**Chapter 1: Introduction to Computers, Programs, and Java™**

**\*Page 15**

**------------**

Towards the bottom of the page in Figure 1.6, the text states:

“Welcome to **Java**” is displayed on the console

However, it should be:

“Welcome to Java**!**” is displayed on the console

\*note the missing exclamation mark in the text

**Page 19**

**----------**

Towards the top of the page, the text states:

“A single space should be added on both sides of a binary operator, as shown in (a), **rather in** (b).”

The wording in the segment that is colored red might be grammatically correct as-is, but perhaps change to:

“…as shown in (a), **rather** **than** **in** (b).”

**Page 23**

**----------**

**1.** The Note in the top half of this page states:

“This section **and next** section **introduce two** most popular Java IDEs: NetBeans and Eclipse.”

The segments that are colored red above might be grammatically correct as-is, but perhaps change to:

“This section **and the next** section **introduce the two** most popular Java…”

**2.** Just a few lines down, the text states:

“For instructions on downloading and **installing latest** version of…”

Perhaps change to:

“For instructions on downloading and **installing the latest** version of…”

**Chapter 2: Elementary Programming**

**\*Page 46**

**-----------**

In Table 2.3, for the multiplication example, perhaps change **300\*30** to **300 \* 30** (the latter has a space before and after the multiplication operator).

**Page 47**

**----------**

Towards the top of the page, there is a typo in the graphic. The text states:

“Note: Day 0 **is** a week is Sunday”

The word “is” that is colored red above should be changed to “**in**”.

**Page 54**

----------

Towards the middle of the page, there seems to be a missing word in the text. The text states:

“Figure 2.8 The **System.currentTimeMillis( ) returns** the number of…”

Perhaps insert the word “**method**” before the word “returns”.

**\*Page 57**

-----------

In Table 2.5, there are two typos in the leftmost column. The text states “**––**var” and

“var**––**”

The prefix and postfix decrement operators need a small gap in the middle, as in “**--**”.

**\*Pages 63 - 64**

**------------------**

**1.** Listing 2.9 has “**e.g.**” inside the string literals (lines 9, 17 & 21). However, page 64, at the top of the page showing console output, has the output as “**for example**” 3 times.

**2.** Perhaps it was typed as-is deliberately, but on page 64 towards the middle, the text states: “examples in this **booktext** will use int for…”

Perhaps change “**booktext**” to “**textbook**”.

**\*Pages 65 - 66**

**------------------**

**1.** In Listing 2.10, line 10 in the source code is missing a comma inside the string literal after the word “**example**”.

The comma is indeed included in the next page in the graphic that shows the console output.

**2.** In line 37 there is an extra space inside the string literal. The code is:

System.out.println(“ ” + numberOfQuarters + “ quarters**[]**”);

\***[]** denotes the extra, unnecessary space

**3.** Lines 36 – 40 all have the println( ) method to display certain values such as numberOfOneDollars. Every string inside the println( ) methods begins with a space character. For example, line 36 has the code:

System.out.println(**“ ”** + numberOfOneDollars + “ dollars”);

However, the text does not accurately depict the console output because the graphic does not match the println( ) methods. The console output graphic shows the rough equivalent of:

**[][][][][][][][]**11 dollars

**[][][][][][][][]**2 quarters

**[][][][][][][][]**0 dimes

**[][][][][][][][]**1 nickels

**[][][][][][][][]**1 pennies

as opposed to:

**[]**11 dollars

**[]**2 quarters

**[]**0 dimes

**[]**1 nickels

**[]**1 pennies

**Page 68**

**----------**

Towards the middle of the page, in the code example which shows what not to do,

the text states:

System.out.print(**“Enter a double value: ”**);

The words and quotation marks inside the print method are colored purple, perhaps they should be colored green for stylistic purposes.

**Chapter 3: Selections**

**\*Pages 80 - 81**

**------------------**

**1.** Towards the top of the page, CheckPoint Question 3.2.3 asks:

“Can the following conversions involving casting be allowed?”

The first two statements that this question references are:

boolean b = true;

**i** = (int)(b);

There appears to be a typo in the text because the variable “i” that is colored red above needs to be declared as: **int** **i** = (int)(b);

**2.** The left margin of this page contains the phrase: if statement**?**

The question mark that is colored red above appears to be unnecessary.

**3.** At the top of page 81, the text states:

“The flowchart of the preceding statement is shown in Figure 3.1b.”

The “preceding statement” is:

if (radius >= 0) {

area = radius \* radius \* PI;

System.out.println(“The area for the circle of **radius ” +**

**radius** + “ is ” + area);

}

In Figure 3.1b, however, the text concatenates the strings inside the println( ) a little differently. The corresponding strings in Figure 3.1b are:

System.out.println(“The area for the circle of” + **“ radius ” + radius** “ is ” + area);

Perhaps **modify Figure 3.1b** to match the corresponding strings in page 81.

**\*Page 83**

**-----------**

CheckPoint question 3.4.2,code samples (a) and (b), have typos:

The strings “**is** even” and “**is** odd” should have a space, i.e., “**[]**is even” and “**[]**is odd”in 4 different println( ) statements and perhaps add a period after “odd” in both (a) and (b) for stylistic consistency.

**\*Page 85**

**-----------**

In CheckPoint Question 3.5.1, int variable “z” is not properly initialized within the if statement. The text states:

**z** = x + y;

At first, I thought that the reader is supposed to assume that it has been initialized already, but then I saw that CheckPoint 3.5.2 does initialize it as:

**int z** = x + y;

**\*Page 86**

-----------

**1.** The first 5 codeboxes from the top have an extra space character that will be displayed on the console. The code is:

System.out.println(“The area**[]**” + “[]is ” + area);

Perhaps remove the extra space that is colored red above.

**2.** The third codebox from the top has two different fonts for the quotation marks in the println( ).

**\*Page 87**

**------------**

At the bottom of the page, there is an extra decimal after the 5 that will not be displayed in the console if typed as it is in the println( ). The code in the text is:

System.out.println(x + “ is approximately 0.5”);

will display

0.5000000000000001 is approximately 0.5.

Should be: 0.5000000000000001 is approximately 0.5

**Page 89**

**----------**

Listing 3.3, line 7, has the keyword **int** inside the casting parentheses colored green as opposed to purple-ish and it should be bold for stylistic consistency.

**\*Page 90**

**-----------**

1. Listing 3.3, line 28 has the words “ **should be** ” inside the println( ) but the bottom laptop icon on the same page has the word “**is**” instead.

2. In Listing 3.3, line 26, the string has a period after the word “**wrong**”. The second graphic from the top with the laptop icon needs a period after “**wrong**”. So does the magnifying glass graphic.

**\*Pages 93 – 94**

**-------------------**

In Listing 3.5, lines 9-11 have a print( ) method which instructs the user to enter 0, 1, 2 or 3 based on his/her tax-filing status. The code is:

9 **System.out.print**(“(0-Single filer, 1-married jointly or ” +

10 “qualifying widow(er), 2-married separately, 3-head of ” +

11 “household) Enter the filing status: ”);

However, the console output on page 94 has 3 lines of output for the aforementioned

print( ) method. I do not think escape sequences are introduced yet at this point in the text.

**\*Page 95**

**-----------**

CheckPoint Question 3.9.1 has a code box (the one to the right of the page) that contains the **logical and operator** even though it has not been introduced yet at this point in the text. It is introduced immediately after this CheckPoint question.

