/java-deitel-exercises

2) https://github.com/pdeitel/JavaHowToProgram11e_EarlyObjects/tree/master

3) https://github.com/guto-alves/java-how-to-program-11e

Especially thanks to **Hrivu Samaddar** for the correction!

Chapter 01

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

1.1 Fill in the blanks in each of the following statements:

a) Computers process data under the control of sets of instructions called **programs**.
b) The key logical units of the computer are the **input unit**, **output unit**, **memory unit**, **central processing unit**, **arithmetic and logic unit** and **secondary storage unit**.
c) The three types of languages discussed in the chapter are **machine languages**, **assembly languages** and **high-level languages.**
d) The programs that translate high-level language programs into machine language are called **compilers**.
e) **Android** is an operating system for mobile devices based on the Linux kernel and Java.
f) **Release candidate** software is generally feature complete, (supposedly) bug free and ready for use by the community.
g) The Wii Remote, as well as many smartphones, use **accelerometer** which allows the device to respond to motion.

1.2 Fill in the blanks in each of the following sentences about the Java environment:

a) The **java** command from the JDK executes a Java application.
b) The **javac** command from the JDK compiles a Java program.
c) A Java source code file must end with the **.java** file extension.
d) When a Java program is compiled, the file produced by the compiler ends with the **.class** file extension.
e) The file produced by the Java compiler contains **bytecodes** that are executed by the Java
Virtual Machine.

1.3 in the blanks in each of the following statements (based on Section 1.5):

a) Objects enable the design practice of **information hiding** although they may know how to communicate with one another across well-defined interfaces, they normally are not allowed
to know how other objects are implemented.
b) Java programmers concentrate on creating **classes**, which contain fields and the set of
methods that manipulate those fields and provide services to clients.

c) The process of analyzing and designing a system from an object-oriented point of view
is called **object oriented analysis and design (OOAD)**.
d) A new class of objects can be created conveniently by **inheritance** the new class (called the subclass) starts with the characteristics of an existing class (called the superclass),
possibly customizing them and adding unique characteristics of its own.
e) **The Unified Modeling language (UML)** is a graphical language that allows people who design software systems to use an industry-standard notation to represent them.
f) The size, shape, color and weight of an object are considered **attributes** of the object's class.

1.4 Fill in the blanks in each of the following statements:

a) The logical unit that receives information from outside the computer for use by the computer is the **input unit**.
b) The process of instructing the computer to solve a problem is called **programming**.
c) **High level language** is a type of computer language that uses English like abbreviations for ma-
chine-language instructions.
d) **Output unit** is a logical unit that sends information which has already been processed by
the computer to various devices so that it may be used outside the computer.
e) **Memory** and **storage** are logical units of the computer that retain information.
f) is a logical unit of the computer that performs calculations.
g) **CPU** is a logical unit of the computer that makes logical decisions.
h) **High level language** languages are most convenient to the programmer for writing programs quickly and easily.
i) The only language a computer can directly understand is that computer's **Machine language**.
j) **CPU** is a logical unit of the computer that coordinates the activities of all the other
logical units.

1.5 Fill in the blanks in each of the following statements:

a) **Java** is a platform independent programming language that was built with the objective of allowing programs to be written once and then run on a large variety of electronic devices without modification.
b) **Standard**, **Enterprise** and **Micro** are the names of the three editions of Java that can be used to build different kinds of applications.
c) **Bandwith** is the information-carrying capacity of communication lines, and has rapidly increased over the years and become more affordable. Its availability is a cornerstone for building applications that are significantly connected.
d) An **assembler** is a translator that can convert early assembly-language programs to machine language with reasonable efficiency.

1.6 Fill in the blanks in each of the following statements:

a) Java programs normally go through five phases **editing**, **compilation**, **loading**, **execution**, and **testing**.
b) A(n) **IDE** provides many tools that support the software development process, such as editors for writing and editing programs, debuggers for locating logic errors in programs, and many other features.
c) The command java invokes the **JVM**, which executes Java programs.
d) A(n) **emulator** is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with it.
e) The **class loader** takes the .class files containing the program's bytecodes and transfers
them to primary memory.
f) The **bytecode verifier** examines bytecodes to ensure that they're valid.

1.7 Explain what a just-in-time (JIT) compiler of Java does.

Ans: A just-in-time (JIT) compiler of Java is a component of the Java virtual machine (JVM) that dynamically compiles bytecode into native machine code at runtime, just before the code is executed. This allows the Java code to run faster because it is translated into a form that can be more directly executed by the CPU.

When Java code is executed, it is initially interpreted by the JVM. This interpretation process can be relatively slow, because the JVM must repeatedly analyze and execute the same code each time the program is run. To improve performance, the JIT compiler optimizes commonly executed code by translating it into machine code and storing the compiled code in memory.

The next time the code is executed, the JVM can directly execute the optimized machine code, which is typically much faster than interpreting the Java bytecode.

1.8 One of the world's most common objects is a wrist watch. Discuss how each of the following terms and concepts applies to the notion of a watch: object, attributes, behaviors, class, inheritance (consider, for example, an alarm clock), modeling, messages, encapsulation, interface and information hiding.

Object: A watch is an object that can be described as a timepiece that is worn on the wrist to keep track of time. It can be physical, digital or a combination of both.
Attributes: A watch has various attributes such as its material, color, shape, size, design, and the type of information it provides, such as time, date, alarms, and other features.

Behaviors: A watch has certain behaviors, such as displaying the time accurately, having the ability to set alarms, tracking time in different time zones, and other features that allow users to interact with it.
Class: A watch belongs to the class of timepieces, which include other objects that are used to measure or display time, such as clocks, timers, sundials and hourglasses.
Inheritance: Inheritance is the concept in object-oriented programming where a class can derive properties and attributes from another class. For example, an alarm clock is a type of watch that can inherit properties from the watch class.
Modeling: Modeling is the process of representing real-world objects or systems using object-oriented programming languages or other modeling languages. A watch can be modeled using these languages to create software simulations that replicate its behaviors and interactions with users.
Messages: In object-oriented programming, objects communicate with each other through messages. For example, if a user wants to set an alarm on a watch, they would send a message to the watch object informing it to set the alarm.
Encapsulation: Encapsulation is a mechanism in object-oriented programming that protects the internal workings of an object from external access. This ensures that the object's state and behavior are kept private and can only be accessed through defined interfaces. A watch can implement encapsulation by hiding its internal workings from users and only exposing certain methods and properties.

Interface: An interface is a set of methods and properties that define how an object can be used by other objects. A watch's interface could include methods for displaying the time, setting alarms and changing time zones, and properties such as the watch's physical size and color.
Information hiding: Information hiding is the practice of limiting access to an object's internal attributes and functionality to ensure that it is used correctly and efficiently. A watch can implement information hiding by only exposing certain attributes and methods to users, and hiding the details of how it functions internally.

Chapter 02

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

2.1 Fill in the blanks in each of the following statements:

a) A(n) { left brace begins the body of every method, and a(n) right brace ends the body of every method.
b) You can use the if statement to make decisions.
c) // begins an end-of-line comment.
d) Space, newlines and tabs are called white space.
e) Keywords are reserved for use by Java.
f) Java applications begin execution at method main .
g) Methods System.out.print, System.out.println and System.out.printf display information in a command window.

2.2 State whether each of the following is true or false. If false, explain why.:

a) Comments cause the computer to print the text after the // on the screen when the program executes.
Ans: False. Comments are avoided by the compiler.
b) All variables must be given a type when they're declared.
Ans: True.
c) Java considers the variables number and NuMbEr to be identical.
Ans: False. Variables are case-sensitive.
d) The remainder operator (%) can be used only with integer operands.
Ans: False. It also works with non-integer.
e) The arithmetic operators *, /, %, + and - all have the same level of precedence.
Ans: False. Some of them has higher priority.

2.3 Write statements to accomplish each of the following tasks:

a) Declare variables c, thisIsAVariable, q76354 and number to be of type int.

```
1  int c, thisIsAVariable, q76354, number;
```

b) Prompt the user to enter an integer.

```
1  System.out.print("Enter an integer: ");
```

c) Input an integer and assign the result to int variable value. Assume Scanner variable input can be used to read a value from the keyboard.

```
1  value = input.nextInt();
```

d) Print "This is a Java program" on one line in the command window. Use method System.out.println.

```
1 System.out.println("This is a Java program");
```

e) Print "This is a Java program" on two lines in the command window. The first line should end with Java. Use method System.out.printf and two %s format specifiers.

```
1 System.out.printf("%s%n%s%n", "This is a Java", "program");
```

f) If the variable number is not equal to 7, display "The variable number is not equal to 7" .

```
1     if (number != 7)
2 System.out.println("The variable number is not equal to 7");
```

2.4 Identify and correct the errors in each of the following statements:

a) if (c < 7);
System.out.println("c is less than 7");
Ans: There will be no semicolon after the condition of if.

b) if (c => 7)
System.out.println("c is equal to or greater than 7");
Ans: The condition has an invalid relational operation.

2.5 Write declarations, statements or comments that accomplish each of the following tasks:

a) State that a program will calculate the product of three integers.

```
1 // A program to calculate the product of three integer.
```

b) Create a Scanner called input that reads values from the standard input.

```
1 Scanner input = new Scanner(System.in);
```

c) Declare the variables x, y , z and result to be of type int.

```
1 int x, y, z, result;
```

d) Prompt the user to enter the first integer.

```
1 System.out.print("Enter first integer: ");
```

e) Read the first integer from the user and store it in the variable x.

```
1 x = input.nextInt();
```

f) Prompt the user to enter the second integer.

```
1  System.out.print("Enter second integer: ");
```

g) Read the second integer from the user and store it in the variable y.

```
1  y = input.nextInt();
```

h) Prompt the user to enter the third integer.

```
1  System.out.print("Enter third integer: ");
```

i) Read the third integer from the user and store it in the variable z.

```
1  z = input.nextInt();
```

j) Compute the product of the three integers contained in variables x, y and z, and assign the result to the variable result.

```
1  result = x * y * z;
```

k) Use System.out.printf to display the message "Product is" followed by the value of the variable result.

```
1  System.out.printf("Product is %d%n", result);
```

2.6 Using the statements you wrote in Exercise 2.5, write a complete program that calculates and prints the product of three integers.

```
1  import java.util.Scanner;
2
3  /**
4   * Main
5   */
6  public class Main {
7     public static void main(String[] args) {
8        int x, y, z, result;
9        Scanner input = new Scanner(System.in);
10        System.out.print("Enter first integer: ");
11        x = input.nextInt();
12        System.out.print("Enter second integer: ");
13        y = input.nextInt();
14        System.out.print("Enter third integer: ");
15        z = input.nextInt();
16        input.close();
```

```
17
18          result = x * y * z;
19          System.out.printf("Product is %d%n", result);
20      }
21  }
```

2.7 Fill in the blanks in each of the following statements:

a) Comments are used to document a program and improve its readability.
b) A decision can be made in a Java program with a(n) if.
c) Calculations are normally performed by mathematical statements.
d) The arithmetic operators with the same precedence as multiplication are division and modulo operator.
e) When parentheses in an arithmetic expression are nested, the innermost set of parentheses is evaluated first.
f) A location in the computer's memory that may contain different values at various times throughout the execution of a program is called a(n).variable.

2.8 Write Java statements that accomplish each of the following tasks:

a) Display the message "Enter an integer: ", leaving the cursor on the same line.

```
1  System.out.print("Enter an integer: ");
```

b) Assign the product of variables b and c to variable a.

```
1  a = b * c;
```

c) Use a comment to state that a program performs a sample payroll calculation.

```
1  // This is a program to perform a sample payroll calculation
```

2.9 State whether each of the following is true or false. If false, explain why.

a) Addition is executed first in the following expression: a * b / (c + d) * 5.
Ans: True.
b) The following are all valid variable names: AccountValue, $value, value_in_$, account_no_1234, US$, her_sales_in_$, his_$checking_account, X!, _$_, a@b, and _name.
Ans: False. Because they can't contain special characters like @ or !.
c) In 2 + 3 + 5 / 4, addition has the highest precedence.
Ans: False. The division has more priority than addition.

d) The following are all invalid variable names: name@email.com, 87 , x%, 99er, and 2_.
Ans: True.

2.10 Assuming that x = 5 and y = 1, what does each of the following statements display?
a) System.out.printf("x = %d%n", x + 5);
Ans: x = 10
b) System.out.printf("Value of %d * %d is %d\n", x, y, (x * y));
Ans: Value of 5 * 1 is 5
c) System.out.printf("x is %d and y is %d", x, y);
Ans: x is 5 and y is 1
d) System.out.printf("%d is not equal to %d\n", (x + y), (x * y));
Ans: 6 is not equal to 5

2.11 Which of the following Java statements contain variables whose values are modified?
a) p = i + j + k + 7;
Ans: Modified.
b) System.out.println("variables whose values are modified");
Ans: Not modified.
c) System.out.println("a = 5");
Ans: Not modified.
d) value = input.nextInt();
Ans: Modified.

