Week 12

Program 4: due before 10pm on Monday 11/19

Team Project: Food Query and Meal Analysis
(D-Team 30pts) Milestone #1: due before 10pm Nov 16th
(A-Team 20pts) Milestone #2: due before 10pm Nov 26th
(A-Team 100pts) Milestone #3: due before 10pm Dec 12th

Read: Module 12 (links to Java 8 feature documentation)

THIS WEEK:
- make and Makefiles examples
- JavaFX class Main
- Design a Scene
- EventHandler
- Functional Interfaces
    - java.util.function.Function
    - java.util.Comprator
- Anonymous Inner Classes
- Lambda expressions

Next Week
- Java Streams
    - java.nio.file.Files
    - java.nio.file.Paths
    - java.nio.Streams

Makefile Examples

what is a makefile?

what a build utility?

Steps

p1 Makefile (simple targets)

```
make:
        javac *.java

test:
        java TestProg

clean:
        \rm *.class
```

p2 Makefile (basics with JUnit)

```
main:
        javac -g -cp .:/usr/share/java/junit4.jar *.java

junit:
        java -cp .:/usr/share/java/junit4.jar org.junit.runner.JUnitCore TestTree

clean:
        \rm *.class
```

p3 Makefile with command-line argument and output to a file

```
all:
    javac -cp . *.java
    java -cp . AnalysisProg data.txt > results.txt

xlint:
        javac -Xlint:unchecked *.java

clean:
    \rm *.class
```

p2 Makefile with variables and Java classpath example

```
# From: https://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html

JFLAGS = -g
JC = javac
CP = .:/usr/share/java/junit4.jar

.SUFFIXES: .java .class

#
# compiles .java source into .class files
#
.java.class:
       $(JC) $(JFLAGS) -cp $(CP) $*.java

CLASSES = \
      SearchTreeADT.java \
      BalancedSearchTree.java \
      TestSearchTree.java \

default: classes

classes: $(CLASSES:.java=.class)

#
# make clean - removes all class files and permit new build of all
#
clean:
      $(RM) *.class

#
# make junit -  run the TestSearchTree JUnit test class
#
junit:
      java -cp $(CP) org.junit.runner.JUnitCore TestSearchTree


#
# make copy - use scp to copy p2 files from assignment directory to current directory
#
# CAUTION: this will overwrite existing files with same name in same directory
#          without warning or chance to cancel copy
#
# TO USE:
#          remove echo from start of scp command
#          replace cslogin with your CS login username
#
copy:
      echo scp -r cslogin@best-linux.cs.wisc.edu:/p/course/cs400-deppeler/public/html-
s/assignments/p2/files/* .
```

JavaFX: class Main extends Application

File -> New -> Other -> JavaFX Project

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;

public class Main extends Application {


   @Override
   public void start(Stage primaryStage) {

        primaryStage.show();
   }

   public static void main(String[] args) {

        launch(args);
   }
}
```

User Interface (UI) Controls

Use a Label to display non-editable text to the user

```
Label usernameLabel = new Label( "Username: " );
usernameLabel.setFont(Font.font("Arial",FontWeight.BOLD,12))
```

Use a TextField to get input from the user

```
TextField usernameField = new TextField();
usernameField.setPromptText("type username here");
usernameField.setMaxWidth(50);  // pixels
```

Use a button to allow the user to perform some action