**\*Page 101**

**-------------**

In Listing 3.8,the code in line 35 has a println( ) with the string: “Sorry**,** no match”

However, the display output on the same page shows “Sorry**:** no match”

\*Note: the colon should be replaced with a comma

**Page 105**

**-----------**

1. There appears to be an extra space in CheckPoint Question 3.13.2. The text appears to read: switch (x +**[][]**3) as opposed to switch (x +**[]**3)

2. The text is missing a closing parenthesis after the word “operands.”

The text states: “The symbols ? and : appearing together is called a conditional operator **(**also known as a *ternary operator* because it uses three operands[ **) ]**. It is…”

Should be: “The symbols ? and : appearing together is called a conditional operator **(**also known as a *ternary operator* because it uses three operands**)**. It is…”

**\*Page 106**

**-------------**

At the top of this page, before CheckPoint 3.14.1, the text states:

“Conditional expressions can be embedded. For example, the following code assigns 1,0, or -1 to status if **n1 > n1**, n1 = = n2, or n1 < n2:

status = n1 > n2? 1: (n1 == n2? 0: -1);”

The expression highlighted in red should be: **n1 > n2**

**\*Page 107**

**-------------**

CheckPoint Question 3.15.2 asks:

**“True or false? All the binary operators except = are left associative.”**

Online, the answer is listed as “true.” However, isn’t it technically correct to say that the answer is false because the assignment operator **AND** the augmented assignment operators are all right associative? Towards the middle of the page, the text states: “All binary operators except assignment operators are *left associative*.”

Shouldn’t CheckPoint Question 3.15.2 instead ask:

**“True or false? All the binary operators except assignment operators are left associative.”**

**Page 108**

**------------**

In Section 3.16, the text has a bullet-point list of some of the most common IDE debugging features. The first two bullet-points have italicized text while the other four bullet-points have bold text.

**Chapter 4: Mathematical Functions, Characters, and Strings**

**Page 122**

**-----------**

**1.** Towards the middle of the page, the KeyPoint for Section 4.2 has a typo. It reads “**lass**” as opposed to “**class**”.

**2.** The text states: “You have already used the pow(a, b) method to compute…and the **random( )** method for generating…”

“pow(a, b)” is stylized bold and colored green, whereas “random( )” is printed normally

**\*Page 123**

-------------

In Table 4.2, the description of the sqrt(x) method has the expression **x>=0** which appears to need two spaces, as in **x >= 0** (**x[]>=[]0**).

**\*Page 124**

**-------------**

**1.** The text states: “This method generates a random double value greater than or equal to 0.0 and less than 1.0 (**0** <= Math.random( ) < 1.0).”

Perhaps change the **0** to **0.0**

**2.** Towards the bottom of the page, in Section 4.2.5, the text states:

**“(50 + int)**(Math.random( ) \* 50); 🡪 Return a random integer between 50 and 99.”

Should be: **50 + (int)**(Math.random( ) \* 50);

Also, perhaps mention in the text that “between 50 and 99” means **inclusive the two end numbers.**

**\*Page 126**

**-------------**

**1.** CheckPoint 4.2.4 states:

“Write a statement that converts **PI** / 7 to an angle in degrees and assigns the result to a variable.”

Perhaps change to:

“Write a statement that converts **Math.PI** / 7 to an angle…”

**2.** CheckPoint 4.2.5.has two incorrect answers online. In the physical textbook, the CheckPoint challenges the reader to:

“Write an expression that obtains a random integer between 34 and 55. Write an expression that obtains a random integer between 0 and 999. Write an expression that obtains a random number between 5.5 and 55.5.”

Due to previous examples, the reader would be correct to assume that the text means inclusive the two end numbers. However, the answers online are:

(a) 34 + (int)(Math.random( ) \* **(55-34)**)

Which only returns a random number between 34 and 54 as opposed to 34 and 55.

**The answer provided for (c) online has the same error.**

**3.** The green text under the Section 4.3 header states:

“**A** character data type represents a single character.”

Perhaps change to:

“**The** character data type represents a single character.”

**Page 128**

**-----------**

Possible stylistic issue. In the bottom half of this page, the text demonstrates how to include quotation mark characters inside a string literal. The text shows the output as:

He said **“Java is fun”**

Throughout the text, the entire string is almost always/always colored green. In this example, only “Java is fun” is colored green perhaps because the text wants to emphasize that portion of the output. However, perhaps the entire string should be green, as in:

**He said “Java is fun”**

**\*Page 131**

**-------------**

The text states:

For example, the following code declares **message** to be a string with the value **“Welcome to Java”**.

I realize that the text might have been written as-is on purpose, but in case it wasn’t, perhaps remove the double quotation marks from **Welcome to Java** since it’s already colored green and the double quotation marks are not part of the value itself. Either way, the first quotation marks, before the ‘W’ in “Welcome” are a different style of font.

**\*Page 133**

**-------------**

**1.** Towards the bottom of this page, at the bottom of Section 4.4.3, the text states:

message += “ and Java is fun”;

So the new message is “Welcome to Java and Java is **fun.**”

I believe there should be no period after “fun” in the string above.

**2.** Also, towards the bottom of the page, there is a missing space inside a string.The text states:

“The output is “i + j is 12” because “**i + j is**” is concatenated with the value of i first.”

Shouldn’t the text be changed to include a space after “is”, as in:“**i + j is[]**”?

**\*Page 134**

**-------------**

At the middle of the page, the text has the code:

Scanner input = new Scanner(System.in);

**System.out.println**(“Enter a line: ”);

String s = input.nextLine( );

System.out.println(“The line entered is: ” + s);

The laptop graphic which shows the console output seems incorrect. It shows:

Enter a line: **Welcome to Java**

The line entered is Welcome to Java

For the laptop icon graphic to be accurate, perhaps change **println( )** to **print( )**.

**Page 137**

**-----------**

Possible stylistic issue. Section 4.4.8 starts at the top of this page, and it provides the following example that demonstrates how to obtain a substring from a string:

String message = **“Welcome to Java”**;

message = message.substring(**0,11**) + **“HTML”**;

The string **message** now becomes **Welcome** to **HTML**.

Perhaps change the color of the word “to” in the third line above from black to green, as in: The string **message** now becomes **Welcome to HTML**.

**\*Pages 148-49**

**------------------**

At the bottom of the page, the **last printf( ) has a space before \n**, which if my understanding is correct, affects the figure on top of page 149. The text has the output as:

**5.6[][][][][]**

But perhaps it should technically be:

**5.6[][][][][][]**

**Chapter 5: Loops**

**\*Page 161**

**-------------**

Towards the middle of the page, the text contains the code:

System.out.**printIn**(“Welcome to Java!”);

\*Note: “println” is typed with an uppercase “I” instead of lowercase “l.”

**\*Page 162**

**-------------**

The text under the Caution header states:

“For example, the following loop displays Welcome to **Java** 101 times rather…”

However, the actual code right underneath has “Welcome to Java**!**” with an exclamation mark inside the println.

**Page 166**

**------------**

In Listing 5.4, lines 5,6,7 & 9 contain **comments that have black font** (as opposed to green).

**\*Page 168**

-------------

Towards the middle of this page, the text has code for controlling a loop with user confirmation or a sentinel value. The code is:

char continueLoop = ‘Y’;

while (continueLoop = = ‘Y’) {

// loop body

System.out.print(“Enter Y to continue and N to quit: ”);

continueLoop = input.**getLine( )**.charAt(0);

However, the program doesn’t run correctly, but it worked when I changed it to:

continueLoop = input.**nextLine( )**.charAt(0);

**\*Page 170**

**-------------**

Towards the middle of the page, the quotation marks in the println are a different style of font and they would cause a syntax error typed as they are because the second quotation marks are facing the wrong way. The text is:

System.out.println(**“**The sum is *“* + sum);

**Page 177**

**-----------**

CheckPoint Question 5.8.2 contains code which appears to have an extra space:

while (sum <**[][]**10000) {

\*Note: Each pair of square brackets signifies a space character.

**\*Pages 178 - 179**

**---------------------**

In Listing 5.7,

**1.** In line 5, the spaces inside the println( ) are not represented in the display output graphic on the next page.

**2.** On page 179, the console output has a black-and-white dashed line which appears to be a stylistic error.

**3.** Line 12 has one dash that is longer than the others inside the println( ).

**Page 181**

**-----------**

Towards the middle of the page, text states:

“The fundamental problem is **the** floating-point numbers are represented by approximation.”

Should “the” be changed to “**that**”?

**\*Page 186**

**-------------**

In Listing 5.12**,** lines 13 and 14 in have **quotation marks inside the println( )** methods that are stylistically different (the font type) and facing the wrong way.

**\*Pages 187 - 188**

**---------------------**

**1.** The Note at the bottom of page 187 states:

“The continue statement is **always inside a loop**.”