2.12 Given that y = ax2 + 5x + 2, which of the following are correct Java statements for this equation?
a) y = a * x * x + 5 * x + 2;
Ans: Correct.
b) y = a * x * x + (5 * x) + 2;
Ans: Correct.
c) y = a * x * x + 5 * (x + 2);
Ans: Not correct.
d) y = a * (x * x) + 5 * x + 2;
Ans: Correct.
e) y = a * x * (x + 5 * x) + 2;
Ans: Not correct.
f) y = a * (x * x + 5 * x + 2);
Ans: Not correct.

2.13 State the order of evaluation of the operators in each of the following Java statements, and show the value of x after each statement is performed:

a) x = 7 + 3 * 6 / 2 – 1;

Ans: First there will be multiplication and then division. Later other arithmetic. So the result will be 15.

b) x = 2 % 2 + 2 * 2 - 2 / 2;

Ans: Here modulo operation, multiplication and division will occur first then addition, so the result will be 3.

c) x = (3 * 9 * (3 + (9 * 3 / (3))));

Ans: First innermost operation 9 * 3 / 3 will be executed and later on addition and multiplication. So the result will be, 324.

2.14 Write an application that displays the numbers 1 to 4 on the same line, with each pair of adjacent numbers separated by one space. Use the following techniques:

a) Use one System.out.println statement.
b) Use four System.out.print statements.
c) Use one System.out.printf statement.

```
1  /**
2   * Main
3   */
4  public class Main {
5     public static void main(String[] args) {
6        System.out.println(1 + " " + 2 + " " + 3 + " " + 4);
7
8        System.out.print(1 + " ");
9        System.out.print(2 + " ");
10        System.out.print(3 + " ");
11        System.out.print(4 + " ");
12
13        System.out.printf("%n%d %d %d %d%n", 1, 2, 3, 4);
14     }
15 }
```

2.15 (Arithmetic) Write an application that asks the user to enter two integers, obtains them from the user and prints the square of each, the sum of their squares, and the difference of the squares (first number squared minus the second number squared). Use the techniques shown in Fig. 2.7.

```
1  import java.util.Scanner;
2  public class Main {
```

```
3     public static void main(String[] args) {
4        Scanner input = new Scanner(System.in);
5        System.out.print("Enter first integer: ");
6        int first = input.nextInt();
7        System.out.print("Enter second integer: ");
8        int second = input.nextInt();
9        input.close();
10
11        System.out.printf("First: %d%nSecond: %d%n", first, second);
12        System.out.printf("First squared: %d%nSecond squared: %d%n", first *
first, second * second);
13        System.out.printf("Sum of squares: %d%nDifference of squares: %d
%n", first * first + second * second, first * first - second * second);
14     }
15  }
```

2.16 (Comparing Integers) Write an application that asks the user to enter one integer, obtains it from the user and displays whether the number and its square are greater than, equal to, not equal to, or less than the number 100. Use the techniques shown in Fig. 2.15.

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        int number = input.nextInt();
7        input.close();
8
9        int square = number * number;
10       if (number > 100) {
11          System.out.printf("%d's square is greater than 100%n", number);
12       }
13       else if (number < 100) {
14          System.out.printf("%d's square is less than 100%n", number);
15       }
16       else if (number == 100) {
17          System.out.printf("%d's square is equal to 100%n", number);
18       }
19    }
20 }
```

2.17 (Arithmetic, Smallest and Largest) Write an application that inputs three integers from the user and displays the sum, average, product, smallest and largest of the numbers. Use the techniques shown in Fig. 2.15. [Note: The calculation of the average in this exercise should result in an integer representation of the average. So, if the sum of the values is 7, the average should be 2, not 2.3333....]

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        int num1, num2, num3, sum, product, smallest, largest;
7        double average;
8
9        System.out.print("Enter first integer: ");
10       num1 = input.nextInt();
11
12       System.out.print("Enter second integer: ");
13       num2 = input.nextInt();
14
15       System.out.print("Enter third integer: ");
16       num3 = input.nextInt();
17       input.close();
18
19       sum = num1 + num2 + num3;
20       product = num1 * num2 * num3;
21       average = sum / 3.0;
22
23       smallest = num1;
24       if (num2 < smallest) {
25          smallest = num2;
26       }
27       if (num3 < smallest) {
28          smallest = num3;
29       }
30
31       largest = num1;
32       if (num2 > largest) {
33          largest = num2;
34       }
35       if (num3 > largest) {
36          largest = num3;
```

```
37      }
38
39      System.out.printf("Sum is %d%n", sum);
40      System.out.printf("Average is %.2f%n", average);
41      System.out.printf("Product is %d%n", product);
42      System.out.printf("Smallest is %d%n", smallest);
43      System.out.printf("Largest is %d%n", largest);
44   }
45 }
```

2.18 (Displaying Shapes with Asterisks) Write an application that displays a box, an oval, an arrow and a diamond using asterisks (*):

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        System.out.println("*********   ***    *       *    ");
6        System.out.println("*         * *   *  ***     * *   ");
7        System.out.println("*         * *    * *****   *  *  ");
8        System.out.println("*         * *    *   *     *   * ");
9        System.out.println("*         * *    *   *     *    *");
10       System.out.println("*         * *    *   *     *   * ");
11       System.out.println("*         * *    *   *      * *  ");
12       System.out.println("*         * *   *    *      **   ");
13       System.out.println("*********   ***    *       *    ");
14    }
15 }
```

2.19 What does the following code print?

```
> System.out.printf("*%n**%n***%n****%n*****%n");
```

Ans:

```
*
**
***
****
*****
```

2.20 What does the following code print?

```
1  System.out.println("*");
2  System.out.println("***");
3  System.out.println("*****");
4  System.out.println("****");
```

```
5  System.out.println("**");
```

Ans:

```
*
***
*****
****
**
```

2.21 What does the following code print?

```
1  System.out.print("*");
2  System.out.print("***");
3  System.out.print("*****");
4  System.out.print("****");
5  System.out.println("**");
```

Ans:

```
***************
```

2.22 What does the following code print?

```
1  System.out.print("*");
2  System.out.println("***");
3  System.out.println("*****");
4  System.out.print("****");
5  System.out.println("**");
```

Ans:

```
****
*****
******
```

2.23 What does the following code print?

```
1  System.out.printf("%s%n%s%n%s%n", "*", "***", "*****");
```

Ans:

```
*
***
*****
```

2.24 (Largest and Smallest Integers) Write an application that reads five integers and determines and prints the largest and smallest integers in the group. Use only the programming techniques you learned in this chapter.

```
1  import java.util.Scanner;
2
3  public class Main {
```

```
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         int number1;
7         int number2;
8         int number3;
9         int number4;
10         int number5;
11
12         number1 = input.nextInt();
13         number2 = input.nextInt();
14         number3 = input.nextInt();
15         number4 = input.nextInt();
16         number5 = input.nextInt();
17
18         input.close();
19
20         if (number1 > number2 && number1 > number3 && number1 > number4 && number1 >
number5) {
21             System.out.printf("%d is the largest number%n", number1);
22         } else if (number2 > number1 && number2 > number3 && number2 > number4 && number2
> number5) {
23             System.out.printf("%d is the largest number%n", number2);
24         } else if (number3 > number1 && number3 > number2 && number3 > number4 && number3
> number5) {
25             System.out.printf("%d is the largest number%n", number3);
26         } else if (number4 > number1 && number4 > number2 && number4 > number3 && number4
> number5) {
27             System.out.printf("%d is the largest number%n", number4);
28         } else if (number5 > number1 && number5 > number2 && number5 > number3 && number5
> number4) {
29             System.out.printf("%d is the largest number%n", number5);
30         }
31
32         if (number1 < number2 && number1 < number3 && number1 < number4 && number1 <
number5) {
33             System.out.printf("%d is the smallest number%n", number1);
34         } else if (number2 < number1 && number2 < number3 && number2 < number4 && number2
< number5) {
35             System.out.printf("%d is the smallest number%n", number2);
36         } else if (number3 < number1 && number3 < number2 && number3 < number4 && number3
< number5) {
37             System.out.printf("%d is the smallest number%n", number3);
38         } else if (number4 < number1 && number4 < number2 && number4 < number3 && number4
< number5) {
39             System.out.printf("%d is the smallest number%n", number4);
40         } else if (number5 < number1 && number5 < number2 && number5 < number3 && number5
< number4) {
41             System.out.printf("%d is the smallest number%n", number5);
42         }
```

```
43    }
44 }
```

2.25 (Divisible by 3) Write an application that reads an integer and determines and prints whether it's divisible by 3 or not.

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.print("Enter an integer: ");
7        int number = input.nextInt();
8        input.close();
9
10       if (number % 3 == 0) {
11          System.out.printf("%d is divisible by 3%n", number);
12       } else {
13          System.out.printf("%d is not divisible by 3%n", number);
14       }
15    }
16 }
```

2.26 (Multiples) Write an application that reads two integers, determines whether the first number tripled is a multiple of the second number doubled, and prints the result.

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        int num1, num2;
7        System.out.print("Enter first number: ");
8        num1 = input.nextInt();
9        System.out.print("Enter second number: ");
10       num2 = input.nextInt();
11       input.close();
12
13       if (num1 * 3 % (num2 * 2) == 0) {
14          System.out.printf("%d is a multiple of %d%n", num1 * 3, num2 * 2);
15       } else {
16          System.out.printf("%d is not a multiple of %d%n", num1 * 3, num2 *
2);
```

```
17        }
18     }
19  }
```

2.27 (Checkerboard Pattern of Asterisks) Write an application that displays a checkerboard pattern:

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.println("Enter the size of the checkerboard: ");
7        int size = input.nextInt();
8        input.close();
9
10        for (int i = 0; i < size; i++) {
11           if (i % 2 == 0) {
12              System.out.print(" ");
13           }
14           for (int j = 0; j < size; j++) {
15              System.out.print("* ");
16           }
17           System.out.println();
18        }
19     }
20  }
```

2.28 (Diameter, Circumference and Area of a Circle)

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.print("Enter the radius of a circle: ");
7        int radius = input.nextInt();
8        System.out.printf("Diameter: %d%nCircumference: %f%nArea: %f%n",
2 * radius, 2 * Math.PI * radius,
9              Math.PI * radius * radius);
10        input.close();
11     }
12  }
```

2.29 (Integer Value of a Character)

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.print("Enter a character: ");
7        char character = input.next().charAt(0);
8        input.close();
9
10        System.out.printf("The character %c has the value %d%n", character,
((int) character));
11     }
12  }
```

2.30 (Separating the Digits in an Integer)

```
1  import java.util.Scanner;
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.print("Enter a five-digit integer: ");
7        int number = input.nextInt();
8        input.close();
9
10        int digit1 = number / 10000;
11        int digit2 = (number % 10000) / 1000;
12        int digit3 = (number % 1000) / 100;
13        int digit4 = (number % 100) / 10;
14        int digit5 = number % 10;
15
16        System.out.printf("%d   %d   %d   %d   %d%n", digit1, digit2, digit3,
digit4, digit5);
17     }
18  }
```

2.31 (Table of Squares and Cubes)

```
1  public class Main {
2     public static void main(String[] args) {
3        System.out.println("number    square    cube");
4        for (int i =0; i <= 10; i++)
5        {
```

```
6           System.out.println(i + "        " + i * i + "        " + i * i * i);
7       }
8   }
9 }
```

2.32 (Negative, Positive and Zero Values)

```
 1 import java.util.Scanner;
 2
 3 public class Main {
 4   public static void main(String[] args) {
 5     Scanner input = new Scanner(System.in);
 6     int negativeCount = 0;
 7     int positiveCount = 0;
 8     int zeroCount = 0;
 9
10     for (int i = 0; i < 5; i++) {
11       System.out.print("Enter a number: ");
12       int num = input.nextInt();
13
14       if (num < 0) {
15         negativeCount++;
16       } else if (num > 0) {
17         positiveCount++;
18       } else {
19         zeroCount++;
20       }
21     }
22     input.close();
23
24     System.out.println("Negative numbers: " + negativeCount);
25     System.out.println("Positive numbers: " + positiveCount);
26     System.out.println("Zero numbers: " + zeroCount);
27   }
28 }
```

Chapter 03

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

3.1 Fill in the blanks in each of the following statements:

a) Each class declaration that begins with keyword **public** must be stored in a file that has exactly the same name as the class and ends with the .java filename extension.
b) Keyword **class** in a class declaration is followed immediately by the class's name.
c) Keyword **new** requests memory from the system to store an object, then calls the corresponding class's constructor to initialize the object.
d) Each parameter must specify both a(n) **type** and a(n) **name** .
e) By default, classes that are compiled in the same directory are considered to be in the same package, known as the **default package** .
f) Java provides two primitive types for storing floating-point numbers in memory: **float** and **double** .
g) Variables of type double represent **double precision** floating-point numbers.
h) Scanner method **nextDouble** returns a double value.
i) Keyword public is an access **modifier** .
j) Return type **void** indicates that a method will not return a value.
k) Scanner method **nextLine** reads characters until it encounters a newline character, then returns those characters as a String.
l) Class String is in package **java.lang** .
m) A(n) **import declaration** is not required if you always refer to a class with its fully qualified class name.
n) A(n) **floating point number** is a number with a decimal point, such as 7.33, 0.0975 or 1000.12345.
o) Variables of type float represent **single** _ -precision floating-point numbers.
p) The format specifier **%f** is used to output values of type float or double.
q) Types in Java are divided into two categories—**primitive** types and **reference** types.

