the final will focus on the new material - exceptions

Exceptions

Chapter 10

5TH EDITION

Lewis & Loftus

java

Software Solutions

Foundations of Program Design

Exceptions

- Exception handling is an important aspect of object-oriented design
- Chapter 10 focuses on:
  - the purpose of exceptions
  - exception messages
  - the try-catch statement
  - propagating exceptions
  - the exception class hierarchy
  - GUI mnemonics and tool tips
  - more GUI components and containers

Outline

- ➡ Exception Handling
- The try-catch Statement
- Exception Classes
- I/O Exceptions
- Tool Tips and Mnemonics
- Combo Boxes
- Scroll Panes and Split Panes

nogui stuff

Exceptions

- **An *exception* is an object that describes an unusual or erroneous situation**
- **Exceptions are *thrown* by a program, and may be *caught* and *handled* by another part of the program** for any program that has error, then the main throws IOExeption: this will let the compiler know that you know there is an issue, so it will compile it
- **A program can be separated into a normal execution flow and an *exception execution flow***
- **An *error* is also represented as an object in Java, but usually represents a unrecoverable situation and should not be caught**

Exception Handling

- Java has a predefined set of exceptions and errors that can occur during execution
- A program can deal with an exception in one of three ways:
  - ignore it
  - handle it where it occurs
  - handle it an another place in the program
- The manner in which an exception is processed is an important design consideration

Exception Handling

- If an exception is ignored by the program, the program will terminate abnormally and produce an appropriate message
- The message includes a *call stack trace* that:
  - indicates the line on which the exception occurred
  - shows the method call trail that lead to the attempted execution of the offending line
- See `Zero.java` (page 535)

Outline

The try Statement

- **To handle an exception in a program, the line that throws the exception is executed within a *try block***
- **A try block is followed by one or more *catch* clauses**
- **Each catch clause has an associated exception type and is called an *exception handler***
- **When an exception occurs, processing continues at the first catch clause that matches the exception type**
- **See `ProductCodes.java` (page 538)**

The finally Clause

- A try statement can have an optional clause following the catch clauses, designated by the reserved word `finally`
- The statements in the finally clause always are executed
- If no exception is generated, the statements in the finally clause are executed after the statements in the try block complete
- If an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause complete

Exception Propagation

functions are stacked - if there is an issue, it goes back to where it was called. if it is not handled there, then it will go back up another level to see if it is handled

- **An exception can be handled at a higher level if it is not appropriate to handle it where it occurs**
- **Exceptions *propagate* up through the method calling hierarchy until they are caught and handled or until they reach the level of the `main` method**
- **A try block that contains a call to a method in which an exception is thrown can be used to catch that exception**
- **See `Propagation.java` (page 541)**
- **See `ExceptionScope.java` (page 542)**

Outline

```
public void level1()
 {
   System.out.println("Level 1 beginning.");

   try
   {
     level2();
   }
   catch (ArithmeticException problem)
   {
     System.out.println ();
     System.out.println ("The exception message is: " +
                         problem.getMessage());
     System.out.println ();
     System.out.println ("The call stack trace:");
     problem.printStackTrace();
     System.out.println ();
   }

   System.out.println("Level 1 ending.");
 }
```

Exception Handling

The try-catch Statement

Exception Classes

I/O Exceptions

Tool Tips and Mnemonics

Combo Boxes

Scroll Panes and Split Panes

The Exception Class Hierarchy

- **Classes that define exceptions are related by inheritance, forming an exception class hierarchy**
- **All error and exception classes are descendents of the `Throwable` class**
- **A programmer can define an exception by extending the `Exception` class or one of its descendants**
- **The parent class used depends on how the new exception will be used**

Checked Exceptions

- **An exception is either *checked* or *unchecked***
- **A *checked exception* either must be caught by a method, or must be listed in the *throws clause* of any method that may throw or propagate it**
- **A throws clause is appended to the method header**
- **The compiler will issue an error if a checked exception is not caught or asserted in a throws clause**

Unchecked Exceptions

- **An unchecked exception does not require explicit handling, though it could be processed that way**
- **The only unchecked exceptions in Java are objects of type `RuntimeException` or any of its descendants**
- **Errors are similar to `RuntimeException` and its descendants in that:**
  - **Errors should not be caught**
  - **Errors do not require a throws clause**