However, the Note at top of page 188 states:

“The break and continue statements in Java are different from goto statements. They operate only in a loop **or a switch statement**.”

The explanations above might be grammatically correct, but the wording has the potential to cause confusion. A reader might interpret the wording on page 188 to mean that a continue statement can work in a “loop or switch statement” as opposed to only in a loop. At least I know I did, and I had to re-read the text for clarification.

Perhaps modify the Note on page 188 to state something similar to:

“The break and continue statements in Java are different from goto statements. **A break statement can operate only in a loop or switch statement. A continue statement can operate only in a loop. The break statement breaks out of the loop or switch, and the continue statement breaks out of the current loop iteration**.”

**2.** The note towards the bottom of page 188 contains the following code which finds a smallest factor:for (**int factor = 2**; n % factor != 0; factor++)**;**

I have previously brought-up this for-loop due to the semicolon at the end of the loop and suggested that perhaps it was not meant to be there. However, after reviewing the loop and its context, it seems like the semi-colon was indeed meant to be there in order to demonstrate how simple a loop can be. This for-loop, however, appears to have a logic issue. **The smallest-factor value will be inaccessible** because the variable “factor” is declared in the for-loop, which has an empty body. Thus, perhaps “int factor” should be declared and initialized outside the of the for-loop or declared outside of the for-loop but initialized in the for-loop. Two possible solutions are:

for (factor = 2; n % factor != 0; factor++); // int factor; declared outside loop

for ( ; n % factor != 0; factor++); // int factor = 2; initialized outside loop

**CONTINUED FROM THE PREVIOUS PAGE**

Below is a sample program that will not compile if the for-loop in the text is used.

import java.util.\*;

public class FindSmallestFactor {

public static void main(String[ ] args) {

Scanner input = new Scanner(System.in);

System.out.print(“Enter an integer to find its smallest factor: ”);

int n = input.nextInt( );

for (**int factor = 2**; n % factor != 0; factor++);

// code doesn’t compile because “factor” is not accessible outside of the for loop

System.out.println(“The smallest factor of integer ” + n +

“ other than 1 is ” + factor); }}

**\*Page 192**

**-------------**

In Listing 5.15, line 8 contains the code:

System.out.println(“The first 50 prime numbers are **\n**”);

The linefeed is not accounted for in the laptop icon graphic that shows the console output.

**Chapter 6: Methods**

**\*Page 208**

**-------------**

At the top of the pagethe text states: “In a method definition, you define what the method is to do. To execute the method, you have to *call* or *invoke* it. The program that calls the **function** is called a *caller*.”

Should the word function above be changed to “**value-returning method**” or simply “**method**”, or is it technically correct to call a value-returning method a function in Java? The note on the previous page leads me to believe that the term “function” is only for other languages.

**\*Pages 208 – 209**

**----------------------**

In Listing 6.1, line 7, the string literal inside the println( ) displays:

The maximum **of**…

In Figure 6.2, the string literal inside the println( ) displays:

The maximum **between**…

Figure 6.2 corresponds to Listing 6.1, so perhaps change the word “between” to the word “of”.

**Pages 216 - 217**

**-------------------**

It seems like CheckPoint 6.5.4 is on the wrong page. CheckPoint 6.5.4 states:

“For (a) in the **preceding** question, show the contents…”

Code sample (a), however, does not come before CheckPoint 6.5.4. All four code boxes are right below (after) CheckPoint 6.5.4. It seems like CheckPoint 6.5.4 is better placed immediately before the beginning of Section 6.6 on page 217.

Another possible solution is to change CheckPoint 6.5.3 to have it state:

“What is pass by value? Show the result of **programs (a), (b), (c), and (d)**.”

And to change CheckPoint 6.5.4 to have it state:

“For (a) **below**…”

**Chapter 7: Single-Dimensional Arrays**

**Page 271**

**-----------**

Towards the bottom of the page, code box (b) has two keywords “int” and “if” that should be bold for stylistic purposes.

**Page 272**

**-----------**

Towards the bottom of the page, the text contains a small typo. The text states:

“…postconditions are the things that are true after the method is returned**:**

To better understand…”

Perhaps replace the colon (:) after the word “returned” with a period.

**Page 275**

**-----------**

Towards the bottom of the page, the text that shows the output omits **1. 2. 3. 4.** which are included inside the string literal in the code above.

The text states:

The output of the preceding code is as follows:

Index is 4 1. Index is 4

Index is -6 **SHOULD BE 🡪** 2. Index is -6

Index is 0 3. Index is 0

Index is -4 4. Index is -4

**Chapter 9: Objects and Classes**

**\*Page 326**

**-------------**

In Listing 9.1 line 12, the string “**is**” needs two spaces, as in “**[]**is**[]**”.

**\*Page 336**

**-------------**

CheckPoint Question 9.5.5 (b) provides a sample class named ShowErrors and then asks the reader to find what is wrong with the program. Online, the answer is listed as:

“The program compiles fine, but it has a runtime error because variable c is null when the println statement is executed.”

However, it seems to me that the class cannot be compiled because it does not have a main method. Line 2 has the following code: **public void method1( ) {**

Even when adding a main method and a Circle class, to my limited understanding, the program does not compile. Furthermore, even if the program ran fine, the string literal inside the println in line 4 would display a message such as:

“What is radius 1.0” as opposed to, for example, “Radius is 1.0”

**\*Note:** I believe that the reason why this CheckPoint question is confusing is because stylistically, it strays from other CheckPoint questions in the text. Most CheckPoints include all the code that they want you to consider. This one, however, leaves it up to the reader to assume that a Circle class is present along with a main method somewhere. The text will almost always ask you to assume certain conditions if need be. This CheckPoint, however, does not.

**\*Note:** I first brought-up the CheckPoint Question above about 9 months ago. Now that I have a lot more experience with Java, the answer is obvious and the question is not confusing. However, I still believe that it is a good idea to add more context to the question.

**\*Page 337**

**-----------**

**1.** Towards the top of the page, the text has a small typo (comma instead of period). The text states:

“method to return the date and time as a string**,** For example, the following”

Should be:

“method to return the date and time as a string**.** For example, the following”

**2.** Also, the println( ) a few lines below appears to have an extra space, i.e., Jan 1,**[][]**1970

**Page 338**

**-----------**

There is a typo in the second note from the top. The text states:

“The random numbers generated **from the Random are** deterministic…”

Perhaps add the word “class” after the word “Random” so that the text states:

“The random numbers generated from the Random class are deterministic…”

**\*Page 340**

**------------**

The full title of Listing 9.6 is:

Listing 9.6, Circle.java (for **CircleWithStaticMembers**)

Perhaps add the word “Test” to change the title to:

Listing 9.6, Circle.java (for **TestCircleWithStaticMembers**)

**\*Page 341**

**-------------**

In Listing 9.7, lines 12 and 24 have an **escape sequence which is not shown in the console** output at the bottom of the same page. There should be two blank lines on the console output since the source code has println( ) + \n on both lines.

**Page 342**

**-----------**

**1.** Towards the top of the page, the text has a typo. The text states:

“The main method creates two circles c1 and c2 (lines 9 **and18**).”

The text needs a space between the word “and” and the number “18.”

**2.** Towards the bottom of the page, in line 3 of the code, there appears to be an extra space between “int” and “k”.

**Page 344**

**-----------**

CheckPoint Questions 9.7.1 (a) and (b). The code box in (a) defines class F. The class’s methods, as written, appear to go against convention (they are not written in camelCase) They are written as: “void **imethod**” and “static void **smethod**.”

Perhaps change to: **iMethod** and **sMethod**. Code box (b) has the same issue.

I realize that the methods might have been typed as they are on purpose, but nonetheless I choose to bring-up the issue in case the names are indeed typos.

**Page 345**

**-----------**

In Figure 9.15, the middle code box is given as follows:

package p1;

public class C2 {

**can access C1**

}

Perharps, for stylistic consistency, **add a semicolon after C1** to match the examples in the other code boxes.