3.2 State whether each of the following is true or false. If false, explain why.:

a) By convention, method names begin with an uppercase first letter, and all subsequent words in the name begin with a capital first letter.
Ans: False. For convention names also begin with small letter.
b) An import declaration is not required when one class in a package uses another in the same package.
Ans: True.
c) Empty parentheses following a method name in a method declaration indicate that the method does not require any parameters to perform its task.

Ans: True
d) A primitive-type variable can be used to invoke a method.
Ans: False. Primitive-type variables are not objects.
e) Variables declared in the body of a particular method are known as instance variables and can be used in all methods of the class.
Ans: False. Such is called local variable.
f) Every method's body is delimited by left and right braces ({ and }).
Ans: True.
g) Primitive-type local variables are initialized by default.
Ans: Only instance primitive type variables are initialized by default.
h) Reference-type instance variables are initialized by default to the value null.
Ans: True.
i) Any class that contains public static void main(String[] args) can be used to execute an app.
Ans: True.
j) The number of arguments in the method call must match the number of parameters in the method declaration's parameter list.
Ans: True.
K) Floating-point values that appear in source code are known as floating-point literals and are type float by default.
Ans: False. Such literals are type double by default.

3.3 What is the difference between a local variable and an instance variable?

Local variables are declared and used inside method block. It can't be accessed from other places and it has a little scope. On the other hand, instance variable is declared in the class rather than the method and can be accessed by any method.

3.4 Explain the purpose of a method parameter. What is the difference between a parameter and an argument?

The purpose is to make sure a method can take input. And the difference between a parameter and a method is that method's parameter is the list of extra values that a method will take and on the other hand an argument will pass the value to the method.

3.5 (Keyword new) What's the purpose of keyword new? Explain what happens when you use it.

New keyword is used for creating a new object.

When we use it, the constructor of the class is called to initialize the object.

3.6 (Default Constructors) A class declares a constructor that takes two parameters. How would you create an instance of the class with no parameters?

To create a instance of class without parameters, then we have to use default constructor. When a default constructor is calleld it's instance variables are initialized to their default values.

```
1  class InnerMain {
2
3     int a, b;
4  }
5
6  public class Main {
7     public static void main(String[] args) {
8        InnerMain obj = new InnerMain();
9        System.out.println(obj.a);
10        System.out.println(obj.b);
11     }
12 }
```

3.7 (Instance Variables) Explain the purpose of an instance variable.

Instance variable is used when each of the objects need to contain some information which will be used throughout all methods.

3.8 (Using Classes without Importing Them) Most classes need to be imported before they can be used in an app. Why is every app allowed to use classes System and String without first importing them?

Every app is allowed to use **System** and **String** without first importing them, because they are included in java.lang package, which is imported in almost all programs implicitly.

3.9 (Using a Class without Importing It) Explain how a program could use class Scanner without importing it.

Without explicitly importing class, we can use it if the class is in the same package as we are working. In this way we can also use Scanner without using it.

3.10 (set and get Methods) Explain the disadvantage of creating a class that has no set and get methods for an instance variable.

Without get and set method an instance variable can be used by any object and it's value can be modified which will arise a security risk for an application.

Without get and set method then we have to make our instance variable public, and in this case almost any other objects can access the value as well as edit it. So we'll lose the control over our application and it may also arise bugs and other issues.

```
1  class InnerMain {
2     int a, b;
3  }
4
5  public class Main {
6     public static void main(String[] args) {
7        InnerMain obj = new InnerMain();
8        System.out.println(obj.a);
9        System.out.println(obj.b);
10
11        obj.a = 10;
12        obj.b = 20;
13        System.out.println(obj.a);
14        System.out.println(obj.b);
15     }
16 }
```

So to prevent this and enable more broader control we can use get and set method, where we can set rules on those methods for greater control.

```
class InnerMain {
1     int a, b;
2     int setA(int a) {
3        if (a < 0) {
4           System.out.println("Invalid value");
5           return 1;
6        }
7        this.a = a;
8        return 0;
9     }
10     int setB(int b) {
11        if (b > 0) {
12           System.out.println("Invalid value");
13           return 1;
14        }
```

```
15        this.b = b;
16        return 0;
17    }
18 }
```

Chapter 04

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

4.1 Fill in the blanks in each of the following statements:

a) All programs can be written in terms of three types of control structures: **sequence**, **selection** and **repetition.**
b) The **if ... else** statement is used to execute one action when a condition is true and another when that condition is false.
c) Repeating a set of instructions a specific number of times is called **counter controlled or definite** repetition.
d) When it's not known in advance how many times a set of statements will be repeated, a(n) **sentinel** value can be used to terminate the repetition.
e) The **sequence** structure is built into Java; by default, statements execute in the order they appear.
f) Instance variables of types char, byte, short, int, long, float and double are all given the value **zero** by default.
g) Java is a(n) **strongly typed** language; it requires all variables to have a type.
h) If the increment operator is **prefixed** to a variable, first the variable is incremented by 1, then its new value is used in the expression.

4.2 State whether each of the following is true or false. If false, explain why.:

a) An algorithm is a procedure for solving a problem in terms of the actions to execute and the order in which they execute.

Ans: True

b) A set of statements contained within a pair of parentheses is called a block.

Ans: False. A set of statements withing braces is called a block.

c) A selection statement specifies that an action is to be repeated while some condition remains true.

Ans: False. A repetition statement specify that an action is to be repeated while some condition remain true.

d) A nested control statement appears in the body of another control statement.

Ans: True.

e) Java provides the arithmetic compound assignment operators +=, -=, *=, /= and %= for abbreviating assignment expressions.

Ans: True.

f) The primitive types (boolean, char, byte, short, int, long, float and double) are portable across only Windows platforms.

Ans: False. As they are cross-platform, they can support almost all platforms.

g) Specifying the order in which statements execute in a program is called program control.

Ans: True.

h) The unary cast operator (double) creates a temporary integer copy of its operand.

Ans: False. The unary cast operator (double) creates a temporary floating point copy of its operand.

I) Instance variables of type boolean are given the value true by default.

Ans: False. Instance variables of type boolean are given the value false by default.

j) Pseudo-code helps you think out a program before attempting to write it in a programming language.

Ans: True.

4.3 Write four different Java statements that each add 1 to integer variable x.

```
1  X = X + 1;
2  X++;
3  ++X;
4  X+=1;
```

4.4 Write Java statements to accomplish each of the following tasks:

a) Use one statement to assign the sum of x and y to z, then increment x by 1.

```
1  z = x++ + y;
```

b) Test whether variable count is greater than 10. If it is, print "Count is greater than 10".

```
1 if ( count > 10 )
2       System.out.println(“Count is greater than 10”);
```

c) Use one statement to decrement the variable x by 1, then subtract it from variable total and store the result in variable total.

```
1 total = total - --x;
```

d) Calculate the remainder after q is divided by divisor , and assign the result to q. Write this statement in two different ways.

```
1 Q = Q % divisor;
2 Q %= divisor;
```

4.5 Write Java statements to accomplish each of the following tasks:

a) Declare variables sum of type int and initialize it to 0.

```
1 Int sum = 0;
```

b) Declare variables x of type int and initialize it to 1.

```
1 int x = 1;
```

c) Add variable x to variable sum, and assign the result to variable sum.

```
1 sum = sum + x;
```

d) Print "The sum is: ", followed by the value of variable sum.

```
1 System.out.println(“The sum is: ” + sum);
```

4.6 Combine the statements that you wrote in Exercise 4.5 into a Java application that calculates and prints the sum of the integers from 1 to 10. Use a while statement to loop through the calculation and increment statements. The loop should terminate when the value of x becomes 11.

```
1  public class Main {
2    public static void main(String[] args) {
3      int sum = 0;
4      int x = 1;
5      while ( x <= 10 )
6      {
7        sum += x++;
8      }
9      System.out.println(sum);
10   }
11 }
```

4.7 Determine the value of the variables in the statement product *= x++; after the calculation is performed. Assume that all variables are type int and initially have the value 5.

Ans: The value of x will be 25.

4.8 Identify and correct the errors in each of the following sets of code:

```
1  a) while (c <= 5)
2  {
3  product *= c;
4  ++c;
```

Ans: The closing bracket is missing.

```
1  b) if (gender == 1)
2  System.out.println("Woman");
```

```
3  else;
4  System.out.println("Man");
```
Ans: There's a semicolon after else, so it'll not work as expected.

4.9 What is wrong with the following while statement?
```
1  while (z >= 0)
2        sum += z;
```

Ans: It'll cause an infinity loop. Cause the value of z will remain unchanged.
4.10 Compare and contrast the if single-selection statement and the while repetition statement. How are these two statements similar? How are they different?
Ans: The if statement and while repetition statement both works on a certain condition. If the condition is true then both of these works, or they avoids a lenght of statements.
On the other hand, if selection statement can execute some statements once but the while repetition can execute certain statements until the condition is met.

4.11 Explain what happens when a Java program attempts to divide one integer by another. What happens to the fractional part of the calculation? How can you avoid that outcome?
Ans: When a Java program attempts to divide one integer by another using the division operator ("/"), the result will be an integer division. The fractional part of the calculation is truncated, meaning it is discarded, and we will get the integer quotient as the result. For example, if you divide 7 by 2, the result will be 3, and the fractional part (0.5) is discarded.

To avoid integer division and preserve the fractional part, we can use casting to convert one or both of the integers to floating-point numbers. For example: (double) a / b or a / (double) b.

4.12 Describe the two ways in which control statements can be combined.
Ans: Control statement can be combined in two ways. Those are,
1. Sequential Combination: Control statements are executed in sequence, one after the other. This is the default behavior in Java. For example, statements in a method are executed one after the other, from top to bottom.

2. Nested Combination: In this process one control statement appears inside another. If the outer condition one is true and executed then inner one works.

4.13 What type of repetition would be appropriate for obtaining an input from the user until the user indicates there is no more input to provide? What type would be appropriate for calculating the factorial of 5? Briefly describe how each of these tasks could be performed.

Ans: For such type of repetition, while loop will be much more effective. For instance, the following code will take input from the user and will create a list until the user press 0.

```
1  import java.util.LinkedList;
2  import java.util.Scanner;
3
4  public class Main {
5     public static void main(String[] args) {
6        LinkedList<Integer> list = new LinkedList<Integer>();
7        Scanner sc = new Scanner(System.in);
8        while (true)
9        {
10          int temp = sc.nextInt();
11          if (temp == 0) break;
12          list.add(temp);
13       }
14       sc.close();
15       System.out.println(list.toString());
16    }
17 }
```

For finding the factorial of five, while repetition will be much more effiecint and time saving. The code will be like,

```
1  public class Main {
2     public static void main(String[] args) {
3        int fact = 5, result = 1;
4        while (fact != 1)
5           result *= fact--;
6        System.out.println(result);
7     }
8  }
```

4.14 If integers x and y are set to 7 and 3, what is the value of x after x = y++ and x = ++y?

Ans: 3 and 4

4.15 Identify and correct the errors in each of the following pieces of code. [Note: There may be more than one error in each piece of code.]

```
1  a) if (age >= 65);
```

```
2  System.out.println("Age is greater than or equal to 65");
3  else
4  System.out.println("Age is less than 65)";
```

Ans: At line 1, there's a semicolon after the if condition. And in the fourth line, the semicolon will have to be put after the right parentheses.

b)

```
1  int x == 1, total == 0;
2  while (x <= 10)
3  {
4  total ++x;
5  System.out.println(x);
6  }
```

Ans: At line one, we have to assign a number to x and, so we have to use, x = 1; And on the fourth line, ++ is a unary operator, so we just have to use one operand.

c)

```
1  while (x <= 100)
2  total += x;
3  ++x;
```

Ans: After while statement block statement or braces are missing. In this code only the total += x; will be executed, which will cause an infinity loop. So we have to use braces to define block statement.

d)

```
1  while (y =! 0)
2  {
3  System.out.println (y);
```

Ans: Here the control statement would be y != 0 and there will be a right brace at the very end.