```
Button submitButton = new Button("Submit);
Image imgOK = new Image(getClass().getResourceAsStream("ok.png"));
Button okButton = new Button("OK", new ImageView(imgOK);
```

Other UI Controls

TextArea - multiple lines of text input
RadioButton – one option of many. Selecting one deselects another
ToggleButton – button that is either on or off
CheckBox – a tri-state control, checked, unchecked, undefined
ChoiceBox – provides a list of choices or shows selected choice
PasswordField – text field that masks the inputted characters
ScrollBar - bar with increment and decrement buttons
ScrollPane - allows the user to scroll the content through panel
ListView – display a list
TableView – visualize unlimited rows broken into defined number of columns
... more and make your own

Panel Layouts

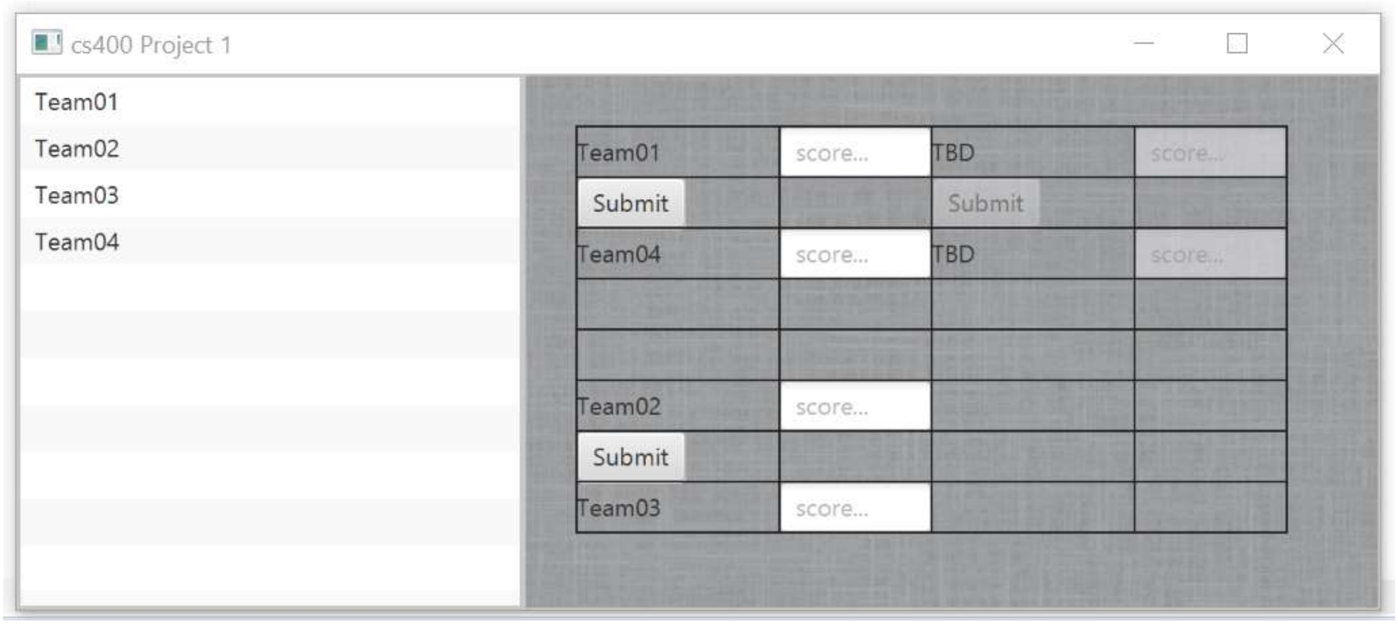
```
// Place controls in desired pane layout
BorderPane borderPanel = new BorderPanel();
.setTop(control_or_pane)
.setLeft(control_or_pane)
.setCenter(control_or_pane)
.setRight(control_or_pane)
.setBottom(control_or_pane)
```

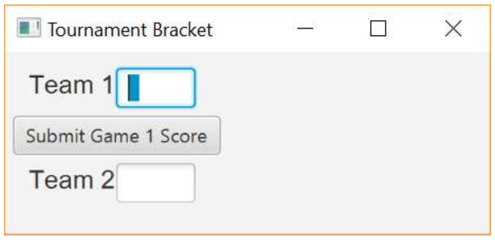
```
HBox hBox = new HBox();
```

```
VBox vBox = new VBox();
```

```
GridPane gPane = new GridPane();
```

Design a Scene





JavaFX: Add Code (EventHandlers) to Controls

User Interface(UI) Controls require EventHandlers
https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

EventListener -

EventHandler -

javafx.event.EventHandler

```
@FunctionalInterface
public interface EventHandler<T extends Event> extends EventListener
{
    public void handle(T event) ;
}
```

Steps to add code to a UI control

1. define a class that implements the EventHandler interface
2. create an instance of that class
3. register that instance as setOnAction for the UI control