**\*Page 346**

**-----------**

Figures 9.16 (a) and (b) have typos. The letter “s” is lowercase six times when it should be uppercase. The text reads: (**s**tring[ ] args) { AND **s**ystem.out.println

**\*Page 347**

**-------------**

Listing 9.8 is titled: **Circle.java(for CircleWithPrivateDataFields)**

Perhaps change to: **Circle.java (for TestCircleWithPrivateDataFields)**

Note the added space between the ‘a’ and ‘(’ characters.

**\*Page 350**

**-------------**

**1.** In Listing 9.10, the console output does not accurately reflect the source code. On line 21, the code needs one more tab escape sequence. Code is “**\t\t**” but should be

“**\t\t\t**” in order for the numbers to line up correctly under “Area”.

**2.** The escape sequence \n in line 13 is not shown in the console output.

**Page 351**

**-----------**

In CheckPoint Question 9.10.1, there seems to be an **extra space on line 3** between the new operator and “Count( );”

**\*Pages 354 - 355**

**---------------------**

**1.** Listing 9.11, line 34, has the code:

System.out.println(“**\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_** \_”);

The laptop icon section that shows the display output, however, has a solid line.

**2.** There appears to be two typos at the bottom of page 355. The text states:

“We call such an object **as** immutable object and its class **as** immutable class.”

Perhaps change to:

“We call such an object **an** immutable object and its class **an** immutable class.”

**\*Page 356**

**-------------**

**1.** With regards to your response about updating page 355 to state:

“If a class is immutable, then all its **instance** data fields must be private…”

Page 356 also needs to be updated since the text states:

“For a class to be immutable, it must meet the following requirements: **All data fields** must be private.”

The text needs to be updated to state:

“For a class to be immutable, it must meet the following requirements: All **instance** data fields must be private.”

**2.** CheckPoint 9.12.3 provides the class below and asks the reader whether it is immutable:

public class A {

private int[] values;

public int[] getValues( ) {

return values;

}

}

The answer provided online is: “**No, because values is a reference type.**” However, is it correct to answer that the class is immutable because although the field “values” is a reference type, it is never initialized within class A? I unsuccessfully tried to mutate field “values” in Class A like dateCreated is mutated by class Test. The field dateCreated is mutable because it is initialized every time a client program creates a Student object. Class A, however, only has a default constructor which does not initialize “values”.

**\*Note:** I believe that the reason why this CheckPoint question is confusing is because stylistically, it strays from other CheckPoint questions in the text. Most CheckPoints include all the code that they want you to consider. This one has the reader assume certain conditions. For example, the text assumes that you will create an instance of A and that in the constructor or somewhere in the code you will eventually initialize the variable “values”. These are crucial assumptions that the student must make. Perhaps the text should be explicit about what it wants the reader to assume. The text can state:

“Is the following class immutable? **(Hint: try creating an instance of A in another class)**”

**\*Note:** I tested the solution that Dr. Liang provided (**a.getValues( )[0] = 9;**), but I am still unable to mutate the field “values” after creating an instance of class A as I do in “Sample Code” below.

**CONTINUED ON THE NEXT PAGE**

**CONTINUED FROM THE PREVIOUS PAGE**

**Sample Code**

public Class A {

private int[ ] values;

private int[ ] getValues( ) {

return values; }}

Class B {

public static void main(String[ ] args) {

A a = new A( );

a.getValues( )[0] = 9; }} // **throws NullPointerException**

The issue is that this CheckPoint asks: “Is the following class immutable?”

Class A’s method getValues( ) does indeed return the reference to the private field

“int[ ] values”. **Is that alone enough to declare a class, technically speaking, mutable even though the array is never initialized?** The statement **a.getValues( )[0] = 9;** above throws a **NullPointerException** and thus I cannot alter the field “values”. I also attempted to initialize the array outside of class A and was not successful.

**IF it is true** that for a class to be mutable I have to actually be able to access and mutate its fields (as opposed to simply accessing its null fields), then perhaps the text should be modified using one of the options below so that the answer online (which says that this class is mutable) can be accurate:

**Option 1.** is to initialize “values” outside the constructor:

private int[ ] values = new int[some number];

**Option 2.** is to initialize “values” in a constructor:

A( ) {

values = new int[some number];

}

\***Page 358**

**-------------**

In code box (b), there is an extra **\* Math.PI;** in the getArea( ) method body.

**Chapter 10: Object-Oriented Thinking**

**\*Pages 340 & 373**

**----------------------**

Towards the middle of the page, the text states:

“Constants in a class are shared by all objects of the class. Thus, constants should be declared as **final static**.”

Example given in the text: **final static double PI** = 3.14159265358979323846;

Listing 10.4, however, has “**public static final double**” for both of the aforementioned constants. I realize that the syntax “**static final**” is legal, but was wondering whether “**final static**” is preferred by convention.

**\*Page 370**

**-------------**

In Listing 10.2, line 7 has a javadoc comment (**/\*\* Default constructor \*/**) that perhaps should read “**no-arg constructor**”.

**\*Note:** I believe that by “**Default constructor**” the text means default as in the default Loan object that is going to be created if the user creates a loan without any special characteristics. Nonetheless, it is a bit confusing because Chapter 9 introduces actual “default constructors” which are public, no-arg, empty body constructors that the compiler implicitly includes in a class with no user-defined constructors.

**Page 371**

**-----------**

In Listing 10.2, line 15 has the **keyword “int” colored green**. **Perhaps it should be** **blue-ish/purple** like the other keywords.

**\*Page 375**

**-------------**

CheckPoint 10.3.1 asks: “Is the BMI class defined in Listing 10.4 immutable?”

The answer online is given as “yes.”

However, the class has two data fields (KILOGRAMS\_PER\_POUND & METERS\_PER\_INCH) which are declared public. I was wondering if the class is immutable nonetheless because the public fields are constants.

Page 355 states: “If a class is immutable, then all its data fields must be private and…”

**\*Page 376**

**-------------**

In Section 10.4.2, the text states:

“**We refer aggregation** between two objects as…”

Perhaps change to: “We refer **to** aggregation between two objects as…”

Also, at the bottom of this page, the text states:

“In Figure 10.6, **each student has only one multiplicity**-address-and each address can be shared by up to 3 students.”

**Doesn’t each student object have 2 multiplicities?** The previous page defines a multiplicity as a value “which is placed at the side of the class to specify how many of the class’s objects are involved in the relationship in UML.” My understanding is that a Student object has a multiplicity with Address and a separate multiplicity with Name.

Perhaps modify the text to state:

“In Figure 10.6, **each student has only one address** and each address can be shared by up to 3 students.”

**\*Page 377**

**-------------**

**1.** The text states:

“The **relation** “a student has a name” and “a student has an address” are implemented in the data **field** name and address in the Student class.”

Perhaps change to:

“The **relations** “a student has a name” and “a student has an address” are implemented in the data **fields** name and address in the Student class.”

**2.** The text also states:

“Figure 10.7 The **composition** relations are implemented using data fields in classes.”

However, a composition relationship only exists between the Student class and the Name class, not between the Student class and Address class. I see that there is a Note at the bottom of the page which explains why aggregation and composition are both called composition for simplicity, but perhaps that note should be at the top of page 377 instead.

**3.** Figure 10.8 is a little confusing. The figure illustrates that “A person **may have** a supervisor.” However, it has a multiplicity of 1 with “supervisor,” which confuses me a bit because I was expecting an **interval multiplicity instead**, i.e., 0..1, since a person **may** have a supervisor, but is not required to.

**\*Page 378**

**-------------**

Towards the middle of the page, the parameter name of the Course( ) constructor method should be changed to match the UML chart for the Course class on the same page.

The text states: “A Course object can be created using the constructor

Course(String **name**) by passing…”

Should be: “A Course object can be created using the constructor

Course(String **courseName**) by passing…”

**\*Page 380**

**-------------**

The text introduces a class named StackOfIntegers. The class contains the public instance method **empty( ) which returns true if the stack is empty**. However, shouldn’t the method be **renamed to isEmpty( ) in order to convey that it returns boolean**?

Empty( ) can be misinterpreted as a method to be invoked when an user wants to empty, i.e., delete all elements from the array.