4.16

What does the following program print?

```
 1  // Exercise 4.16: Mystery.java
 2  public class Mystery
 3  {
 4  public static void main(String[] args)
 5  {
 6  int x = -2;
 7  int total = 0;
 8  while (x <= 10)
 9  {
10  int y = x + 2;
11  x++;
```

```
12  total += y;
13  System.out.printf("Y is: %d and total is %d\n", y, total);
14  }// end while
15  }// end main
16  } // end class Mystery
```

Ans: The output will be,

```
Y is: 0 and total is 0
Y is: 1 and total is 1
Y is: 2 and total is 3
Y is: 3 and total is 6
Y is: 4 and total is 10
Y is: 5 and total is 15
Y is: 6 and total is 21
Y is: 7 and total is 28
Y is: 8 and total is 36
Y is: 9 and total is 45
Y is: 10 and total is 55
Y is: 11 and total is 66
Y is: 12 and total is 78
```

Chapter 05

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

5.1 Fill in the blanks in each of the following statements:

a) Typically, **for** statements are used for counter-controlled repetition and **while** statements for sentinel-controlled repetition.
b) The do...while statement tests the loop-continuation condition **after** executing the loop's body; therefore, the body always executes at least once.
c) The **switch** statement selects among multiple actions based on the possible values of an integer variable or expression, or a String.
d) The **continue** statement, when executed in a repetition statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.
e) The **and** operator can be used to ensure that two conditions are both true before choosing a certain path of execution.
f) If the loop-continuation condition in a for header is initially **false**, the program does not execute the for statement's body.
g) Methods that perform common tasks and do not require objects are called **static** methods.

5.2 State whether each of the following is true or false. If false, explain why.:

a) The default case is required in the switch selection statement.

Ans: False. It's optinal.

b) The break statement is required in the last case of a switch selection statement.

Ans: False. Without break the rest of the statements work but it's optional.

c) The expression ((x > y) && (a < b)) is true if either x > y is true or a < b is true.

Ans: False. For conditional and operator both of the operands have to be true.

d) An expression containing the || operator is true if either or both of its operands are true.

Ans: True.

e) The comma (,) formatting flag in a format specifier (e.g., %,20.2f) indicates that a value should be output with a thousands separator.

Ans: True.

f) To test for a range of values in a switch statement, use a hyphen (–) between the start and end values of the range in a case label.

Ans: False. We cannot test a range of values in switch-case statement.

g) Listing cases consecutively with no statements between them enables the cases to perform the same set of statements.

Ans: True.

5.3 Write a Java statement or a set of Java statements to accomplish each of the following tasks:

a) Sum the odd integers between 1 and 99, using a for statement. Assume that the integer variables sum and count have been declared.

```
1  sum = 0;
2  for (count = 1; count <= 99; count += 2)
3      sum += count;
```

b) Calculate the value of 2.5 raised to the power of 3, using the pow method.

```
1  double result = Math.pow(2.5, 3);
```

c) Print the integers from 1 to 20, using a while loop and the counter variable

```
1  i = 1;
2  while (i <= 20)
3  {
4      System.out.print(i);
5      if (i % 5 == 0)
6              System.out.println();
7      else
8              System.out.print('\t');
9      ++i;
```

```
10 }
```

d) Repeat part (c), using a for statement.

```
1  for (i = 1; i <= 20; i++)
2  {
3       System.out.print(i);
4       if (i % 5 == 0)
5            System.out.println();
6       else
7            System.out.print('\t');
8  }
```

5.4 Find the error in each of the following code segments, and explain how to correct it:

```
1  a) i = 1;
2  while (i <= 10);
3  ++i;
4  }
```

Ans: There's a semocolon after while header which will cause an infinity loop and a left brace is missing.

```
1  b) for (k = 0.1; k != 1.0; k += 0.1)
2       System.out.println(k);
```

Ans: Using for with a floating-point may not work, because it's precision is different in different machines.

c)

```
 1  switch (n)
```

```
2 {
3 case 1:
4 System.out.println("The number is 1");
5 case 2:
6 System.out.println("The number is 2");
7 break;
8 default:
9 System.out.println("The number is not 1 or 2");
10 break;
11 }
```

Ans: First case is missing a break block. The code will also work but it's recommended to avoid logical errors.

d) The following code should print the values 1 to 10:

```
1 n = 1;
2 while (n < 10)
3 System.out.println(n++);
```

Ans: The loop will work to print the values between 1 to 9 and will not print 9 becuase of < operator. So we can use while (n < 11) to avoid this error.

5.5 Describe the four basic elements of counter-controlled repetition.

Ans: The four basic elements are, a control variable, the initial value, the increment and the loop-continuation condition.

A Control variable: It is a variable and the loop depends on this variable.

The initial value: It refers to the initial value of the control variable.

The increment: It refers to the change that happens on control variable on each loop.

The loop-continuation condition: It's the condition of the loop.

5.6 Compare and contrast the while and for repetition statements.

The main difference between for repetition and while loop is how the basic four elements of counter-controlled repetition is applied. In while loop, we can only define the loop-continuation condition, where in the for loop we can define control variable, initialize the value, loop-continuation condition as well as the increment.

5.7 If you need to execute the body of a loop at least once, would it be better to use a do...while statement or a while statement?

Ans: For executing the body of the loop at least once, most better option will be do..while statement over while statement. Because in the do..while statement, the inner body will be executed before checking the loop-continuation condition but in the while loop that will occur in the beginning.

5.8 Compare and contrast the break and continue statements.

Ans: The main contrast between these two are that break statement will exit the loop from the occurance and the rest of the lines will be ignored as well as loop-continuation statement. On the other hand, continue will re-run the loop. It'll also avoid the rest of the lines will be ignored as well as loop-continuation statement, but in this time the loop will run again.

5.9 Find and correct the error(s) in each of the following segments of code:

```
1  a) while (i = 1; i <= 10, i+)
2  System.out.println(i);
```

Ans: Here, the increment part has a logical error. It should be i++.

b) The following code should print whether an integer value is negative or zero:

```
1  switch (value)
2  {
3  Case value < 0:
4  System.out.println("Negative");
5  case 0:
6  System.out.println("Zero");
7  }
```

Ans: We can't use control operator in switch-case.

c) The following code should output the odd integers from 19 to 1:

```
1  for (int i = 19; i > 1; i =+ 1)
2  System.out.println(i);
```

Ans: Here the increment part should be i -= 1 and the loop-continuation should be i >= 1;

d) The following code should output the even integers from 1 to 50:

```
1  counter = 0;
2  do
3  {
4  System.out.println(counter + 1);
5  counter += 2;
6  } while (counter <= 51);
```

Ans: Here the first initialize point of counter should be counter = 1. And the condition should be (counter <= 49)

5.10 What does the following program do?

```
1  // Exercise 5.10: Printing.java
2  public class Counting {
3  public static void main( String[] args )
4  {
5  Scanner s = new Scanner(System.in);
6  for (int i = 1; i < 3; i++)
7  {
8  for (int j = 1; j < 5; j++)
9  System.out.print('*');
10  System.out.println("\n######");
11  } // end outer for loop
12  } // end main
13  } // end class Counting
```

Ans: Here the code will print,

```
****
######
****
######
```

5.11 (Extremes) Write an application that finds the minimum and maximum amongst several integers and then computes the sum of the two extremes. The user will be prompted to input how many values the application should ask the user to input.

Ans:

```
1  import java.util.Scanner;
```

```
2
3  public class Main {
4     public static void main(String[] args) {
5        Scanner sc = new Scanner(System.in);
6        System.out.print("How many integers you want to check? ");
7        int n = sc.nextInt();
8
9        int arr[] = new int[n];
10        for (int i=0; i <n; i++)
11           arr[i] = sc.nextInt();
12
13        int max = arr[0], min = arr[0];
14        for (int i=0; i <n; i++)
15        {
16           if (arr[i] > max)
17              max = arr[i];
18           if (arr[i] < min)
19              min = arr[i];
20        }
21        sc.close();
22
23        System.out.println("Max : " + max);
24        System.out.println("Min : " + min);
25        System.out.println("Sum : " + (max + min));
26     }
27  }
```

5.12 (Integers Divisible by 3) Write an application that calculates the sum of those integers between 1 and 30 that are divisible by 3.
Ans:

```
1  public class Main {
2     public static void main(String[] args) {
3        var sum = 0;
4        for (int i = 1; i <= 30; i++)
5           if (i % 3 == 0)
6              sum += i;
7        System.out.println(sum);
8     }
9  }
```

5.13 (The Sum of a Series) Find the summation of the sequence of numbers 1, 2, 3 ... n, where n ranges from 1 to 100. Use type long. Display the results

in a tabular format that shows n and the corresponding sum. If this were a product instead of a sum, what difficulty might you encounter with the variable that accumulates the product?

Ans:

```
1  public class Main {
2    public static void main(String[] args) {
3      var sum = 0;
4      for (int i = 1; i <= 100; i++) {
5        System.out.print(i + " + ");
6        sum += i;
7      }
8      System.out.println(" = " + sum);
9    }
10 }
```

5.14 (Modified Compound-Interest Program)

Ans:

```
1  public class Interest {
2    public static void main(String[] args) {
3      double amount; // amount on deposit at end of each year
4      double principal = 1000.0; // initial amount before interest
5      double rate = 5; // interest rate
6      // display headers
7      System.out.printf("%s%20s%n", "Year", "Amount on deposit");
8
9      for (; rate < 11; rate++) {// calculate amount on deposit for each of ten
years
10       System.out.println("Rate = " + rate);
11       for (int year = 1; year <= 10; ++year) {
12         // calculate new amount for specified year
13         amount = principal * Math.pow(1.0 + rate, year);
14         // display the year and the amount
15         System.out.printf("%4d%,20.2f%n", year, amount);
16       }
17     }
18   }
19 }
```

5.15 (Triangle Printing Program)

Ans:

```
1  public class Main {
2    public static void main(String[] args) {
3      int n = 10;
```

```java

    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= i; j++) {
        System.out.print("*");
      }
      System.out.println();
    }

    System.out.println();

    for (int i = n; i >= 1; i--) {
      for (int j = 1; j <= i; j++) {
        System.out.print("*");
      }
      System.out.println();
    }

    System.out.println();

    for (int i = n; i >= 1; i--) {
      for (int j = n; j >= i; j--) {
        System.out.print(" ");
      }
      for (int j = 1; j <= i; j++) {
        System.out.print("*");
      }
      System.out.println();
    }

    System.out.println();

    for (int i = 1; i <= n; i++) {
      for (int j = n; j >= i; j--) {
        System.out.print(" ");
      }
      for (int j = 1; j <= i; j++) {
        System.out.print("*");
      }
      System.out.println();
    }
  }
}
```

Chapter 06

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

6.1 Fill in the blanks in each of the following statements:

a) A method is invoked with a(n) **method call**.
b) A variable known only within the method in which it's declared is called a(n) **local variable**.
c) The **return** statement in a called method can be used to pass the value of an expression back to the calling method.
d) The keyword **void** indicates that a method does not return a value.
e) Data can be added or removed only from the **top** of a stack.
f) Stacks are known as **last in, first out (LIFO)** data structures; the last item pushed (inserted) onto the stack is the first item popped (removed) from the stack.
g) The three ways to return control from a called method to a caller are **return**, **return expression** and **encountering the last closing right brace**.
h) An object of class **SecureRandom** produces truly random numbers.
i) The method-call stack contains the memory for local variables on each invocation of a method during a program's execution. This data, stored as a portion of the method-call stack, is known as the **stack frame** or **activation record** of the method call.
j) If there are more method calls than can be stored on the method-call stack, an error known as a(n) **stackoverflow** occurs.
k) The **scope** of a declaration is the portion of a program that can refer to the entity in the declaration by name.
l) It's possible to have several methods with the same name that each operate on different types or numbers of arguments. This feature is called method **overloading**.

6.2 For the class Craps in Fig. 6.8, state the scope of each of the following entities:

a) the variable randomNumbers.

Class body

b) the variable die1.

block that defines method rollDice's body.

c) the method rollDice.

Class body

d) the method main.

Class body

e) the variable sumOfDice.

block that defines method main's body.

6.3 Write an application that tests whether the examples of the Math class method calls shown in Fig. 6.2 actually produce the indicated results.

```
1  public class MathTest
2  {
3  public static void main(String[] args)
4  {
5  System.out.printf("Math.abs(23.7) = %f%n", Math.abs(23.7));
6  System.out.printf("Math.abs(0.0) = %f%n", Math.abs(0.0));
7  System.out.printf("Math.abs(-23.7) = %f%n", Math.abs(-23.7));
8  System.out.printf("Math.ceil(9.2) = %f%n", Math.ceil(9.2));
9  System.out.printf("Math.ceil(-9.8) = %f%n", Math.ceil(-9.8));
10 System.out.printf("Math.cos(0.0) = %f%n", Math.cos(0.0));
11 System.out.printf("Math.exp(1.0) = %f%n", Math.exp(1.0));
12 System.out.printf("Math.exp(2.0) = %f%n", Math.exp(2.0));
13 System.out.printf("Math.floor(9.2) = %f%n", Math.floor(9.2));
14 System.out.printf("Math.floor(-9.8) = %f%n", Math.floor(-9.8));
15 System.out.printf("Math.log(Math.E) = %f%n", Math.log(Math.E));
16 System.out.printf("Math.log(Math.E * Math.E) = %f%n",
17 Math.log(Math.E * Math.E));
18 System.out.printf("Math.max(2.3, 12.7) = %f%n", Math.max(2.3, 12.7));
19 System.out.printf("Math.max(-2.3, -12.7) = %f%n",
20 Math.max(-2.3, -12.7));
21 System.out.printf("Math.min(2.3, 12.7) = %f%n", Math.min(2.3, 12.7));
22 System.out.printf("Math.min(-2.3, -12.7) = %f%n",
```

```
23  Math.min(-2.3, -12.7));
24  System.out.printf("Math.pow(2.0, 7.0) = %f%n", Math.pow(2.0, 7.0));
25  System.out.printf("Math.pow(9.0, 0.5) = %f%n", Math.pow(9.0, 0.5));
26  System.out.printf("Math.sin(0.0) = %f%n", Math.sin(0.0));
27  System.out.printf("Math.sqrt(900.0) = %f%n", Math.sqrt(900.0));
28  System.out.printf("Math.tan(0.0) = %f%n", Math.tan(0.0));
29  }
30  }
```

6.4 Give the method header for each of the following methods:

a) Method hypotenuse, which takes two double-precision, floating-point arguments side1 and side2 and returns a double-precision, floating-point result.

```
1  double hypotenuse(double side1, double side2)
```

b) Method smallest, which takes three integers x, y and z and returns an integer.

```
1  int smallest(int x, int y, int z)
```

c) Method instructions, which does not take any arguments and does not return a value.