The throw Statement

can create own exepetions - use throw instead of throws

- **Exceptions are thrown using the *throw* statement**
- **Usually a throw statement is executed inside an if statement that evaluates a condition to see if the exception should be thrown**
- **See `CreatingExceptions.java` (page 545)**
- **See `OutOfRangeException.java` (page 546)**

Outline

Exception Handling

The try-catch Statement

Exception Classes

I/O Exceptions

Tool Tips and Mnemonics

Combo Boxes

Scroll Panes and Split Panes

I/O Exceptions

- **Let's examine issues related to exceptions and I/O**
- **A *stream* is a sequence of bytes that flow from a source to a destination**
- **In a program, we read information from an input stream and write information to an output stream**
- **A program can manage multiple streams simultaneously**

Standard I/O

- **There are three standard I/O streams:**
  - ***standard output* – defined by `System.out`**
  - ***standard input* – defined by `System.in`**
  - ***standard error* – defined by `System.err`**
- **We use `System.out` when we execute `println` statements**
- **`System.out` and `System.err` typically represent a particular window on the monitor screen**
- **`System.in` typically represents keyboard input, which we've used many times with `Scanner` objects**

The IOException Class

- **Operations performed by some I/O classes may throw an `IOException`**
  - **A file might not exist**
  - **Even if the file exists, a program may not be able to find it**
  - **The file might not contain the kind of data we expect**
- **An `IOException` is a checked exception**

Writing Text Files

- In Chapter 5 we explored the use of the `Scanner` class to read input from a text file
- Let's now examine other classes that let us write data to a text file
- The `FileWriter` class represents a text output file, but with minimal support for manipulating data
- Therefore, we also rely on `PrintStream` objects, which have `print` and `println` methods defined for them

Writing Text Files

- **Finally, we'll also use the `PrintWriter` class for advanced internationalization and error checking**
- **We build the class that represents the output file by combining these classes appropriately**
- **See `TestData.java` (page 549)**
- **Output streams should be closed explicitly**

filewriter
bufferwritter - collects all the files, instead of opening each file one by one
printwritter

Outline

- Exception Handling
- The try-catch Statement
- Exception Classes
- I/O Exceptions
- ➡ Tool Tips and Mnemonics
- Combo Boxes
- Scroll Panes and Split Panes

Tool Tips

- A *tool tip* provides a short pop-up description when the mouse cursor rests momentarily on a component
- A tool tip is assigned using the `setToolTipText` method of a Swing component

```
JButton button = new JButton ("Compute");
button.setToolTipText ("Calculate size");
```

Mnemonics

- A *mnemonic* is a keyboard alternative for pushing a button or selecting a menu option
- The mnemonic character should be chosen from the component's label, and is underlined
- The user activates the component by holding down the ALT key and pressing the mnemonic character
- A mnemonic is established using the `setMnemonic` method:

```
JButton button = new JButton ("Calculate");
button.setMnemonic ("C");
```

Disabled Components

- Components can be *disabled* if they should not be used
- A disabled component is "grayed out" and will not respond to user interaction
- The status is set using the `setEnabled` method:

```
JButton button = new JButton (“Do It”);
button.setEnabled (false);
```

GUI Design

- **The right combination of special features such as tool tips and mnemonics can enhance the usefulness of a GUI**

- **See `LightBulb.java` (page 553)**
- **See `LightBulbPanel.java` (page 554)**
- **See `LightBulbControls.java` (page 556)**

Outline

Exception Handling

The try-catch Statement

Exception Classes

I/O Exceptions

Tool Tips and Mnemonics

➡ Combo Boxes

Scroll Panes and Split Panes

Combo Boxes

- A *combo box* provides a menu from which the user can choose one of several options
- The currently selected option is shown in the combo box
- A combo box shows its options only when the user presses it using the mouse
- Options can be established using an array of strings or using the `addItem` method

The JukeBox Program

- **A combo box generates an action event when the user makes a selection from it**
- **See JukeBox.java (page 559)**
- **See JukeBoxControls.java (page 560)**

