Chapter 12 - Exception Handling and I/O
12.1 (454) - **Introduction** - Exceptions are runtime errors. Exception handling enables a program to deal with runtime errors and continue its normal execution. An exception is an object that represents an error or a condition that prevents execution from proceeding normally
12.2 (454-458) - **Exception Handling Overview** - The advantages of using exception handling: enables a method to throw an exception to its caller, enabling the caller to handle the exception.
12.3 (459-461) - **Exception Types** - Exceptions are object, and objects are defined using classes. The root class for exceptions is java.lang.Throwable. Subclasses of Error and Exception (460), Subclasses of RuntimeException (461). Exceptions occur during the execution of a method (RuntimeException and Error are *unchecked exceptions*). All others are *checked exceptions* (using a try-catch block).
12.4 (462-468) - **More on Exception Handling** - Declaring an exception, throwing an exception, catching an exception. (462 - 463).
Catching blocks need to be specific —> general (example pg. 467)
12.6 (471-471) - **When to Use Exceptions** - A method should throw an exception wit the error needs to be handled by its caller.
logic vs. exceptions (pg. 472 top)
12.11 (480-485) - **File Input and Output** -
Writing Date Using PrintWriter (480-481)
main method throws java.io.IOException
java.io.File file = new java.io.File("Text.txt");
java.io.PrintWriter output = new java.io.PrintWriter(file); output.close()
Reading Data Using Scanner (482-483)
Read from the keyboard:
Scanner input = new Scanner(system.in);
Read from a file:
Scanner input = new Scanner(new File(filename));
The File Class - pg478-479

```
Scanner userScan = new Scanner(System.in);
  System.out.println("Enter filename: ");
  String filename = userScan.nextLine().trim();
  File f = new File(filename);
  String path = f.getPath();
  int dot = path.lastIndexOf(".");
     String extension = path.substring(dot + 1);
     if (extension.equals("txt") == true || extension.equals("dat") == true)

 Scanner fileScan = new Scanner(f);

  while (fileScan.hasNextLine()) {
  String line = fileScan.nextLine();
  String [] tokenArray = line.split(" ");

  //Count words excluding empty lines
  if (line.isEmpty() == false) {
   wordCount = wordCount + tokenArray.length;}
  //Count characters in each word
  for (int i = 0; i < tokenArray.length; i++) {
   String token = tokenArray[i];
   wordLength = wordLength + token.length();}}

  fileScan.close();

  System.out.println("Word Count: " + wordCount + "\n"
+ "Average Word Length: " + (wordLength / wordCount))
  return;

int classValue = parseInt(     .getText())

substring(int startIndex);
substring(int startIndex, int endIndex);

public class IllegalExprException extends Exception{
 public IllegalExprException(String message){
  super(message);}}

catch (IllegalExprException e){
    String message = e.getMessage();
    System.out.println(message);}
```

Chapter 13 - Abstract Classes and Interfaces
13.1 (500) - **Introduction** - A superclass defines common behavior for related subclasses, interfaces are used to define common behavior for classes (including unrelated classes).
13.2 (500-504) - **Abstract Classes** - An abstract class can not be used to create objects. An abstract class can contain abstract methods that are implemented in concrete subclasses.
Largest number in a list (pg. 506)
Calendar and Gregorian Calendar (pg. 508-509)
13.5 (510-513) - **Interfaces** - An interface is a class-like construct for defining common operations for objects.
 - modifier interface InterfaceName{ }
 - you implement an interface

13.6 (513-517) - **The Comparable Interface** - The Comparable interface defines the compareTo method for comparing objects.
 - public interface Comparable<E> { public int compareTo(E o);}
 - public final class String extends Object implements Comparable <String> { @Override public int compareTo(String o) {}}

```
public int compareTo(Employee emp) {
ComparableRectangles [] = {}
  double salary1 = this.getSalary();  java.Arrays.sort(rectangles);
  double salary2 = emp.getSalary();

  if (salary1 < salary2) { return -1;}
  if (salary1 == salary2) {return 0;}
  else { return 1; }}
```

Interfaces vs. Abstract Classes (pg. 523)

Chapter 15 - Event-Driven Programming
15.1 (594-596) - **Introduction**
Button (Event Source Object) —> Event (Event Object) —> Handler (Event handler object).
Writing a handler class & handler event (pg.595 ln 32-35).

15.2 (596-597) - **Events and Event Sources** - An event is an object created from an event source. Firing an event means to create an event and delegate the handler to handle the event.
event-driven programming - running a Java GUI program, the program interacts with the user and the events drive its execution
event - a signal to the program that something has happened
Table for User Action to Event Registration method (pg. 597)

15.3 (597-601) - **Registering Handlers and Handling Events** - A handler is an object that must be registered with an event source object and it must be an instance of and appropriate event-handling interface.
Ex. lengthy handler = handler class (pg.600 ln 43-46)

15.4 (601-602) - **InnerClasses** - An inner class, or nested class is a class defined within the scope of another class. Inner classes are helpful for defining handler classes.
pg. 602 Bullet Points
1. compilation name: OuterClassName$*InnerClassName*.class
2. inner classes can reference the data and methods defined in the outer class in which it nests
3. inner classes can have visibility modifiers
4. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstick members of the outer class.
5. Creating an object for a static or non-static inner class