[Note: Such methods are commonly used to display instructions to a user.]

```
1  void instructions()
```

d) Method intToFloat, which takes integer argument number and returns a float.

```
1  float intToFloat(int number)
```

6.5 Find the error in each of the following program segments. Explain how to correct the error.

a)

```
1 void g()
2 {
3 System.out.println("Inside method g");
4 void h()
5 {
6 System.out.println("Inside method h");
7 }
8 }
```

Error: Method h is declared within method g.

Correction: Move the declaration of h outside the declaration of g.

b)

```
1 int sum(int x, int y)
2 {
3 int result;
4 result = x + y;
5 }
```

Error: The method is supposed to return an integer, but does not.

Correction: Delete the variable result, and place the statement

```
1 return x + y;
```

in the method, or add the following statement at the end of the method body:

```
1 return result;
```

c)

```
1 void f(float a);
```

```
2 {
3 float a;
4 System.out.println(a);
5 }
```

Error: The semicolon after the right parenthesis of the parameter list is incorrect, and

the parameter a should not be redeclared in the method.

Correction: Delete the semicolon after the right parenthesis of the parameter list, and

delete the declaration float a;.

d)

```
1 void product()
2 {
3 int a = 6, b = 5, c = 4, result;
4 result = a * b * c;
5 System.out.printf("Result is %d%n", result);
6 return result;
7 }
```

Error: The method returns a value when it's not supposed to.

Correction: Change the return type from void to int.

6.6 Declare method sphereVolume to calculate and return the volume of the sphere. Use the following statement to calculate the volume:

```
1 import java.util.Scanner;
2 public class Sphere
3 {
4 // obtain radius from user and display volume of sphere
5 public static void main(String[] args)
```

```
6 {
7 Scanner input = new Scanner(System.in);
8 System.out.print("Enter radius of sphere: ");
9 double radius = input.nextDouble();
10 System.out.printf("Volume is %f%n", sphereVolume(radius));
11 } // end method determineSphereVolume
12 // calculate and return sphere volume
13 public static double sphereVolume(double radius)
14 {
15 double volume = (4.0 / 3.0) * Math.PI * Math.pow(radius, 3);
16 return volume;
17 } // end method sphereVolume
18 } // end
```

6.7 What is the value of x after each of the following statements is executed?

a) x = Math.abs(-7.5);

> 7.5

b) x = Math.floor(5 + 2.5);

> 7.0

c) x = Math.abs(9) + Math.ceil(2.2);

> 12.0

d) x = Math.ceil(-5.2);

> -5.0

e) x = Math.abs(-5) + Math.abs(4);

> 9.0

f) x = Math.ceil(-6.4) - Math.floor(5.2);

> -11.0

g) x = Math.ceil(-Math.abs(-3 + Math.floor(-2.5)));

> -6.0

6.8 (Parking Charges)

```java
import java.util.Scanner;

class Solution
{
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      int hours_parked = scanner.nextInt();
      scanner.close();

       double fee = calculateParkingFee(hours_parked);
       System.out.println(hours_parked + " hours parked = " + fee);
   }

   public static double calculateParkingFee(int hours_parked) {
      double fee = 0.0;
      if (hours_parked <= 3) {
         fee = 2.00;
      } else if (hours_parked <= 19) {
         fee = 2.00 + 0.50 * (hours_parked - 3);
      } else {
         fee = 10.00;
      }
      return fee;
   }
}
```

6.9 (Rounding Numbers)

```java
import java.util.Scanner;

class Solution
{
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      double x = scanner.nextDouble();
      scanner.close();
```

```
9
10      double y = Math.floor(x + 0.5);
11      System.out.println("Original: " + x + ", Rounded: " + y);
12    }
13 }
```

6.10 (Rounding Numbers)

```
1  import java.util.Scanner;
2
3  public class Solution
4  {
5    public static void main(String[] args) {
6      Scanner scanner = new Scanner(System.in);
7      double x = scanner.nextDouble();
8      scanner.close();
9
10      System.out.println("Original: " + x + "\n" +
11                  "Round to Integer: " + roundToInteger(x) + "\n" +
12                  "Round to Tenths: " + roundToTenths(x) + "\n" +
13                  "Round to Hundredths: " + roundToHundredths(x) + "\n" +
14                  "Round to Thousandths: " + roundToThousandths(x));
15    }
16
17    public static double roundToInteger(double x) {
18      return Math.floor(x + 0.5);
19    }
20
21    public static double roundToTenths(double x) {
22      return Math.floor(x * 10 + 0.5) / 10;
23    }
24
25    public static double roundToHundredths(double x) {
26      return Math.floor(x * 100 + 0.5) / 100;
27    }
28
29    public static double roundToThousandths(double x) {
30      return Math.floor(x * 1000 + 0.5) / 1000;
31    }
32 }
```

6.11 Answer each of the following questions:
a) What does it mean to choose numbers “at random”?

Ans: Choosing at random means to choose a integer or floating point number without learning it's value randomly.

b) Why is the nextInt method of class SecureRandom useful for simulating games of chance?
Ans: Random can be useful to develop neutral mobs and mods in games, like in a car racing game, the direction of other bot cars.

c) Why is it often necessary to scale or shift the values produced by a SecureRandom object?
Ans: Scalling or shifting may be necessary to get the output in between a number or a range.

d) Why is computerized simulation of real-world situations a useful technique?
Ans: With simulations we can observe real world situations and other complex systems, which enables researchers to learn more about a system.

6.12 Write statements that assign random integers to the variable n in the following ranges:

a) 2 ≤ n ≤ 6.
```
> int n = random.nextInt(5) + 2;
```

b) 4 ≤ n ≤ 50.
```
> int n = random.nextInt(47) + 4;
```

c) 0 ≤ n ≤ 7.
```
> int n = random.nextInt(8);
```

d) 1000 ≤ n ≤ 1030.
```
> int n = random.nextInt(31) + 1000;
```

e) –5 ≤ n ≤ 1.
```
> int n = random.nextInt(7) - 5;
```

f) –2 ≤ n ≤ 9.
```
> int n = random.nextInt(12) - 2;
```

6.13 Write statements that will display a random number from each of the following sets:

a) 0, 3, 6, 9, 12.
```
1  n = (int) (Math.random() * 5) * 3;
```

b) 1, 2, 4, 8, 16, 32.

```
n = (int) Math.pow(2, (int) (Math.random() * 6));
```

c) 10, 20, 30, 40.

```
n = (int) (Math.random() * 4 + 1) * 10;
```

6.14 (Floor and Ceil)

```
public class Round {
  public static void main(String[] args) {
    Round round = new Round();

    System.out.println(round.myFloor(3.5));
    System.out.println(round.myFloor(-3.5));
    System.out.println(round.myCeil(-3.5));
    System.out.println(round.myCeil(-3.6));
  }

  double myFloor(double x) {
    return Math.floor(x);
  }

  double myCeil(double x) {
    return Math.ceil(x);
  }
}
```

6.15 (Hypotenuse Calculations)

Ans:

```
public class Triangle {
  public static void main(String[] args) {
    double side1, side2, hypotenuse;
    side1 = 3.0;
    side2 = 4.0;
    hypotenuse = hypotenuse(side1, side2);
    System.out.println("Given sides of lengths " + side1 + " and " + side2 + "
the hypotenuse is " + hypotenuse);

    side1 = 5.0;
     side2 = 12.0;
     hypotenuse = hypotenuse(side1, side2);
     System.out.println("Given sides of lengths " + side1 + " and " + side2 + "
the hypotenuse is " + hypotenuse);
```

```
13
14        side1 = 8.0;
15        side2 = 15.0;
16        hypotenuse = hypotenuse(side1, side2);
17        System.out.println("Given sides of lengths " + side1 + " and " + side2 + "
the hypotenuse is " + hypotenuse);
18     }
19
20     static double hypotenuse(double a, double b) {
21        return Math.sqrt(a * a + b * b);
22     }
23  }
```

Chapter 07

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

7.1 Fill in the blanks in each of the following statements:

a) Lists and tables of values can be stored in **arrays** and **collection**.
b) An array is a group of **variables** (called elements or components) containing values that all have the same **type**.
c) The **enhanced for statement** allows you to iterate through an array's elements without using a counter.
d) The number used to refer to a particular array element is called the element's **index**.
e) An array that uses two indices is referred to as a(n) **two dimensional** array.
f) Use the enhanced for statement **for(double d: array)** to walk through double array numbers.
g) Command-line arguments are stored in **an array of String**.
h) Use the expression **args.length** to receive the total number of arguments in a command line. Assume that command-line arguments are stored in String[] args.
i) Given the command *java MyClass test*, the first command-line argument is **test**.
j) A(n) **ellipsis (...)** in the parameter list of a method indicates that the method can receive a variable number of arguments.

7.2 Determine whether each of the following is true or false. If false, explain why.

a) An array can store many different types of values.

Ans: False. An array can store only one type of value.

b) An array index should normally be of type float.

Ans: False. An array index can be of integer type.

c) An individual array element that's passed to a method and modified in that method will contain the modified value when the called method completes execution.

Ans: False. A called method receives and manipulates a copy of the value of such an element, so modifications do not affect the original value.

d) Command-line arguments are separated by commas.

Ans: False. They are seperated by whitespace.

7.3 Perform the following tasks for an array called fractions:

a) Declare a constant ARRAY_SIZE that's initialized to 10.

```
1  final int ARRAY_SIZE = 10;
```

b) Declare an array with ARRAY_SIZE elements of type double, and initialize the elements to 0.

```
1  double[] fractions = new double[ARRAY_SIZE];
```

c) Refer to array element 4.

```
1  fractions[3]
```

d) Assign the value 1.667 to array element 9.

```
1  fractions[8] = 1.667;
```

e) Assign the value 3.333 to array element 6.

```
1  fractions[5] = 3.333;
```

f)Sum all the elements of the array, using a for statement. Declare the integer variable x as a control variable for the loop.

```
1  double total = 0.0;
2      for (int x = 0; x < fractions.length; x++)
3              total += fractions[x];
```

7.4 Perform the following tasks for an array called table:

a) Declare and create the array as an integer array that has three rows and three columns.

```
1  int[][] table = new int[ARRAY_SIZE][ARRAY_SIZE];
```

b) How many elements does the array contain? **Nine**.

c) Use a for statement to initialize each element of the array to the sum of its indices. Assume that the integer variables x and y are declared as control variables.

```
1  for (int x = 0; x < table.length; x++)
2      for (int y = 0; y < table[x].length; y++)
3              table[x][y] = x + y;
```

7.5 Find and correct the error in each of the following program segments:

a) final int ARRAY_SIZE = 5;
ARRAY_SIZE = 10;

Ans: Assigning a value to a constant after it has been initialized.

Correction: Assign the correct value to the constant in a final int ARRAY_SIZE declaration or declare another variable.

b) Assume

```
1  int[] b = new int[10];
2      for (int i = 0; i <= b.length; i++)
3              b[i] = 1;
```

Ans: *Referencing an array element outside the bounds of the array (b[10]).*

Correction: Change the <= operator to <.

c) Assume `int[][] a = {{1, 2}, {3, 4}};`

`a[1, 1] = 5;`

Ans: *Array indexing is performed incorrectly.*

7.6 Fill in the blanks in each of the following statements:

a) A one-dimensional array p contains five elements. The names of the third and fourth elements are **p[2]** and **p[3]**.

b) A one-dimensional array k has three elements. The statement **k[1] = 2** sets the value of the second element to 2.

c) A statement to declare a two-dimensional int array r that has 3 rows and 4 columns is **int r[][] = new int[3][4]**.

d) A 5-by-6 array contains **5** rows, **6** columns and **30** elements.

e) The name of the element in column 5 and row 6 of an array d is **d[5][4]**.