Outline

Exception Handling

The try-catch Statement

Exception Classes

I/O Exceptions

Tool Tips and Mnemonics

Combo Boxes

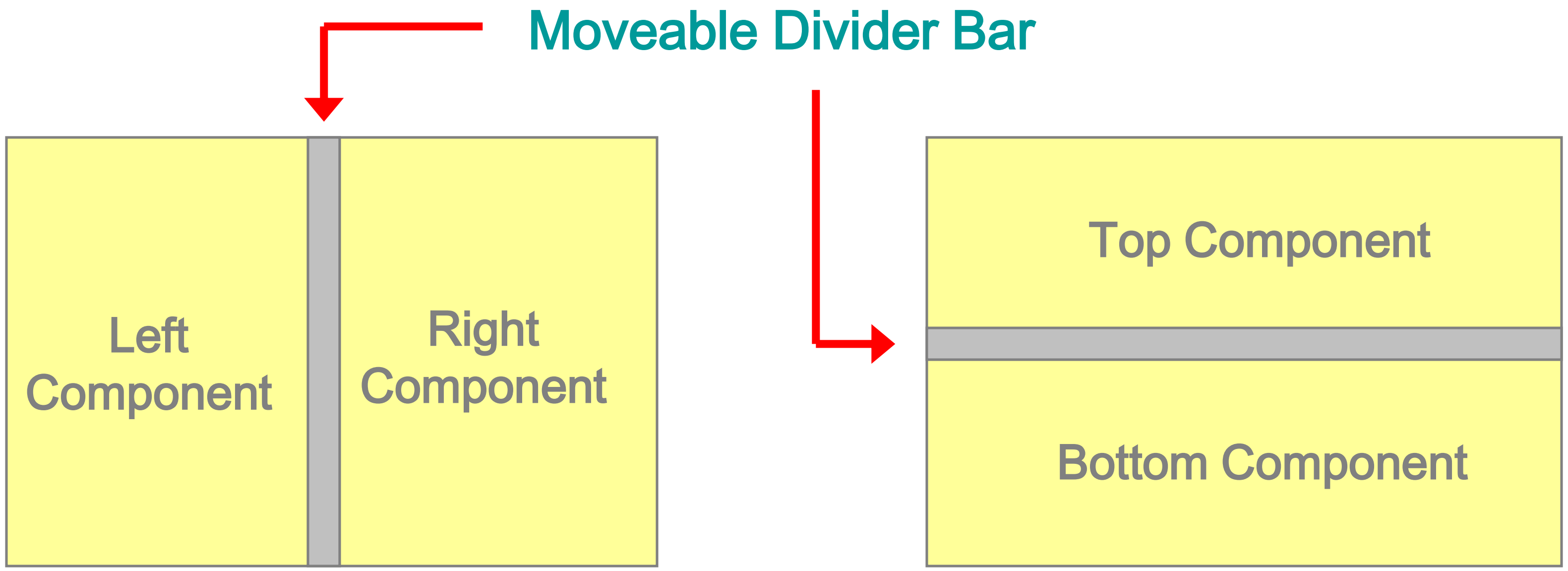
➡ Scroll Panes and Split Panes

Scroll Panes

- A *scroll pane* is useful for images or information too large to fit in a reasonably-sized area
- A scroll pane offers a limited view of the component it contains
- It provides vertical and/or horizontal scroll bars that allow the user to scroll to other areas of the component
- No event listener is needed for a scroll pane
- See `TransitMap.java` (page 564)

Split Panes

- A split pane (`JSplitPane`) is a container that displays two components separated by a moveable divider bar
- The two components can be displayed side by side, or one on top of the other



Split Panes

- **The orientation of the split pane is set using the `HORIZONTAL_SPLIT` or `VERTICAL_SPLIT` constants**
- **The divider bar can be set so that it can be fully expanded with one click of the mouse**
- **The components can be continuously adjusted as the divider bar is moved, or wait until it stops moving**
- **Split panes can be nested**

Lists

- The Swing `Jlist` class represents a list of items from which the user can choose
- The contents of a `JList` object can be specified using an array of objects
- A `JList` object generates a *list selection event* when the current selection changes
- See `PickImage.java` (page 568)
- See `ListPanel.java` (page 570)

Lists

- A `JList` object can be set so that multiple items can be selected at the same time
- The *list selection mode* can be one of three options:
  - single selection – only one item can be selected at a time
  - single interval selection – multiple, contiguous items can be selected at a time
  - multiple interval selection – any combination of items can be selected
- The list selection mode is defined by a `ListSelectionModel` object

Summary

- **Chapter 10 has focused on:**
  - **the purpose of exceptions**
  - **exception messages**
  - **the try-catch statement**
  - **propagating exceptions**
  - **the exception class hierarchy**
  - **GUI mnemonics and tool tips**
  - **more GUI components and containers**