**\*Page 383**

**-------------**

**1.** Towards the middle of the page, the text states:

“An instance created using valueOf **maybe** shared,…”

Perhaps change to:

“An instance created using valueOf **may be** shared,…”

**2.** Towards the bottom of the page, 4 println statements should have a space after “is.”

One example: System.out.println(“x1 = = x2 **is**” + (x1 = = x2));

which displays: “**x1 = = x2 isfalse**” as opposed to “**x1 = = x2 is false**”

**3.** The quotation marks in all 4 printlns and in the wrapper class examples on the same page are a different style than used elsewhere in the text.

**\*Page 384**

**-------------**

Towards the top of the page, the text states:

“**Float and Double,** MIN\_VALUE represents the minimum…”

Perhaps change to: “**For** Float and Double,…”

**\*Pages 387 - 388**

**---------------------**

**1.** The Note on this page explains why it is a good idea to use **new BigDecimal(String)** to obtain a predictable BigDecimal (as opposed to **new BigDecimal(double)**).

The text states:

“For example, new BigDecimal(**“1.0”**) is not **1.0**, but is actually 0.1000000000000000555…”

Shouldn’t the text instead state:

“For example, new BigDecimal(**0.1**) is not **0.1**, but is actually 0.10000000000000000555…”

**2.** The Note on this page states:

“Since a double value is approximated, **so** the result from new BigDecimal(double)…”

The word “**so**” seems to be a typo.

**3.** In Listing 10.9, line 7, **the quotation marks are facing the wrong way** **and the words in the print( ) method are not colored green**.

**4.** In Listing 10.9, line 8, **the “int” keyword is not colored blue-ish/purple**.

**5.** In Listing 10.9, line 9, perhaps **add a space after the concatenation operator** for readability.

**6.** In line 15, the **quotation marks are a differently style** than in other parts in the text.

**\*Page 389**

**-------------**

At the bottom of this page, the text has a graphic representation of String objects’ memory allocation. The String objects are created in the statements at the left of the graphic. The text makes it clear that Strings s2 and s4 will point to different memory locations because the objects are constructed using the new operator, but to really drive the point home, **perhaps add String s4 to the graphic and how it points to a separate memory location than s2** (the graphic only shows where Strings s1, s2, and s3 point to but omits representing string s4).

**\*Page 390**

**-------------**

Towards the top of the page, the text states:

“**S2** = = s4 is also false, because s2 and s4 are two different string objects.”

It seems like the variable “**S2**” got inadvertently capitalized.

**\*Page 391**

**-------------**

The text states:

“For example, the following statements all evaluate to true:

“Java is fun”.matches(“Java.\*”)

“Java is cool”.matches(“Java.\*”)

“Java is powerful”.matches(“Java.\*”) ”

Perhaps add a **semicolon after each statement**. On the same page (a few lines down), the same issue exists with the statement:

“440-02-4534”.matches(“\\d{3}-\\d{2}-\\d{4}”)

**\*Page 397**

**-------------**

The text has a typo because it terminates a statement with a colon instead of with a semicolon. The text states:

“For example, in the following statement: stringBuilder.reverse( )**:**”

**\*Page 399**

**-------------**

In Listing 10.10, lines 14 and 15 have **quotation marks** inside the println method **facing the wrong way** and they have a different style of font than used elsewhere in the text.

**Chapter 11: Inheritance and Polymorphism**

**Page 413**

**-----------**

Figure 11.1, the UML chart for the Circle class, first lists the **getPerimeter( )** method followed by the **getDiameter( ) method**. In the actual class (Listing 11.2), however, the **order of the two methods is switched**.

I do not know enough about UML to say that it is technically an issue, but I do notice that all the other fields and methods in GeometricObject, Circle, and Rectangle are in the same exact order as the UML charts.

**\*Pages 415-16**

**------------------**

In the text explaining Listing 11.2, Circle.Java & Listing 11.3 Rectangle.java, the text states:

“The keyword extends (**lines 1 and 2**) tells the compiler that…”

The keyword “extends” **only appears on line 1** in Listings 11.2 and 11.3.

The text has the same issue on page 416 where it explains Rectangle.java

**\*Page 417**

**-------------**

**1.** In Listing 11.4, line 10, the code “**Rectangle rectangle**” **appears to need bold font**.

**2.** The escape sequence **\n** in line 11 **does not seem to be shown in the console ouput** section on the same page.

**\*Page 419**

**-------------**

Line 14 has an extra space inside a string literal. The code is:

System.out.println(“(3) Performs Employee’s tasks**[]**”);

**\*Page 421**

**-------------**

Online, the answer for CheckPoint 11.3.1 is listed as:

“**The default constructor of B** attempts to invoke the default of constructor of A, but **class A’s default constructor is not defined**.”

However, to my understanding class B does not have a default constructor because it has an explicit no-arg constructor. Because B has a no-arg, no body constructor, it implicitly invokes A’s no-arg constructor, which does not exist. Thus, shouldn’t the answer be:

“**The no-arg constructor of B** attempts to invoke the no-arg constructor of A, but **class A’s no-arg constructor is not defined**.”

**Page 427**

**-----------**

The text states:

Method m (line 9) takes a parameter of the Object type. You can invoke m with any object (e.g., new GraduateStudent( ), new Student( ), new Person( ), and new Object( ) ) in lines 3-6**)**.

The last parenthesis does not have a matching one. Should be:

“…and new Object( ) ) **(**lines 3-6**)**.”

**Page 430**

**-----------**

At the top of the page, the text has sample code with the method “someMethod(Object myObject)”. Inside this method, the text states:

**… // Some lines of code**

**Perhaps change the color of the ellipsis** from green to black so that the text is stylistically consistent (e.g., the second ellipsis a few lines down).

**\*Page 431**

**-------------**

The text states:

“Casting can be done only when the source object is an instance of the target class. The program uses the instanceof operator to ensure that the source object is an instance of the target class before performing a casting **(line 15)**.”

Perhaps “**(line 15)**” should be changed to “**(lines 15 and 21)**” because in line 21 the instanceof operator is also used to ensure that “object” is an instance of Rectangle.

**\*Page 432**

**-------------**

CheckPoint 11.9.2 has two different names for the same reference variable. I am assuming that the intended name is “object” because that is the name in the (b) and (c) sections of the CheckPoint. The text states:

“GeometricObject **object1** = new GeometricObject( );

Are the following Boolean expressions true of false?

…

(**object** instance of GeometricObject)

…

(**object** instance of Circle) ”

Furthermore, part (c) of this CheckPoint asks:

“ Can the following statements be compiled?

GeometricObject object = new GeometricObject( );

Circle circle = (Circle)object; ”

The answer listed online is: “Causing a runtime exception (ClassCastException)”

It is true that running the above code will cause a ClassCastException, **but at this point in the text I am still unsure about what the technical definition of “compile” is**. For example, if I type in a nonsense statement such as “aaa;” into my IDE, I get a message saying “java: Errors occurred while compiling module…” However, running the code in (c) does not produce such a message, it displays the “Exception in thread…” message instead. Does this mean that technically speaking, it WAS able to compile but the code caused an Exception? I guess what I am trying to say is that I have difficulty answering whether it can be “compiled” because I am still unsure about the technical definition of the word. Perhaps the text does address it and I need to go back and re-read. Either way, perhaps the answer should be updated with a yes or a no to answer the question of whether the text can be compiled (e.g.: “No. Causing a runtime…”).

**\*Page 433**

**-------------**

**1.** There are possible typos in (i) and (j) in this CheckPoint. (i) states:

“Suppose the method makeAppleCider is defined in the Apple class. Can **Fruit** invoke this method? Can orange invoke this method?”

(j) states:

“Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Can **Fruit** invoke this method?

Online, however, the question changes “Fruit” to “fruit.”

Perhaps (i) in the physical text needs to be changed to “**fruit**” as in: Fruit **fruit** = new GoldenDelicious( ) in the previous page.

**2.** In Section 11.10, the text states:

“Another method defined in the Object class that is often used is the equals method. Its signature is

public boolean equals(Object **o**)

…

The default implementation of the equals method in the Object class is:

public boolean equals(Object **obj**) {

return this == obj;

} ”

Two different names are used for the same identifier (“o” and “obj”). I don’t know if it is correct-as-is, but stood-out to me.