```
// Define a new Handler class
class MyHandler implements EventHandler<ActionEvent> {
    Button button;
    MyButtonHandler(Button button) { this.button = button; }
    public void handle(ActionEvent e) {
        if (button.getText().equals("Click Me");)
            button.setText("Don't Click Me");
        else
            button.setText("Click Me");
    }
}
```

```
// somewhere in GUI class methods to create UI button


// Create the handler instance


// register the handler as the handler for the object
```

Java 8: Functional Interfaces

java.util.Comparator

```
public interface Comparator<T>
{
    int compare(E obj1, E obj2) ;
}
```

- **Use a Comparator to compare instances whose types are not Comparable.**

1. define a class that implements Comparator (for your type)
2. create an instance of that class
3. use the instance in a method or expression that requires an instance

Example: Define and use a comparator to sort **Fish**

```
                                                          {
  @Override
             compare(Fish f1, Fish f2) {
          dx = f1.x - f2.x;
          dy = f1.y - f2.y;
          dz = f1.z - f2.z;
             dx==0 ? dy==0 ? dz : dy : dx;
  }
}

// Create and use a FishComparator to sort Fish
List<Fish> fish = getRandomFish(10);



Collections.sort( fish, fishComparator );
```

Anonymous Inner Classes

```
// ANON INNER CLASS - class is anonymous, instance has a name
static final Comparator<Fish> FISH_NAME_COMPARATOR =
```

```
/**
 * Create and use a Comparator – created as an ANONYMOUS INNER CLASS
 *
 * NOTE: FishComparator class is not needed for this example to work
 */
public static void example2_anonymous_inner_class() {
     List<Fish> fish = getRandomFish(10);
     System.out.println("not sorted: " + fish);
     //Collections.sort(fish, ____________________ );
     Collections.sort(fish,

       // PUT INSTANCE OF ANONYMOUS INNER CLASS HERE

     );

     System.out.println("after sort: " + fish);
}
```

Lambda expressions

The purpose of lambda expression is to be able to create an instance of a functional interface, by implementing the single function of the interface as an instance of an anonymous inner class. ie. allow creation of an instance by defining the single function in the lambda syntax form.

Java 8 Lambda Expressions

ValueGenerator example

1. Define `ValueGenerator` as a *functional interface*

2. Define a method that uses a `ValueGenerator` instance

3. Use Lambda Expressions

```
public static void main(String [] args) {
  // what sequence is returned for each instruction?




  System.out.println("seq1="+Arrays.toString(s1));
  System.out.println("seq2="+Arrays.toString(s2));
  System.out.println("seq3="+Arrays.toString(s3));
}
```

Lambda Expression Fill-In-The-Blank (Summary)

- lambda expressions must be associated ______________________________
- functional interface defines ____________________ for a lambda expression
- lambda expression defines ________________________________ for that instance
- convenient (shorter syntax) for when ____________________________ is needed (e.g. EventHandler in JavaFX)
- lambda expressions can be saved as a ________

Practice

Given this functional interface:

```
@FunctionalInterface
interface Hash {
   /** Returns a hash index for the given key.
    * @param key A unique identifier to be added to a hash table.
    * @param TS size of the hash table (ensure hash is valid index)
    * @return the hash index (0 to TS-1)
    */
   int hash(Long key, int TS);
}
```

Write a program (or code fragment) that uses lambda expressions to compare the hash indexes returned for at least three different hash functions for the given array of values.

```
Long[] keys = {-66884678L, 11848818630L, 4962768465676L, 37113781045784766L};

Hash hash1 = (x,y) -> { x = (x ^ ( x >> 32)) % y;
   return (int)(x < 0 ? -x : x); };
```

```
int TS = 11;
for ( Long key : keys ) {
   int h1 = hash1.hash(key,TS);
   int h2 = hash2.hash(key,TS);
   int h3 = hash3.hash(key,TS);
   System.out.format("%30d %4d %4d %4d\n",key,h1,h2,h3);
}
```

```
-16688467836824168 20 04 24 94
118488186306385720 84 94 64
496276846567620 84 104 104
37113781045784766 20 94 34 54
```