CP213 Lesson 12

* When a window is interacted with and you want to keep track of it you can use a WindowListener

Ex.

addWindowListener(new CheckOnExit());

* The CheckOnExit shows a confirm box asking if the user is sure
* This would be placed as an inner class in a JFrame
* Whether or not a window listener is registered to respond to window events, a *setDefaultCloseOperation* invocation should still be included. This invocation is usually made in the JFrame constructor.
* If the window listener takes care of closing, resizing minimizing and other behaviours, the JFrame constructor should contain the following:

setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE)

* If it is not included the default close operation for JFrame is to HIDE\_ON\_CLOSE which will not end the program.
* The dispose method:
  + Used to unload the resources being used to run the JFrame
  + Will not end the program, useful if the program has multiple windows open

**Icons**

* Icons are small pictures that are based on digital image files like .gif, .jpg or .tiff
* The class ImageIcon is used to convert a picture file to a swing icon
* An icon can be given as an argument to a JLabel constructor
* Text can be added to an icon as well using the setText method

**The Insets class**

* Used to specify the size of the margin in a button or menu item

**Scroll bars**

* When a text area is created, the number of visible lines and the number of characters per line are specified as follows:

JTextArea memoDisplay = new JTextArea(15, 30);

* Instead of setting a firm maximum, scroll bars can be used
* Scroll bars can be added using the JScrollPane class
* When scroll bars are used text is viewed through a **viewport** which displays a portion of the text to the user
* The JScrollPane class takes a JTextArea object as a parameter so it knows what text to display

JtextArea textArea = new JTextArea(LINES, CHAR\_PER\_LINE);

textArea.setBackground(Color.WHITE);

JScrollPane scrolledText = new JScrollPane(textArea);

* The scroll bar policies can be set as follows

scrolledText.setHorizontalScrollBarPolicy(

JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

scrolledText.setVerticalScrollBarPolicy(

JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);

* The JScrollPane can then be added to a container such as a JPanel or a JFrame

**Visibility**

* You can change the visibility of an object using the setVisible method and giving a true or false argument

**The Paint method and Graphics class**

* Paint draws the component or container on the screen. It is already defined and is called automatically when the figure is displayed on the screen
* The paint method has to be redefined to draw geometric figures like circles and boxes, when redefining always include:

super.paint(g);

An example implementation:

public void paint(Graphics g) {

super.paint(g);

g.drawOval(X\_position, Y\_position, pic\_DIAMETER, pic\_DIAMETER);

}

* The parameter g is of type graphics, when the paint method is called, g is replaced by the graphics object associated with the JFrame. Therefore, the figures are drawn inside the JFrame.
* Calling the repaint() method will recolour the screen.

**Drawing objects in Java**

* You can draw rectangles, if you draw other shapes java will put them in an imaginary rectangle
* The coordinate system has (0,0) is the top left corner, x increases rightward and y increases downward.
* An oval is drawn with the drawOval method, it takes four arguments: 2 for the location then one each for width and height
* To draw a circle, make width=height
* You can draw arcs by drawing an oval and then specifying the portion of that oval that is made visible using the drawArc method
* The setColor method takes a Color.COLOUR parameter to change the colour that you are drawing with
* You can define new colours by doing the following:

Color brown = new Color(200, 150, 0);

* You can also have a user select a colour using JColorChooser which shows a colour choosing dialog

**Resizing the window**

* The pack() method resizes the window to the preferred size
* An invocation of the validate method causes a container to layout its components again
* Every container class has the validate method, which takes no arguments

**Fonts**

* You can set the font using g.setFont(fontObject) within the paint method
* Any font currently available on a system can be used in Java however they guarantee that at least three fonts will be available: Monospaces, SansSerif and Serif
* Font size is called its *point size*