**\*Page 434**

**-------------**

At the top of the page, the text has an example of an overriden method. The text overrides the equals( ) method in the Object class. The method is:

@Override

public boolean equals(Object o) {

if (o instance of Circle)

return radius == ((Circle)o).radius;

else

return false;

Doesn’t this method have the potential to cause confusion? For example, the programmer can invoke the method with a Circle object and a non-Circle argument such as an object from the GeometricObject class. The method would then return “false” but if the programmer is not careful, he/she will assume that the radius of the two objects is not equal, when in reality the method is returning “false” because the programmer sent an argument of the non-Circle type. Wouldn’t this issue be remedied by including a println such as:

else

System.out.println(“The object isn’t a Circle”);

return false;

Furthermore, after reading the Note a few times, I am still unsure as to whether I am understanding it correctly. My understanding is that the comparison operator (= =) is stronger than the equals method because I can check both whether two primitive type values are the same OR whether two objects point to the same memory location, whereas the equals method, without overriding it, can only check whether two objects point to the same memory location.

The Note states:

“The == operator is stronger than the equals method in that the == operator checks whether the two reference variables refer to the same object.”

Shouldn’t the Note state something such as:

“The == operator is stronger than the equals method in that the == operator can check both whether the two reference variables refer to the same object and whether two primitive type values are the same.”

**\*Page 434**

**-------------**

In CheckPoint Question 11.10.2, the text states:

“…instead, it should be equals(Object **circle**), as shown in (b).”

However, the parameter in (b) is (Object **o**), therefore the text should state:

“…instead, it should be equals(Object **o**), as shown in (b).”

**\*Page 435**

**-------------**

**1.** Figure 11.3 is an UML representation of the ArrayList class. The method on the bottom is: +set(index: int, e: E) : E

The **decription to the right**, however, **does not mention that this method returns the element at the specified index**. The description only states: “Sets the element at the specified index.” but perhaps it should state something along the lines of:

“Sets the element at the specified index. Returns the element which was previously at the specified index.”

**2.** The Note introduces a term, “AConcreteType,” but does not give a description of what AConcreteType is or mention that the term will be covered later.

**\*Page 438**

**-------------**

The text states:

“You cannot use the get(index) and set(index, element) methods if the **element** is not in the list.”

Perhaps change to:

“You cannot use the get(index) and set(index, element) methods if the **index** is not in the list.”

\***Page 439**

**-------------**

**1.** Page 439 shows the console output for Listing 11.9. The text states:

“Enter **numbers**…”

However, Listing 11.9’s line 9 on the previous page has “Enter **integers**…”

**2.** The text incorrectly references the wrong lines of code in Listing 11.9. The text states: “you can replace the code in **lines 20 and 21** using the following code:…”

should be: “you can replace the code in **lines 21 and 22** using the following code:…”

**\*Page 440**

**-------------**

**1.** In CheckPoint Question 11.11.3, the text states:

“Suppose the ArrayList list contains {“Dallas”, “Dallas”, “**Hous-**

**ton**”, “Dallas”}.”

The hyphen is enclosed inside quotation marks, therefore the answer should be:

After list.remove(“Dallas”), the list becomes {“Dallas”, “**Hous-ton**”, “Dallas”}.

I realize that this is a pedantic “correction” that perhaps was even intentionally written as-is, especially since no reader would be confused by this small issue, but nonetheless I present it just in case you deem it significant enough to address.

**2.** In Section 11.12, the bottom third of the page has the code:

String[ ] array = {“red”, “green”**,** “blue”};

The second comma (the one after the string literal “green”) is green and perhaps should be changed to black.

**\*Page 441**

**-------------**

The text states:

“This section introduces a stack class to store objects. You can use an ArrayList to implement **Stack**, as show in Listing11.10.”

I believe **Stack** should be changed to **MyStack**.

**\*Page 442**

**-------------**

In the text explaining Listing 11.10, the text states:

“The push(Object **element**) method (lines 24-26) adds the specified element to the stack.”

However, in Listing 11.10 the signature for the method is:

public void push(Object **o**)

**\*Page 444**

**-------------**

CheckPoint 11.14.1 asks the reader: “What modifier should you use **on a class** so a class in the same package can access it, but a class in a different package cannot access it?”

CheckPoint 11.14.2 asks the reader: “What modifier should you use so a class in a different package **cannot access the class** but its subclasses in any package can access it?”

Strictly speaking, the questions are asking about using modifiers on a class (as opposed to its members), therefore I believe a student can answer “default” for 11.14.1 and “not possible” for 11.14.2. After reading 11.14.2, however, a more reasonable student can assume that the questions are asking about a class’s members, not the class itself. Nonetheless, there is room for confusion, thus **perhaps the questions should clarify that they refer to a class’s members**.

**\*Page 445**

**-------------**

In Section 11.5, the Key Point states:

“Neither a final class nor a final method can be extended.”

Shouldn’t the text instead state:

“A final class can’t be extended and a final method can’t be overriden.”

**Chapter 12: Exception Handling and Text I/O**

**\*Page 453**

**-------------**

The third objective from the top states: “To distinguish exception types: Error (**fatal**) vs. Exception (**nonfatal**) and checked vs. unchecked.”

The text, however, does not use the terms fatal/nonfatal again within the chapter itself.

**\*Pages 455-56**

**------------------**

**1.** In page 455, Listing 12.2, line 16 of the code has an extra space inside the println that is perhaps not meant to be there. The line is:

System.out.println(“Divisor cannot be zero**[]**”);

**2.** In page 456, Listing 12.4, line 26 of the code has the same extra space. The line is:

“cannot be divided by zero**[]**”;

**3.** In Listing 12.2, line 7, there is a **line comment that is colored gray as opposed to green**.

**\*Page 459**

**-------------**

CheckPoint 12.2.5 asks the reader: “What is the output of the following code?”

This CheckPoint has information that has not yet been introduced in the text. First, line 6 has: throw new **Exception**(“value is too small”);

The **Exception class**, however, has not been introduced yet. It is **introduced at the bottom of the same page**.

Line 9 has: System.out.println(ex.**getMessage( )**);

The **getMessage( ) method**, however, has not been introduced yet. It is **first introduced in the note on page 463**.

CheckPoint Question 12.2.6 also asks about the Exception class and getMessage( ).

I only bring this up because there are only a few CheckPoint Questions up to this point in the text where it asks the reader a question based on details not yet introduced.

**\*Page 461**

**-------------**

Towards the top of this page, the text states:

“Runtime exceptions are represented in the RuntimeException class, which describes programming errors, such as bad casting, accessing an **out-of-bounds array**,…”

Perhaps change to: “**accessing an out-of-bounds index**” or “**accessing an out-of-bounds array index**”

**\*Page 465**

**-------------**

Towards the top of this page, the text has a note that states:

“Java forces you to deal with checked exceptions. If a method declares a checked exception (i.e., an exception other than Error or RuntimeException), you must **invoke** it in a try-catch block or…”

I was not expecting the text to use the word “**invoke**.” I was expecting “**you must handle it in a try-catch block**” or “**you must catch it in a try-catch block**.” I would not bring up this issue because I do not know enough to comment on the use of a Java word, but then I realized that on the column to the right of the Note the text states: “**catch or declare checked exceptions**” as opposed to “**invoke or declare checked exceptions**.”

**\*Page 467**

**-------------**

The text states:

“The example demonstrates declaring, throwing, and catching exceptions by modifying the setRadius method in the Circle class in Listing 9.8, **Circle.java(CircleWithPrivateDataField)**.”

Perhaps change to:

**Circle.java(for TestCircleWithPrivateDataFields)**

On the following page, the text states:

“The original Circle class remains intact except that the class name is changed to CircleWithException, a new constructor CircleWithException(newRadius) is added, and the setRadius method now declares an exception and throws it if the radius is negative.”