7.7 Determine whether each of the following is true or false. If false, explain why.

a) To refer to a particular location or element within an array, we specify the name of the array and the order of the element in the array, assuming ordering starts at position 1.

Ans: False. Ordering starts at position 0.

b) An array declaration initializes the elements in the array to the integer 0 by default.

Ans: True.

c) To indicate that 200 locations should be reserved for integer array p, you write the declaration int p[] = new int[200];

Ans: True.

d) For an application that initializes the elements of a twenty-element integer array to zero, it is preferable to use some kind of loop.

Ans: False. Java will by default initialize them to 0.

e) To access all the elements in a two-dimensional array using a loop, the traversal across rows must be done in the outer loop and the traversal across columns in the inner loop.

Ans: True.

7.8 Write Java statements to accomplish each of the following tasks:

a) Display the value of the tenth element of array r.

```
1       System.out.println(array[9]);
```

b) Initialize each of the six elements of one-dimensional integer array g to -1.

```
1  public class Test {
2     public static void main(String[] args) {
3        int[] array = new int[6];
4        for (int i = 0; i < array.length; i++) {
5           array[i] = -1;
6        }
7     }
8  }
```

c) Find the maximum of the first one-hundred elements of floating-point array c.

```
1  float maximum = 0;
2        for (int i = 0; i < array.length; i++) {
3           if (array[i] > maximum) {
4              maximum = array[i];
5           }
6        }
```

d) Copy a hundred-element array a into a hundred-element array b, but in reverse order.

```
1  int[] b = new int[100];
2        for (int i = a.length - 1; i >= 0; i--) {
3           b[i] = a[i];
4        }
```

e) Compute the product of the third to the tenth elements, both inclusive, in a hundred element integer array w.

```
1  int product = 1;
2        for (int i = 2; i < 9; i++) {
3           product *= b[i];
4        }
```

7.9 Consider a two-by-three integer array t.

a) Write a statement that declares and creates t.

```
1  public class Test {
```

```
2    public static void main(String[] args) {
3       int[][] t = new int[2][3];
4    }
5 }
```

b) How many rows does t have?
Ans: 2

c) How many columns does t have?
Ans: 3

d) How many elements does t have?
Ans: 6

e) Write access expressions for all the elements in row 1 of t.

```
t[1][0] = 1;
t[1][1] = 2;
t[1][2] = 3;
```

f) Write access expressions for all the elements in column 2 of t.

```
t[0][2] = 1;
t[1][2] = 2;
```

g) Write a single statement that sets the element of t in row 0 and column 1 to zero.

```
t[0][1] = 1;
```

h) Write individual statements to initialize each element of t to zero.

```
1       t[0][0] = 0;
2       t[0][1] = 0;
3       t[0][2] = 0;
4       t[1][0] = 0;
5       t[1][1] = 0;
6       t[1][2] = 0;
```

i) Write a nested for statement that initializes each element of t to zero.

```
1  public class Test {
2     public static void main(String[] args) {
3        int[][] t = new int[2][3];
4        for (int row = 0; row < t.length; row++) {
5           for (int column = 0; column < t[row].length; column++) {
6              t[row][column] = 0;
```

```
        }
    }
  }
}
```

j) Write a nested for statement that inputs the values for the elements of t from the user.

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
    int[][] t = new int[2][3];
    Scanner input = new Scanner(System.in);
    int value = input.nextInt();
    input.close();
    for (int row = 0; row < t.length; row++) {
      for (int column = 0; column < t[row].length; column++) {
        t[row][column] = value;
      }
    }
  }
}
```

k) Write a series of statements that determines and displays the smallest value in t.

```
int smallest = t[0][0];
    for (int row = 0; row < t.length; row++) {
      for (int column = 0; column < t[row].length; column++) {
        if (t[row][column] < smallest) {
          smallest = t[row][column];
        }
      }
    }
    System.out.printf("Smallest value in t is %d%n", smallest);
```

l) Write a single printf statement that displays the elements of the first row of t.

```
for (int column = 0; column < t[0].length; column++) {
        System.out.printf("%d ", t[0][column]);
    }
```

m) Write a statement that totals the elements of the third column of t. Do not use repetition.

```
int total = t[0][2] + t[1][2];
```

n) Write a series of statements that displays the contents of t in tabular format. List the column indices as headings across the top, and list the row indices at the left of each row.

```
System.out.printf("%s%8s%8s%8s%n", " ", "0", "1", "2");
for (int row = 0; row < t.length; row++) {
  System.out.printf("%d", row);
  for (int column = 0; column < t[row].length; column++) {
    System.out.printf("%8d", t[row][column]);
  }
  System.out.println();
}
```

7.10 (Pixel Quantization)

```
public class PixelQuantization {
  public static void main(String[] args) {
    int[] pixelValues = {15, 35, 50, 75, 90, 105, 130, 155, 180, 200};
    quantizePixels(pixelValues);

    for (int value : pixelValues) {
      System.out.print(value + " ");
    }
  }

  private static void quantizePixels(int[] pixels) {
    for (int i = 0; i < pixels.length; i++) {
      int value = pixels[i];

      if (value >= 0 && value <= 20) {
        pixels[i] = 10;
      } else if (value <= 40) {
        pixels[i] = 30;
      } else if (value <= 60) {
        pixels[i] = 50;
      } else if (value <= 80) {
        pixels[i] = 70;
      } else if (value <= 100) {
        pixels[i] = 90;
      } else if (value <= 120) {
        pixels[i] = 110;
      } else if (value <= 140) {
```

```
28              pixels[i] = 130;
29          } else if (value <= 160) {
30              pixels[i] = 150;
31          } else if (value <= 180) {
32              pixels[i] = 170;
33          } else {
34              pixels[i] = 190;
35          }
36       }
37    }
38 }
```

7.11 Write statements that perform the following one-dimensional-array operations:

a) Set elements of index 10–20, both inclusive, of integer array counts to zero.

```
1       for (int i = 10; i <= 20; i++) {
2          ar[i] = 0;
3       }
```

b) Multiply each of the twenty elements of integer array bonus by 2.

```
1       for (int i = 0; i < 20; i++) {
2          ar[i] *= 2;
3       }
```

c) Display the ten values of integer array bestScores, each on a new line.

```
1        for (int i = 0; i < 10; i++) {
2          System.out.println(ar[i]);
3       }
```

7.12 (Duplicate Elimination)

```
 1 import java.util.Scanner;
 2
 3 public class RemoveDuplicates {
 4    public static void main(String[] args) {
 5       int[] numbers = new int[10];
 6       initializeArray(numbers, -1);
 7       inputNumbers(numbers);
 8       removeDuplicates(numbers);
 9       displayArray(numbers);
10    }
11
12    private static void initializeArray(int[] array, int value) {
```

```
13        for (int i = 0; i < array.length; i++) {
14            array[i] = value;
15        }
16    }
17
18    private static void inputNumbers(int[] numbers) {
19        Scanner scanner = new Scanner(System.in);
20        System.out.println("Enter ten numbers: ");
21        for (int i = 0; i < numbers.length; i++) {
22            numbers[i] = scanner.nextInt();
23        }
24        scanner.close();
25    }
26
27    private static void removeDuplicates(int[] numbers) {
28        for (int i = 0; i < numbers.length; i++) {
29            for (int j = i + 1; j < numbers.length; j++) {
30                if (numbers[i] == numbers[j]) {
31                    numbers[j] = -1; // Set duplicate value to -1
32                }
33            }
34        }
35    }
36
37    private static void displayArray(int[] array) {
38        System.out.println("Array without duplicate values:");
39
40        for (int value : array) {
41            if (value != -1) {
42                System.out.print(value + " ");
43            }
44        }
45    }
46 }
```

7.13 Label the elements of a five-by-six two-dimensional array table to indicate the order in which they're set to zero by the following program segment:

```
1  for (int col = 0; col < 6; col++)
2  {
3      for (int row = 0; row < 5; row++)
4      {
```

```
5             table[row][col] = 0;
6     }
7 }
```

1	6	11	16	21	26
2	7	12	17	22	27
3	8	13	18	23	28
4	9	14	19	24	29
5	10	15	20	25	30

Chapter 08

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

8.1 Fill in the blanks in each of the following statements:

a) A(n) static **import on demand** imports all static members of a class.
b) *String class static* method **format** is similar to method *System.out.printf*, but returns a formatted String rather than displaying a String in a command window.
c) If a method contains a local variable with the same name as one of its class's fields, the local variable **shadows** the field in that method's scope.
d) The public methods of a class are also known as the class's **public services** or **public interface**.
e) A(n) **single-type-import** declaration specifies one class to import.
f) If a class declares constructors, the compiler will not create a(n) **default constructor**.
g) An object's **toString** method is called implicitly when an object appears in code where a String is needed.
h) Get methods are commonly called **accessor methods**, or **query methods**.
i) A(n) **predicate** method tests whether a condition is true or false.
j) For every enum, the compiler generates a static method called **values** that returns an array of the enum's constants in the order in which they were declared.
k) Composition is sometimes referred to as a(n) **has-a** relationship.
l) A(n) **enum**. declaration contains a comma-separated list of constants.
m) A(n) **static** variable represents classwide information that's shared by all the objects of the class.
n) A(n) **single static import** declaration imports one static member.
o) The **principle of least privilege** states that code should be granted only the amount of privilege and access that it needs to accomplish its designated task.
p) Keyword **final** specifies that a variable is not modifiable after initialiation in a declaration or constructor.
q) A(n) **type-import-on-demand** declaration imports only the classes that the program uses from a particular package.
r) Set methods are commonly called **mutator methods** because they typically change a value.
s) Use class **BigDecimal** to perform precise monetary calculations.
t) Use the **throw** statement to indicate that a problem has occurred.

8.2 Explain the notion of package access in Java. Explain the negative aspects of package access.

Package access refers to the visibility of of certain classes or fields, within the same package.
Using package access may limit certain features for other developers and make the code even more complicated. In certain cases, exposing certain methods can cause security issue in the entire system.

8.3 State an example where you can reuse the constructor of a parent class in Java.

In Java, when creating a subclass that extends a parent class, we can reuse the constructor of the parent class using the **super()** keyword.
For example, let's consider a scenario where we have a parent class called "Vehicle" with a constructor that initializes common attributes like "make" and "model." By extending this class to create a subclass "Car," we can reuse the constructor of the "Vehicle" class using "super()" to initialize shared attributes such as "make" and "model" within the "Car" subclass, streamlining the code and maintaining consistency.

8.4 Cylinder Class

```
1  class Cylinder {
2     float radius = 1;
3     float height = 1;
4
5     float getVolume() {
6        return (float) (Math.PI * Math.pow(radius, 2) * height);
7     }
8
9     public float getRadius() {
10       return radius;
11    }
12
13    public void setRadius(float radius) {
14       this.radius = radius;
15    }
16
17    public float getHeight() {
```

```
18        if (height < 0) {
19            throw new IllegalArgumentException("Height cannot be negative");
20        }
21        return height;
22    }
23
24    public void setHeight(float height) {
25        if (height < 0) {
26            throw new IllegalArgumentException("Height cannot be negative");
27        }
28        this.height = height;
29    }
30 }
31
32 public class Exercise_4 {
33    public static void main(String[] args) {
34        Cylinder cylinder = new Cylinder();
35        cylinder.setRadius(5);
36        cylinder.setHeight(10);
37        System.out.println("Volume of cylinder is " + cylinder.getVolume());
38    }
39 }
```

8.5 Modifying the Internal Data Representation of a Class

```
1  public class Time2 {
2     // private int hour; // 0 - 23
3     // private int minute; // 0 - 59
4     private int second; // 0 - 59
5     // Time2 no-argument constructor:
6     // initializes each instance variable to zero
7
8     public Time2() {
9        this(0, 0, 0); // invoke constructor with three arguments
10    }
11
12    // Time2 constructor: hour supplied, minute and second defaulted to 0
```

```
13    public Time2(int hour) {
14       this(hour, 0, 0); // invoke constructor with three arguments
15    }
16
17    // Time2 constructor: hour and minute supplied, second defaulted to 0
18    public Time2(int hour, int minute) {
19       this(hour, minute, 0); // invoke constructor with three arguments
20    }
21
22    // Time2 constructor: hour, minute and second supplied
23    public Time2(int hour, int minute, int second) {
24       if (hour < 0 || hour >= 24)
25          throw new IllegalArgumentException("hour must be 0-23");
26       if (minute < 0 || minute >= 60)
27          throw new IllegalArgumentException("minute must be 0-59");
28       if (second < 0 || second >= 60)
29          throw new IllegalArgumentException("second must be 0-59");
30        // this.hour = hour;
31       setHour(hour);
32       // this.minute = minute;
33       setMinute(minute);
34       this.second = second;
35    }
36
37    // Time2 constructor: another Time2 object supplied
38    public Time2(Time2 time) {
39       // invoke constructor with three arguments
40       this(time.getHour(), time.getMinute(), time.getSecond());
41    }
42
43    // Set Methods
44    // set a new time value using universal time;
45    // validate the data
46    public void setTime(int hour, int minute, int second) {
47       if (hour < 0 || hour >= 24)
48          throw new IllegalArgumentException("hour must be 0-23");
```