15.5 (602-605) - **Anonymous Inner-Class Handlers** - An anonymous inner class is an inner class without a name. It combines defining an inner class and creating an instance of the class into one step.
- Inner Class vs. Anonymous Inner Class (pg. 603 top)

15.6 (605-609) - **Lambda Expressions** - Lambda expressions can be used to greatly simplify coding for event handling.
- Anonymous inner class event handler vs. Lambda expression event handler (pg. 605 bottom)
- 4 Ways to Write a lambda expression
1. (ActionEvent e) —> { circlePane.enlarge(); }
2. (e) —> {circlePane.enlarge();}
3. e —> {circlePane.enlarge();} (only one parameter without an explicit data type)
4. e —> circlePane.enlarge() (only one statement)

SAM interface, functional interface, function object, lambda function, functional programming. (pg. 607 top)
Creating your own actions (pg. 608)

15.8 (611-612) - **Mouse Events**
ex. setOnMouseDragged (597)
e.getButton() == MouseButton.PRIMARY
15.9 (613-616) - **Key Events**
Key Code Constants (pg. 613 bottom)
getCode() = returns the key code associated with the key in this event
e.getCode() == KeyCode.DOWN
15.10 (616-618) - **Listeners for Observable Objects**
ex. 617 top & 618 w/ method call
15.11 (618-626) - **Animation**
pause(), play(), stop()
setcycleCount(Timeline.INDEFINITE);
- **Timeline** (pg. 624), Flashing text (pg. 624-625), Clock animation (pg. 625-626)
Timeline animation = new Timeline(newKeyFrame(Duration.millis(500), eventHandler));

Chapter 16 - JavaFX UI Controls and Multimedia
16.1 (644) - **Introduction**

16.2 (644-646) - **Labeled and Label** - A label is a display area for a short text, a node, or both.
Label (), Label("Text"), Label("Text", node)

16.3 (646-648) - **Button** - A button is a control that triggers an action event when clicked. Button(), Button ("Text"), Button("Text", node), bt.setOnAction(e—> action);

16.4 (648-650) - **CheckBox** - A CheckBox is used for the user to make a selection. CheckBox(), CheckBox("Text").
To check if a check box is selected, use the isSelected() method.

16.5 (651-653) - **RadioButton** - Radio buttons enable you to choose a single item from a group of choices.
ToggleButton(), ToggleButton("Text"), ToggleButton("Text", node)
RadioButton(), RadioButton("Text")
ToggleGroup group = newToggleGroup;
button1.setToggleGroup(group); (Without grouping radio buttons would be independent)
Use the isSelected() to see if the radio button is selected.
(Event Handling for radio buttons pg. 653)

16.6 (654-655) - **TextField** - A text field can be used to enter or display a string.
TextField(), TextField("Text"), tf.setOnAction(e—> text.setText(tf.getText())).
PasswordField hides input text.

16.7 (655-658) - **TextArea** - enables the user to enter multiples lines of text.
TextArea(), TextArea("Text"), DescriptionPane()
16.8 (659-662) - **ComboBox** - A list of items to choose.
ComboBox(), ComboBox(items: ObservableList<String>)… items = FXCollections.observableArrayList(flagTitles);

16.9 (662-664) - **ListView** - Same as ComboBox but enables the user to select a single or multiple values. ListView(), ListView(items: ObservableList<T>)
lv.getSelectionModel().setSelectionMode(SelectionMode.Multiple);
Listenting to item selected (pg. 664 ln44)

16.10 (665-667) - **ScrollBar**
ScrollBar(), increment(), decrement()
Setting new location for item in center pane is done by adding a Listener (pg.667 ln30-37 & below)

16.11 (668-670) - **Slider**
Slider(), Slider(min, max, value)
Setting new location for item in center pane is done by adding a Listener (pg. 669 ln35-41 & below)

Chapter 14 - JavaFx Basics
14.1 (542) - **Introduction -** presents the basics of JavaFX programming and uses JavaFx to demonstrate object-oriented designed programming

14.3 (542-543) -**The Basic Structure of a JavaFX Program -** the javafx.application.Application class defines the essential framework for writing JavaFX programs.
- basics of a JavaFX application (pg. 543) - includes launching and stage set up
14.4 (545-547) - **Panes, Groups, UI Controls, and Shapes**
- diagram that shows the relationship between classes 14.3 (pg. 545)
14.5 (548-551) - **Property Binding**
- binding the center of a circle to hold the height and width of a pane (pg. 549 ln16-17)
- another example of binding, DoubleProperty to another double property (pg. 550)
14.7 (553) - **The Color Class**
- chart that shows all the methods for the Color Class (pg. 553)
14.8 (554-555) **The Font Class**
Label label = new Label("JavaFX");
label.setFont(Font.font("Times New Roman", FontWeight.BOLD, FontPosture.ITALIC, 20))
14.10 (558-567) **Layout Panes and Groups, Flow Pane** - 559, **Grid Pane** - 561, **Border Pane** - 563, **HBox** and **VBox** - 564 - 566
14.11 (567-579) **Shapes, Text -** 567-569, **Line -** 569-570, **Rectangle -** 570-572, **Circle -** 572-573