However, there are a few more minute ways in which CircleWithException differs from Listing 9.8 Circle.java. For example:

1. In Circle.java, the field radius is initialized with a value of 1, whereas in CircleWithException, the same field is not initialized.
2. In Circle.java, the no-arg constructor does not invoke a constructor since radius already has the initial value of 1. However, the no-arg constructor in CircleWithException does invoke another constructor.
3. In Circle.java, within the no-arg constructor the field numberOfObjects is incremented by 1 whereas in CircleWithException, numberOfObjects is incremented only in the overloaded constructor.
4. In Circle.java, line 35 has the method name as: getArea( ) whereas in CircleWithException the corresponding method is named findArea( ).
5. In Circle.java, the getArea( ) method uses the constant Math.PI whereas in CircleWithException the findArea( ) method uses 3.14159 instead.

Ultimately, both classes will produce the same results, with the exception that the setRadius method may throw an IllegalArgumentException, as stated in the text.

**\*Page 469**

**-------------**

**1.** CheckPoint Question 12.4.4 asks the reader:

“If the exception is caught in **the** catch block, will statement4 be executed?”

Perhaps change to:

“If the exception is caught in **a** catch block…” because the code that the question references has two catch blocks.

**2.** CheckPoint Question 12.4.6 has a catch-block with the parameter “**e**” which is declared to be of the Exception type. All the other exception blocks have “**ex**” as opposed to “**e**”.

**\*Page 470**

**-------------**

**1.** CheckPoint 12.4.7 has the following code: String s =“abc”;

that should perhaps be changed to: String s =**[]**“abc”;

**2.** CheckPoint 12.4.10 asks the reader:

“Does the presence of a try-catch block impose overhead when no exception occurs?”

I do not recall the text mentioning anything about try-catch blocks and their effect on overhead at this point in the text.

**\*Page 471**

**-------------**

CheckPoint 12.5.1 has sample code in which the main method invokes another method m( ) that might throw 2 different types of exceptions, Exception1 and Exception2. Shouldn’t the method header for method m( ) contain “throws Exception2” because it does not catch it? The main method declares to throw Exception2, so the reader can assume that it is a checked exception, even though the CheckPoint question does not state whether it is checked/unchecked. However, if my understanding is correct, the code will still compile IF Exception2 is assumed to be unchecked. Perhaps the CheckPoint should clarify on whether the exceptions are checked/unchecked, or add “throws Exception2” to method m( )’s header?

**\*Page 473**

**-------------**

In CheckPoint Question 12.7.1, the word “finally” in the code appears to be colored greenish as opposed to blueish-purple like “try” and “catch.”

**\*Page 475**

**-------------**

The top of this page has an UML diagram for the java.lang.Exception class. The diagram has one of the Exception constructors as: +Exception(message: String, cause: **Exception**)

The Oracle documentation, however, has “**Throwable**” instead of “**Exception**”

**Page 478**

**-----------**

The appears to be an insignificant typo in the UML diagram for the java.io.File class. The description for the mkdirs( ) method states:

“Same as mkdir( ) except that **it creates directory** along with its parent directories…”

Perhaps insert the word “**a**” after “creates” so that the description can state:

“Same as mkdir( ) except that **it creates a directory** along with its parent directories…”

**\*Page 479**

**-------------**

CheckPoint 12.10.1 asks:

“What is wrong about creating a File object using the following statement?

new File(“c:\book\test.dat”);”

The answer online correctly addresses the improper use of the \ character inside a string literal, but shouldn’t it also address the improper practice of using an absolute file name to create the File object? Page 478 states: “Do not use absolute file names in your program. If you use a file name such as c:\\book\\Welcome.java, it will work on Windows but not on other platforms.”

**\*Page 481**

**-------------**

In Listing 12.4, line 6 has the code: System.exit(**0**); This code is executed if the user attempts to create a File object for a file that already exists. Based on what I have read up to this point in the text, **I was expecting a non-zero exit code because the program terminated abnormally**. Furthermore, Listings 12.13 and 12.16 have similar lines of code in which a non-zero arg is used for System.exit( ).

**Page 483**

**-----------**

In Listing 12.15, the code in line 18 has a stylistic issue. The code is:

firstName **+“** ” + mi + “ ” + lastName + “ ” + score);

Should be:

firstName +**[]**“ ” + mi…

**Page 488**

**------------**

The text states: “The World Wide **web**, abbreviated as WWW, W3, or Web,…”

Perhaps change to: “The World Wide **Web**…”

**\*Page 487**

**-------------**

Towards the top of the page, the text states: “For example, **www.google.com/index.html** is the URL for the file **index.html** located on the Google web server.”

Shouldn’t the URL also contain communications protocol information? If so, then perhaps modify the text to state: “For example**, https://**www.google.com/index.html is the URL for the file…”

**\*Page 488**

**-------------**

This page contains sample console output for Listing 12.17, ReadFileFromURL.java. One of the two samples shows the result of entering the URL **http://**www.yahoo.com into the program. According to the text, the file size of the Yahoo URL is 190006 characters. When I attempted to replicate the program’s results, however, **I got a file size of only 8 characters.**

I had to change the user input from “**http**” to “**https**” in order to get a file size similar to the one in Listing 12.17. At my time of testing, I got a file size of 150,750 characters.

**\*Page 489**

**-------------**

In the rightmost box of Figure 12.11, the text reads “**URL4**” but it should be “**URL34**”.

**\*Page 491**

**-------------**

Towards the top of the page, the text states: The getSubURLs(**url**) method…” two different times. I believe the text should be modified to: getSubURLs(**urlString**)

**Page 492**

**-----------**

Number 13 in the Chapter Summary section has a typo. The text states:

“Exception handling should not be used to replace simple tests. You should perform simple **test** using if statements…”

Perhaps change to:

“You should perform simple **tests** using if statements…”

OR

“You should perform **a** simple test using if statements…”

**Chapter 13: Abstract Classes and Interfaces**

**\*Page 501**

**-------------**

The UML graphics for the GeometricObject, Circle and Rectangle classes do not have the keyword “String” capitalized in certain sections. For example, the overloaded GeometricObject constructor is written as:

#GeometricObject(color: **string**, filled: boolean)

I only bring this up because String is capitalized in almost all similar occasions.

**Page 502**

**-----------**

Towards the middle of the page, the text states:

The implementation of Circle and Rectangle is the same as in Listings 11.2 and 11.3, except they extend the GeometricObject class defined in this chapter. You can see the complete code for these two **programs** at…”

Perhaps replace the word “**programs**” with the word “**classes**”? I do not know enough to be able to say without a doubt that “**programs**” should be changed, but after a quick online search it seems to me that a Java class must have a main method in order for it to be considered a program.

**Page 503**

**-----------**

Towards the bottom of the page, the text states:

“The JVM dynamically determines which of these methods to invoke at runtime, depending on the **actual object** that invokes the method.”

Perhaps change to:

“The JVM dynamically determines which of these methods to invoke at runtime, depending on the **actual type of the object** that invokes the method.”

**Page 504**

**------------**

Towards the middle of the page, one of the key points on the left column states:

“**concrete method overridden to be abstract**”

This key point corresponds to the square bullet point that states:

“A subclass can be abstract even if its superclass is concrete. For example, the Object class is concrete, but its subclasses, such as GeometricObject, may be abstract.”

Shouldn’t this particular key point should instead state something such as:

“**concrete superclass with abstract subclass(es)**”

**\*Page 507**

**-------------**

CheckPoint 13.3.2 asks:

Why do the following two lines of code compile but cause a runtime error?

**Number[] numberArray = Integer[2];**

numberArray[0] = Double.valueOf(1.5);

The first statement is missing the **new** operator. Should be:

Number[] numberArray = **new** Integer[2];

**Page 508**

**-----------**

The UML graphic for the java.util.Calendar class has what appears to be an insignificant typo. The description for the +set(field: int, value: int) states:

“Sets the **given calendar to** the specified value.”

For clarity, perhaps the description should be modified to:

“Sets the **given calendar field to** the specified value.”

**\*Page 514**

**-------------**

Towards the middle of the page, there is a typo in the code box for the BigInteger class.

implements Comparable<**Biginteger**> should be Comparable<**BigInteger**>

**\*Page 516**

**-------------**

In Listing 13.10, line 10 contains the code: System.out.print(rectangle **+ “ ”**);

Line 11 contains the code: System.out.println( );

This code prints the String returned by the toString( ) method uing a foreach loop.