```
49      if (minute < 0 || minute >= 60)
50         throw new IllegalArgumentException("minute must be 0-59");
51      if (second < 0 || second >= 60)
52         throw new IllegalArgumentException("second must be 0-59");
53      // this.hour = hour;
54      setHour(hour);
55      // this.minute = minute;
56      setMinute(minute);
57      this.second = second;
58   }
59
60   // validate and set hour
61   public void setHour(int hour) {
62      if (hour < 0 || hour >= 24)
63         throw new IllegalArgumentException("hour must be 0-23");
64      // this.hour = hour;
65      this.second += hour * 3600;
66   }
67
68   // validate and set minute
69   public void setMinute(int minute) {
70      if (minute < 0 || minute >= 60)
71         throw new IllegalArgumentException("minute must be 0-59");
72      // this.minute = minute;
73      this.second += minute * 60;
74   }
75
76   // validate and set second
77   public void setSecond(int second) {
78      if (second < 0 || second >= 60)
79         throw new IllegalArgumentException("second must be 0-59");
80      this.second += second;
81   }
82
83   // Get Methods
84   // get hour value
```

```
85    public int getHour() {
86       return second / 3600;
87       // return hour;
88    }
89
90    // get minute value
91    public int getMinute() {
92       return (second % 3600) / 60;
93    }
94
95    // get second value
96    public int getSecond() {
97       return second % 60;
98       // return second;
99    }
100
101    // convert to String in universal-time format (HH:MM:SS)
102    public String toUniversalString() {
103       return String.format(
104             "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
105    }
106
107    // convert to String in standard-time format (H:MM:SS AM or PM)
108    public String toString() {
109       return String.format("%d:%02d:%02d %s",
110             ((getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12),
111             getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
112    }
113
114 }
```

8.6 Savings Account Class

```
1  class SavingsAccount {
2     private static double annualInterestRate;
3     private double savingsBalance;
4
```

```
5    public SavingsAccount(double savingsBalance) {
6      this.savingsBalance = savingsBalance;
7    }
8
9    public static void modifyInterestRate(double newInterestRate) {
10       annualInterestRate = newInterestRate;
11    }
12
13    public void calculateMonthlyInterest() {
14       savingsBalance += savingsBalance * annualInterestRate / 12;
15    }
16
17    public double getSavingsBalance() {
18       return savingsBalance;
19    }
20 }
21
22 public class Problem_6 {
23    public static void main(String[] args) {
24       SavingsAccount saver1 = new SavingsAccount(2000.00);
25       SavingsAccount saver2 = new SavingsAccount(3000.00);
26
27       SavingsAccount.modifyInterestRate(0.04);
28       for (int i = 0; i < 12; i++) {
29          saver1.calculateMonthlyInterest();
30          saver2.calculateMonthlyInterest();
31       }
32
33       System.out.printf("Saver 1 balance: %.2f\n",
saver1.getSavingsBalance());
34       System.out.printf("Saver 2 balance: %.2f\n",
saver2.getSavingsBalance());
35
36       SavingsAccount.modifyInterestRate(0.05);
37       saver1.calculateMonthlyInterest();
38       saver2.calculateMonthlyInterest();
```

```
39
40        System.out.printf("Saver 1 balance: %.2f\n",
saver1.getSavingsBalance());
41        System.out.printf("Saver 2 balance: %.2f\n",
saver2.getSavingsBalance());
42    }
43 }
```

8.7 (Enhancing Class Time2)

```
 1 class Time2 {
 2    private int hour; // 0 - 23
 3    private int minute; // 0 - 59
 4    private int second; // 0 - 59
 5    // Time2 no-argument constructor:
 6    // initializes each instance variable to zero
 7
 8    public Time2() {
 9       this(0, 0, 0); // invoke constructor with three arguments
10    }
11
12    // Time2 constructor: hour supplied, minute and second defaulted to 0
13    public Time2(int hour) {
14       this(hour, 0, 0); // invoke constructor with three arguments
15    }
16
17    // Time2 constructor: hour and minute supplied, second defaulted to 0
18    public Time2(int hour, int minute) {
19       this(hour, minute, 0); // invoke constructor with three arguments
20    }
21
22    // Time2 constructor: hour, minute and second supplied
23    public Time2(int hour, int minute, int second) {
24       if (hour < 0 || hour >= 24)
25          throw new IllegalArgumentException("hour must be 0-23");
26       if (minute < 0 || minute >= 60)
27          throw new IllegalArgumentException("minute must be 0-59");
```

```
28      if (second < 0 || second >= 60)
29         throw new IllegalArgumentException("second must be 0-59");
30      this.hour = hour;
31      this.minute = minute;
32      this.second = second;
33   }
34
35   // Time2 constructor: another Time2 object supplied
36   public Time2(Time2Second time) {
37      // invoke constructor with three arguments
38      this(time.getHour(), time.getMinute(), time.getSecond());
39   }
40
41   // Set Methods
42   // set a new time value using universal time;
43   // validate the data
44   public void setTime(int hour, int minute, int second) {
45      if (hour < 0 || hour >= 24)
46         throw new IllegalArgumentException("hour must be 0-23");
47      if (minute < 0 || minute >= 60)
48         throw new IllegalArgumentException("minute must be 0-59");
49      if (second < 0 || second >= 60)
50         throw new IllegalArgumentException("second must be 0-59");
51      this.hour = hour;
52      this.minute = minute;
53      this.second = second;
54   }
55
56   // validate and set hour
57   public void setHour(int hour) {
58      if (hour < 0 || hour >= 24)
59         throw new IllegalArgumentException("hour must be 0-23");
60      this.hour = hour;
61   }
62
63   // validate and set minute
```

```java
64    public void setMinute(int minute) {
65       if (minute < 0 || minute >= 60)
66          throw new IllegalArgumentException("minute must be 0-59");
67       this.minute = minute;
68    }
69
70    // validate and set second
71    public void setSecond(int second) {
72       if (second < 0 || second >= 60)
73          throw new IllegalArgumentException("second must be 0-59");
74       this.second = second;
75    }
76
77    // Get Methods
78    // get hour value
79    public int getHour() {
80       return hour;
81    }
82
83    // get minute value
84    public int getMinute() {
85       return minute;
86    }
87
88    // get second value
89    public int getSecond() {
90       return second;
91    }
92
93    // convert to String in universal-time format (HH:MM:SS)
94    public String toUniversalString() {
95       return String.format(
96             "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
97    }
98
99    // convert to String in standard-time format (H:MM:SS AM or PM)
```

```
100     public String toString() {
101         return String.format("%d:%02d:%02d %s",
102                 ((getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12),
103                 getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
104     }
105 
106     public void incrementHour() {
107         if (hour == 23) {
108             hour = 0;
109         } else {
110             hour++;
111         }
112     }
113 
114     public void incrementMinute() {
115         if (minute == 59) {
116             minute = 0;
117             incrementHour();
118         } else {
119             minute++;
120         }
121     }
122 
123     public void tick() {
124         if (second == 59) {
125             second = 0;
126             incrementMinute();
127         } else {
128             second++;
129         }
130     }
131 }
132 
133 public class Time {
134     public static void main(String[] args) {
135         Time2 t1 = new Time2(); // 00:00:00
```

```
136         Time2 t2 = new Time2(2); // 02:00:00
137         Time2 t3 = new Time2(12, 25, 42); // 12:25:42
138
139         System.out.println(t1.toUniversalString());
140         System.out.println(t2.toUniversalString());
141         System.out.println(t3.toUniversalString());
142
143         t1.setTime(13, 27, 6);
144         t1.tick();
145         System.out.println(t1.toUniversalString());
146
147         t2.setHour(22);
148         t2.setMinute(34);
149         t2.setSecond(45);
150         t2.incrementHour();
151         System.out.println(t2.toString());
152
153         t3.setTime(23, 59, 59);
154         t3.tick();
155         System.out.println(t3.toString());
156      }
157   }
```

8.8 (Enhancing Class Date)

```
 1  // Fig. 8.7: Date.java
 2  // Date class declaration.
 3  public class Date {
 4     private int month; // 1-12
 5     private int day; // 1-31 based on month
 6     private int year; // any year
 7     private static final int[] daysPerMonth = { 0, 31, 28, 31, 30, 31, 30, 31, 31,
30, 31, 30, 31 };
 8
 9     // constructor: confirm proper value for month and day given the year
10      public Date(int month, int day, int year) {
11         if (year <= 0)
12            throw new IllegalArgumentException("year (" + year + ") must be
greater than 0");
```

```
13        // check if month in range
14        if (month <= 0 || month > 12)
15           throw new IllegalArgumentException(
16                 "month (" + month + ") must be 1-12");
17        // check if day in range for month
18        if (day <= 0 || (day > daysPerMonth[month] && !(month == 2 && day ==
29)))
19           throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
20        // check for leap year if month is 2 and day is 29
21        if (month == 2 && day == 29 && !(year % 400 == 0 || (year % 4 == 0 &&
year % 100 != 0)))
22           throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
23        this.month = month;
24        this.day = day;
25        this.year = year;
26        System.out.printf("Date object constructor for date %s%n", this);
27     }
28
29     public void nextDay() {
30        if (day == daysPerMonth[month]) {
31           day = 1;
32           if (month == 12) {
33              month = 1;
34              year++;
35           } else
36              month++;
37        } else
38           day++;
39     }
40
41     public void nextMonth() {
42        if (month == 12) {
43           month = 1;
44           year++;
45        } else
46           month++;
47     }
48
49     public void nextYear() {
50        year++;
```

```
51    }
52
53    // return a String of the form month/day/year
54    public String toString() {
55       return String.format("%d/%d/%d", month, day, year);
56    }
57
58    public static void main(String[] args) {
59       Date date = new Date(12, 30, 2020);
60       System.out.println(date);
61       date.nextMonth();
62       System.out.println(date);
63       date.nextYear();
64       System.out.println(date);
65    }
66 }
67 // end class Date
```

8.9 Write code that generates n random numbers in the range 10 – 100. [Note: Only import the Scanner and SecureRandom classes.].

```
1  import java.util.Scanner;
2  import java.security.SecureRandom;
3
4  public class RandGen {
5     public static void main(String[] args) {
6        Scanner scanner = new Scanner(System.in);
7        System.out.print("Enter the number of random numbers to generate:
");
8        int num = scanner.nextInt();
9        scanner.close();
10
11       for (int i = 0; i < num; i++) {
12          System.out.printf("%d ", getRandomInt(10, 100));
13       }
14    }
15
16    private static int getRandomInt(int i, int j) {
17       return i + new SecureRandom().nextInt(j - i + 1);
18    }
19 }
```

8.10 Write an enum type Food, whose constants (APPLE , BANANA, CARROT) take two parameters —the type (vegetable or fruit), and number of calories. Write a program to test the Food enum so that it displays the enum names and their information.

```
enum Food {
   APPLE("fruit", 95), BANANA("fruit", 105), CARROT("vegetable", 25);

   private final String type;
   private final int calories;

   Food(String type, int calories) {
      this.type = type;
      this.calories = calories;
   }

   public String getType() {
      return type;
   }

   public int getCalories() {
      return calories;
   }
}

public class Enum {
   public static void main(String[] args) {
      System.out.println();
      for (Food food : Food.values()) {
         System.out.printf("%s: %s, %d calories%n", food, food.getType(),
food.getCalories());
      }
   }
}
```

8.11 (Complex Numbers)

```
public class Complex {
   private float real;
   private float imaginary;

   public Complex(float real, float imaginary) {
      this.real = real;
      this.imaginary = imaginary;
```

```
8    }
9
10    public Complex() {
11       this(0, 0);
12    }
13
14    public void addNumber(float real, float imaginary) {
15       this.real += real;
16       this.imaginary += imaginary;
17    }
18
19    public void substrNumber(float real, float imaginary) {
20       this.real = real - this.real;
21       this.imaginary = imaginary - this.imaginary;
22    }
23
24    public String toString() {
25       return String.format("%f + %fi", this.real, this.imaginary);
26    }
27
28    public static void main(String[] args) {
29       Complex complex = new Complex(1, 2);
30       System.out.println(complex);
31       complex.addNumber(3, 4);
32       System.out.println(complex);
33       complex.substrNumber(5, 6);
34       System.out.println(complex);
35    }
36 }
```

8.12 (Date and Time Class)

```
1  // Date class declaration.
2  class Date {
3     private int month; // 1-12
4     private int day; // 1-31 based on month
5     private int year; // any year
6     private static final int[] daysPerMonth = { 0, 31, 28, 31, 30, 31, 30, 31, 31,
30, 31, 30, 31 };
7
8     // constructor: confirm proper value for month and day given the year
9     public Date(int month, int day, int year) {
10       if (year <= 0)
```