**Java Threads and Multithreading programs**

* Two basic units of execution: processes and threads (java is mostly concerned with threads)
* Processing time for a single core is shared among processes and threads through an operating system feature called time slicing
* Processes:
  + A process has a self-contained execution environment
  + Each process has its own memory space
  + Most implementations on the java virtual machine run as a single process
  + Often seen as synonymous with programs or applications
* Threads:
  + Sometimes called lightweight processes
  + Creating a new thread requires fewer resources than creating a new process
  + Threads exist within a process, and they share the same memory
  + The thread.sleep method causes that particular thread to sleep for a specified number of milliseconds
  + In the example of the nonresponsive GUI, the window would not close until all the circles were drawn. This is because they were all running on one thread, to fix this you need to have the actionPerformed method to create a new thread to draw the circles.
  + Usually, a derived class of Thread is used to program a thread. The methods run and start are inherited from Thread
  + The derived class override the method run to program the thread
  + The method start initiates the thread processing and invokes the run method

public class MyThread extends Thread{

public static void main (String args []) {

MyThread mt = new MyThread () ;

mt.start();

}

@Override

public void run () {

// your code goes here

}

}

**The runnable interface**

* Another way to create a thread is to use the runnable interface’

Creating a thread with the Runnable interface:



**Race conditions**

* When multiple threads have access to the same variable, it is possible that the variable will end up with the wrong (and often unpredictable) value this is called a **race condition** because the final value depends on the sequence in which the threads access the shared value.
* To fix this you can use thread synchronization by putting the keyword *synchronized* in between the access modifier and return type in the method definition

public synchronized void increment() {

int local;

local = counter;

local++;

counter = local;

}

* Instead of putting it around the whole method you can also do this (which is more efficient):

public void increment() {

int local;

synchronized (this) {

local = counter;

local++;

counter = local;

}

}

**JavaFx**

* The JavaFx scene builder editor is a program released by Oracle to graphically construct user interfaces and generate JavaFx code
* JavaFx applications are split into three separate files:
  + FXML files, which are XML files created by the scene builder that describes the layout of the application
  + JavaFx source coded that contain the start method and load the FXML file
  + Controller files that contain a class with event handles that respond to UI controls
* All of the main windows of a JavaFx application extend the Application class (instead of a JFrame)
* The init() methods is automatically called by the application to initialize things before the application starts, it can be used like a constructor
* When the application terminates, the stop() method is invoked. If you want to quit the application prematurely, you can do so using the Platform.exit()
* Steps to create a JavaFx application:

1. Create a new class to represent your application
2. Have this class extend Application
3. The start() method is automatically called from the launch() method to initialize and show the window. You can set the size and the title of the window from within the start method

* The start method takes a stage parameter
* To handle an event, use the handle() method

**Animation in JavaFx**

* To animate an object, you must take these steps:

1. Create an object
2. Instantiate the transition (animation) class that is to be applied
3. Set the properties of the transition
4. Play the transition using the play() method of the Animation class

Keywords

* AnchorPane
* BoarderPane
* Dispose
  + The dispose method is used to unload the resources being used to run the JFrame. It will not end the program, so it is useful if the program has multiple windows open
* FillTransition
* Graphic
* JavaFX
  + A set of packages that allows programmers to create graphical applications using java. It is expected to overtake swing at some point as the standard library for graphical interfaces. It uses metaphors like stages and scenes to create GUIs.
* Pack method
  + The pack method automatically resizes the window to the preferred size. This size will be the perfect size to house all elements within the JFrame.
* paint
* paintComponent
* Pane
  + The JavaFx version of JPanel. Used inside a stage to position objects properly.
* Race
  + When multiple threads share the same variable and are running at the same time, the resulting value of the variable will be determined based on the order that the threads finish in. This is called a race condition.
* repaint
* Run method
* Scene
* setDefaultCloseOperation
  + By default, if a user clicks the exit button on a JFrame, the frame will just hide itself instead of closing the program. The programmer can specify what should happen when that button is clicked using the setDefaultCloseOperation method.
* Stage
* start method
* Synchronize
* Thread
* Thread.sleep
* Validate
* WindowListner