Perhaps remove the unnecessary space displayed on the console? As in:

Line 10: System.out.print(rectangle);

Line 11: System.out.println( );

**\*Note:** Dr. Liang’s response was that he would simply change the code to:

Line11: System.out.println(rectangle);

Dr. Liang’s solution is more consice. If I remember correctly, I was going to suggest this as well but did not want to modify the code in the text too much in case Dr. Liang had two print statements for a reason.

**Page 520**

**-----------**

The text states:

“ The header for the clone method defined in the Object class is

protected native Object clone( ) throws CloneNotSupportedException**;** ”

**Perhaps remove the semi-colon** at the right-end of the method header?

**\*Page 521**

**-------------**

Figure 13.6 contains a graphic illustration of a House object being cloned. Perhaps it is not meant to be entirely formal or accurate, but I will point out the following issue just in case. The text in the graphic states:

house2 = **house1.clone( );** this is typed twice, perhaps should be:

house2 = **(House) house1.clone( );**

also, perhaps capitalize the “**d**” in “**date** object contents”

**\*Page 523**

**-------------**

CheckPoint 13.7.6 has two typos that prevent the code from compiling. The code is:

**1.** House house1 = new House(1, 1750, **50**);

The House class, as defined in Listing 13.11, does not contain a constructor that takes three args. I believe the code should be:

House house1 = new House(1, **1750.50**); //id and area parameters

Online, the answer is listed as “false” because the statement: System.out.println(house1.equals(house2)**)**; // returns false

**2.** The second typo is in the println statement. The text is missing the closing parenthesis that I colored red above.

**Chapter 17: Binary I/O**

**\*Page 695 - 696**

**---------------------**

**1.** The bottom half of this page contains an UML graphic for the java.io.OutputStream class. The first method is typed as: *+write(****int b****): void*

Perhaps should be: *+write(****b: int****): void*

**2.** The Note at the middle of page 695 states:

“**All the methods** in the binary I/O classes are declared to throw java.io.IOException or a subclass of java.io.IOException.”

At the middle of page 696, the text states:

“**Almost all** the methods in the I/O classes throw java.io.IOException.”

**3.** In the bottom half of page 696, code-box (a) declares to throw an IOException in the main method header. The text states:

public static void main(String[ ] args) throws ***IOException*** {

In the text, “IOException” is italicized. It might have been italicized intentionally in order to emphasize the IOException, but perhaps it was done inadvertently.

**Page 697**

**-----------**

The text has a few typos in the top half of this page. The text states:

“The java.io.InputStream and java.io.OutputStream classes implement the **AutoClosable** interface. The **AutoClosable** interface defines the close( ) method that closes a resource. Any object of the **AutoClosable** type can be used with the try-with-resources syntax…”

The same typo is found in the right column of this page (for a total of four times).

Perhaps change **AutoClosable** to **AutoCloseable**?

Page 698

-----------

This page contains an UML graphic for the java.io.DataInput interface. The graphic shows that the readLine( ) and readUTF( ) methods return “**string**”. The description for the readUTF( ) method states: “Reads a **string** in UTF format.” Perhaps capitalize the ‘s’ in “string” in three different places?

Page 699

-----------

This page contains an UML graphic for the java.io.DataOutput interface. The graphic has the method: +writeUTF(**String s**): void

and explains that the method “Writes s **string** in UTF format.”

Perhaps the UML graphic should be modified to state: +writeUTF(**s: String**): void

and the explanation modified to state: “Writes s **String** in UTF format.”

Page 700

-----------

At the bottom of the page, the text corresponding to Figure 17.11 states:

“DataInputStream filters an input stream of **byte** to data and DataOutputStream converts data into a stream of bytes.”

Perhaps change “**byte**” to “**bytes**”.

\*Page 706

-------------

In Section 17.6, the text states: “Since ObjectInputStream/ObjectOutputStream contains all the **functions** of DataInputStream/DataOutputStream, you can replace…”

Perhaps change “**functions**” to “**methods**”?

**Chapter 18: Recursion**

**\*Page 725**

**-------------**

In Figure 18.4, on the right-side of this page, the text states: 14: **return** fib(0)

I believe that “**return**” should be changed to “**call**”.

**\*Page 728**

**-------------**

Under the Section 18.5 header, the text states: “Sometimes you can find a solution to the original problem by defining a recursive **function** to a problem…”

Perhaps change “**function**” to “**method**”.

**Chapter 19: Generics**

**Page 752**

**-----------**

In Figure 19.1, **perhaps add a semi-colon at the end of the compareTo( )** **method** signature in both (a) and (b).

**Page 753**

**-----------**

Figure 19.3 (a) is an UML representation of the java.util.ArrayList class prior to JDK 1.5. The method: +get(**index:int**) : Object needs a space, as in: +get(**index:[]int**) : Object

\***[]** marks the location where a space is needed.

**Page 756**

**-----------**

Towards the top of the page, the text states: “Instead of using a generic type, you could simply make the **type element** Object, which can accommodate any object type.”

Perhaps change “**type element**” to “**element type**,” might be correct as-is.

**Page 757**

**-----------**

At the top and at the bottom of this page, the main method is typed stylistically different than other parts of the text because it has an extra space between “args” and the closing parenthesis, as in: public static void main(String[ ] args**[]**) {

\***[]** marks the location of the extra space.

**\*Page 763**

**------------**

The right column has two typos: “why <? **Super** T>” and <? **Super** T> type

I believe the ‘**S**’ in “**Super**” should be lowercase.

**Pages 765 - 766**

**--------------------**

**1.** At the bottom of page 755, **Restriction 3** states that: “**A Generic Type Parameter of a Class Is Not Allowed in a Static Context**,” including in an “initializer.” At the top of page 766, the text has the code:

static {

E o2; // Illegal

}

I am **fairly certain that static initialization blocks have not been introduced** up to this point in the text, thus Restriction 3 and the syntax it introduces is a little confusing.

**2.** Towards the middle of the page, the text has the following catch block:

catch (MyException<T> ex) {

. . .

**[]}**

**\*[]** marks the location of the extra space or two

**Perhap push the closing curly brace back a couple of spaces** so that it is flush under the ‘c’ in “catch”.

**Chapter 20: Lists, Stacks, Queues, and Priority Queues**

**Page 776**

**-----------**

**1.** In Section 20.2, the text states: “Maps are efficient data structures for quickly **searching an element** using a key.”

Perhaps change to: “Maps are efficient data structures for quickly searching **for** an element using a key.”

**2.** On the left column, **Priority**Queue is misstyped as “**Prioriry**Queue”.

**Page 778**

**-----------**

In the UML graphic for the java.util.Collection<E> interface, there is no description for the **+equals(o: Object) : boolean** method.

**\*Page 779**

**-------------**

Listing 20.1, TestCollection.java, has an unnecessary space (line 32) and 6 linefeeds that are not accurately displayed in the console output graphic (lines 14, 18, 27, 32, 37, and 42).

**1.** Line 14: System.out.println(“**\n**Is Dallas in collection1? ”

**2.** Line 18: System.out.println(“**\n**” + collection.size( ) +

**3.** Line 27: System.out.println(“**\n**A list of cities in collection2:”);

**4.** Line 32: System.out.println(“**\n**Cities in collection1 or collection2:**[]**”);

**5.** Line 37: System.out.print(“**\n**Cities in collection1 and collection2:”);

**6.** Line 42: Sytem.out.print(“**\n**Cities in collection1, but not in 2: ”);

**The display output graphic should look like:**

A list of cities in collection1:

[New York, Atlanta, Dallas, Madison]

Is Dallas in collection1? true

3 cities are in collection1 now

A list of citie in collection2:

[Seattle, Portland, Los Angeles, Atlanta]

Cities in collection1 or collection2:

[New York, Atlanta, Madison, Seattle, Portland, Los Angeles, Atlanta]

Cities in collection1 and collection2: [Atlanta]

Cities in collection1, but not in 2: [New York, Madison]