```
11          throw new IllegalArgumentException("year (" + year + ") must be
greater than 0");
12       // check if month in range
13       if (month <= 0 || month > 12)
14          throw new IllegalArgumentException(
15                "month (" + month + ") must be 1-12");
16       // check if day in range for month
17       if (day <= 0 || (day > daysPerMonth[month] && !(month == 2 && day ==
29)))
18          throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
19       // check for leap year if month is 2 and day is 29
20       if (month == 2 && day == 29 && !(year % 400 == 0 || (year % 4 == 0 &&
year % 100 != 0)))
21          throw new IllegalArgumentException("day (" + day + ") out-of-range
for the specified month and year");
22       this.month = month;
23       this.day = day;
24       this.year = year;
25       // System.out.printf("Date object constructor for date %s%n", this);
26    }
27
28    public void nextDay() {
29       if (day == daysPerMonth[month]) {
30          day = 1;
31          if (month == 12) {
32             month = 1;
33             year++;
34          } else
35             month++;
36       } else
37          day++;
38    }
39
40    public void nextMonth() {
41       if (month == 12) {
42          month = 1;
43          year++;
44       } else
45          month++;
46    }
47
```

```
48    public void nextYear() {
49       year++;
50    }
51
52    // return a String of the form month/day/year
53    public String toString() {
54       return String.format("%d/%d/%d", month, day, year);
55    }
56 }
57 // end class Date
58
59 class Time {
60    private int hour; // 0 - 23
61    private int minute; // 0 - 59
62    private int second; // 0 - 59
63    // Time2 no-argument constructor:
64    // initializes each instance variable to zero
65
66    public Time() {
67       this(0, 0, 0); // invoke constructor with three arguments
68    }
69
70    // Time2 constructor: hour supplied, minute and second defaulted to 0
71    public Time(int hour) {
72       this(hour, 0, 0); // invoke constructor with three arguments
73    }
74
75    // Time2 constructor: hour and minute supplied, second defaulted to 0
76    public Time(int hour, int minute) {
77       this(hour, minute, 0); // invoke constructor with three arguments
78    }
79
80    // Time2 constructor: hour, minute and second supplied
81    public Time(int hour, int minute, int second) {
82       if (hour < 0 || hour >= 24)
83          throw new IllegalArgumentException("hour must be 0-23");
84       if (minute < 0 || minute >= 60)
85          throw new IllegalArgumentException("minute must be 0-59");
86       if (second < 0 || second >= 60)
87          throw new IllegalArgumentException("second must be 0-59");
88       this.hour = hour;
89       this.minute = minute;
```

```
90        this.second = second;
91     }
92
93     // Time2 constructor: another Time2 object supplied
94     public Time(Time2Second time) {
95        // invoke constructor with three arguments
96        this(time.getHour(), time.getMinute(), time.getSecond());
97     }
98
99     // Set Methods
100    // set a new time value using universal time;
101    // validate the data
102    public void setTime(int hour, int minute, int second) {
103       if (hour < 0 || hour >= 24)
104          throw new IllegalArgumentException("hour must be 0-23");
105       if (minute < 0 || minute >= 60)
106          throw new IllegalArgumentException("minute must be 0-59");
107       if (second < 0 || second >= 60)
108          throw new IllegalArgumentException("second must be 0-59");
109       this.hour = hour;
110       this.minute = minute;
111       this.second = second;
112    }
113
114    // validate and set hour
115    public void setHour(int hour) {
116       if (hour < 0 || hour >= 24)
117          throw new IllegalArgumentException("hour must be 0-23");
118       this.hour = hour;
119    }
120
121    // validate and set minute
122    public void setMinute(int minute) {
123       if (minute < 0 || minute >= 60)
124          throw new IllegalArgumentException("minute must be 0-59");
125       this.minute = minute;
126    }
127
128    // validate and set second
129    public void setSecond(int second) {
130       if (second < 0 || second >= 60)
131          throw new IllegalArgumentException("second must be 0-59");
```

```
132       this.second = second;
133    }
134
135    // Get Methods
136    // get hour value
137    public int getHour() {
138       return hour;
139    }
140
141    // get minute value
142    public int getMinute() {
143       return minute;
144    }
145
146    // get second value
147    public int getSecond() {
148       return second;
149    }
150
151    // convert to String in universal-time format (HH:MM:SS)
152    public String toUniversalString() {
153       return String.format(
154             "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
155    }
156
157    // convert to String in standard-time format (H:MM:SS AM or PM)
158    public String toString() {
159       return String.format("%d:%02d:%02d %s",
160             ((getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12),
161             getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
162    }
163
164    public void incrementHour() {
165       if (hour == 23) {
166          hour = 0;
167       } else {
168          hour++;
169       }
170    }
171
172    public void incrementMinute() {
173       if (minute == 59) {
```

```
174             minute = 0;
175             incrementHour();
176         } else {
177             minute++;
178         }
179     }
180
181     public void tick() {
182         if (second == 59) {
183             second = 0;
184             incrementMinute();
185         } else {
186             second++;
187         }
188     }
189 }
190
191 public class DateAndTime {
192     private Date date;
193     private Time time;
194
195     public DateAndTime(Date date, Time time) {
196         this.date = date;
197         this.time = time;
198     }
199
200     public void tick() {
201         time.tick();
202         if (time.getHour() == 0 && time.getMinute() == 0 &&
time.getSecond() == 0) {
203             date.nextDay();
204         }
205     }
206
207     public void incrementMinute() {
208         time.incrementMinute();
209         if (time.getHour() == 0 && time.getMinute() == 0) {
210             date.nextDay();
211         }
212     }
213
214     public void incrementHour() {
```

```
215         time.incrementHour();
216         if (time.getHour() == 0) {
217             date.nextDay();
218         }
219     }
220 
221     public void incrementMonth() {
222         date.nextMonth();
223     }
224 
225     public void incrementYear() {
226         date.nextYear();
227     }
228 
229     public String toString() {
230         return String.format("%s %s", date, time);
231     }
232 
233     public static void main(String[] args) {
234         Date date = new Date(1, 5, 2023);
235         Time time = new Time(23, 59, 59);
236         DateAndTime dateAndTime = new DateAndTime(date, time);
237 
238         System.out.println(dateAndTime);
239 
240         dateAndTime.incrementHour();
241         dateAndTime.tick();
242         System.out.println(dateAndTime);
243     }
244 }
```

Knowledge should be free and open source 🙃

Chapter 14

Exercise solutions of Java TM How to Program Early Objects TENTH edition

Paul Deitel • Harvey Deitel

BLANK PAGE

Yet an another sample XD

14.1 State whether each of the following is true or false. If false, explain why.

a) When String objects are compared using ==, the result is true if the Strings contain the same values.

Ans: False. '==' operator will check whether they share the same memory or not.

b) A String can be modified after it's created.

Ans: False. A string is an immutable object and thus can't be modified.

14.2 For each of the following, write a single statement that performs the indicated task:

a) Compare the string in s1 to the string in s2 for equality of contents.

```
1  s1.equals(s2)
```

b) Append the string s2 to the string s1, using +=.

```
1  s1 += s2;
```

c) Determine the length of the string in s1 .

```
1  s1.length();
```

14.3 (Palindromes)

```
1  public class Palindrome {
2    public static void main(String[] args) {
3      String s = "madam";
4      System.out.println(isPalindrome(s));
5    }
6
7    static boolean isPalindrome(String s) {
8      int n = s.length();
9      for (int i = 0; i < n/2; i++) {
10       if (s.charAt(i) != s.charAt(n-i-1)) {
11         return false;
12       }
13     }
14     return true;
15   }
16 }
```

14.4 (Comparing Portions of Strings)

```java
import java.util.Scanner;

public class Compare {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.println("Enter first string: ");
    String s1 = input.nextLine();
    System.out.println("Enter second string: ");
    String s2 = input.nextLine();
     System.out.println("Enter number of characters to be compared: ");
     int n = input.nextInt();
     System.out.println("Enter starting index of the comparison: ");
     int i = input.nextInt();
     input.close();

     if (s1.regionMatches(true, i, s2, i, n)) {
       System.out.println("The strings are equal.");
     } else {
       System.out.println("The strings are not equal.");
     }
   }
}
```

14.5 (Random Sentences)

```java
public class SentenceGeneration {
  String[] article = { "the", "a", "one", "some", "any" };
  String[] noun = { "boy", "girl", "dog", "town", "car" };
  String[] verb = { "drove", "jumped", "ran", "walked", "skipped" };
  String[] preposition = { "to", "from", "over", "under", "on" };

  int randomNum(int min, int max) {
    return (int) (Math.random() * (max - min + 1) + min);
  }

   String randomArticle() {
```

```
12        return article[randomNum(0, article.length - 1)];
13     }
14
15     String randomNoun() {
16        return noun[randomNum(0, noun.length - 1)];
17     }
18
19     String randomVerb() {
20        return verb[randomNum(0, verb.length - 1)];
21     }
22
23     String randomPreposition() {
24        return preposition[randomNum(0, preposition.length - 1)];
25     }
26
27     String randomSentence() {
28        String sentence = randomArticle() + " " + randomNoun() + " " +
randomVerb() + " " + randomPreposition() + " "
29              + randomArticle() + " " + randomNoun() + ".";
30        return sentence.substring(0, 1).toUpperCase() + sentence.substring(1);
31     }
32
33     public static void main(String[] args) {
34        SentenceGeneration sentenceGeneration = new
SentenceGeneration();
35        for (int i = 0; i < 20; i++) {
36           System.out.println(sentenceGeneration.randomSentence());
37        }
38     }
39  }
```

14.6 (Project: Limericks)

```
1  public class Limericks {
2     String[] threeRhymer = { "There was a young lady of station\n", "I love
man was her sole exclamation\n",
3           "Isle of Man is the true explanation\n" };
```

```
4     String[] twoRhymer = { "But when men cried, \"You flatter\"\n", "She
replied, \"Oh! no matter!\n" };
5
6     int randomNum(int min, int max) {
7        return (int) (Math.random() * (max - min + 1) + min);
8     }
9
10     String threeRimeGen() {
11        return threeRhymer[randomNum(0, threeRhymer.length - 1)];
12     }
13
14     String twoRimeGen() {
15        return twoRhymer[randomNum(0, twoRhymer.length - 1)];
16     }
17
18     String randomSentence() {
19        String sentence = threeRimeGen() + threeRimeGen() + twoRimeGen()
+ twoRimeGen() + threeRimeGen();
20        return sentence.substring(0, 1).toUpperCase() + sentence.substring(1);
21     }
22
23     public static void main(String[] args) {
24        Limericks sentenceGeneration = new Limericks();
25        for (int i = 0; i < 20; i++) {
26           System.out.println(sentenceGeneration.randomSentence());
27        }
28     }
29  }
```

14.7 (Pig Latin)

```
1  import java.util.Scanner;
2
3  public class PigLatin {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.println("Enter a sentence: ");
```

```
7       String sentence = input.nextLine();
8       input.close();
9
10       String[] words = sentence.split(" ");
11       for (String word : words) {
12          System.out.print(word.substring(1) + word.charAt(0) + "ay ");
13       }
14    }
15 }
```

14.8 (Tokenizing Telephone Numbers)

```
1  import java.util.Scanner;
2
3  public class TokeinizingTelephone {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.println("Enter a telephone number: ");
7        String telephoneNumber = input.nextLine();
8        input.close();
9
10       String[] tokens = telephoneNumber.split("[()\\- ]");
11       String areaCode = tokens[1];
12       String firstThreeDigits = tokens[3];
13       String lastFourDigits = tokens[4];
14       String phoneNumber = firstThreeDigits + lastFourDigits;
15
16       System.out.println("Area code: " + areaCode);
17       System.out.println("Phone number: " + phoneNumber);
18    }
19 }
```

14.9 (Displaying a Sentence with Its Words Reversed)

```
1  import java.util.Scanner;
2
3  public class ReverseSentence {
4     public static void main(String[] args) {
5        Scanner input = new Scanner(System.in);
6        System.out.println("Enter a sentence: ");
7        String sentence = input.nextLine();
8        input.close();
```

```
 9
10        String[] words = sentence.split(" ");
11        for (int i = words.length - 1; i >= 0; i--) {
12           System.out.print(words[i] + " ");
13        }
14     }
15  }
```

14.10 (Longest Word in a Sentence)

```
 1  import java.util.Scanner;
 2
 3  public class LongestWord {
 4     public static void main(String[] args) {
 5        Scanner input = new Scanner(System.in);
 6        System.out.println("Enter a sentence: ");
 7        String sentence = input.nextLine();
 8        input.close();
 9
10         String[] words = sentence.split(" ");
11         int maxLength = 0;
12         String longest_word = "";
13         for (String word : words) {
14            if (word.length() > maxLength) {
15               longest_word = word;
16               maxLength = word.length();
17            }
18         }
19
20         System.out.println("The longest word is: " + longest_word);
21     }
22  }
```

Knowledge should be free and open source 🙃

The simplicity is very **beautiful** and sometimes very, very **cursed**!

Pardon my mistakes...