Week 12

h10 available soon and due before 10pm on 4/22 (w13)
p4 due before 10pm on Tuesday ~~4/16 (w12)~~ 4/19
p5 due before 10pm on **Friday 4/12** (w11 - install JavaFX and complete in week 11)
p6 due before 10pm on **Friday 5/3** (w14 - start in-class in and complete in week 14)

Team Project: QuizGenerator
milestone 1 design: due before 10 pm on Thursday 4/18 (w12 - our design available on 4/19)
milestone 1 GUI: due before 10pm on on Thursday 4/25 (w13)
milestone 3 final program: due before 10pm on Thursday 5/2 (w14)

Peer Mentors: JavaFX, JSON parsing (p4 and team project), exam review.
Zhiyue (W and F 10-12 in 1289) and Yiye (Th 12:30-2:30pm in 1358, F 9-11 in 1358)

Read: Module 12 (get started on Module 13 reading)

THIS WEEK:
- Linux: review remaining w11 pages, make, scripting, vim
- Teams: GitHub Eclipse JavaFX Team Project Work Flow
- JavaFX
  - class Main structure
  - UI Controls
  - Layout managers
  - Add controls to layout manager
- Design a Scene
- Add functionality to UI Control (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

GitHub Eclipse JavaFX Team Project Work Flow

https://canvas.wisc.edu/courses/143052/pages/git-eclipse-java-fx-team-project-work-flow

Canvas

GitHub Eclipse Java FX Team Project Work Flow

Working together on a team project that requires many files, many cycles of programming effort, and multiple team members can be quite challenging.

Eclipse, JavaFX plug-in, Git, and GitHub can all help, but it does require some setup and configuration effort by all members of your team.

We have a page on Canvas with links to help your team get all members and development pieces working together for the greater good and progress of your programming team.

git setup

1. [Each member] Create your own local Eclipse Java FX project
2. [One member] Create shared repository
    1. Copy starter files to your local Eclipse Java Project
    2. Create shared repository on GitHub
    3. Add starter files to shared repository
    4. Add team members as collaborators
    5. Tell your team members the shared repo is ready for them to clone
3. [Each member] Clone the shared GitHub repository to your local Eclipse Java FX project space
4. [Each member] Contribute to the project. Here is the typical work flow
    1. **git pull**
    2. **EDIT WORK (make sure that it always compiles and runs)**
    3. **git add .**
    4. **git commit –m "what was done"**
    5. **git push**
5. [Each member] Build submission
6. [One member] Submit work
7. [Each member] confirm that correct work is submitted

git remote set-url

git config --global username

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 {
   // declare claim structure here
      // can build static structure and some instances
   @Override
   public void start(Stage primaryStage) {
        List<String> args = this.getParameters().getRaw();
        //call methods to build controls, scene(s)
        //and add code to stage
        primaryStage.show();
   }

   public static void main(String[] args) {
        // can process args
        launch(args);
   }
}
```

] - call

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))
```

" .getText()

" .setText("...") ⇐ static measures

Use a TextField to get input from the user

prompt

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

String username.getText(),

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
ComboBox – 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, tip: use an ObservableList
TableView – visualize unlimited rows broken into defined number of columns
... more and make your own

Layout Managers

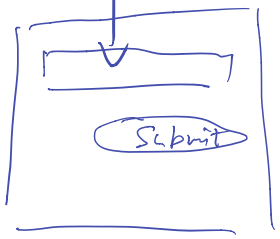
BorderPanel – in last week's demo and p5

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

hBox.getChildren().addAll(usernamelable, usernamefield);


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

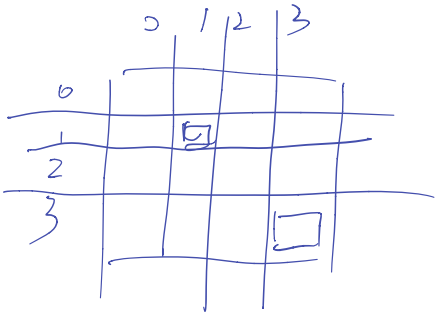
vBox.getChildrenAll(hBox, submitbutton); →




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

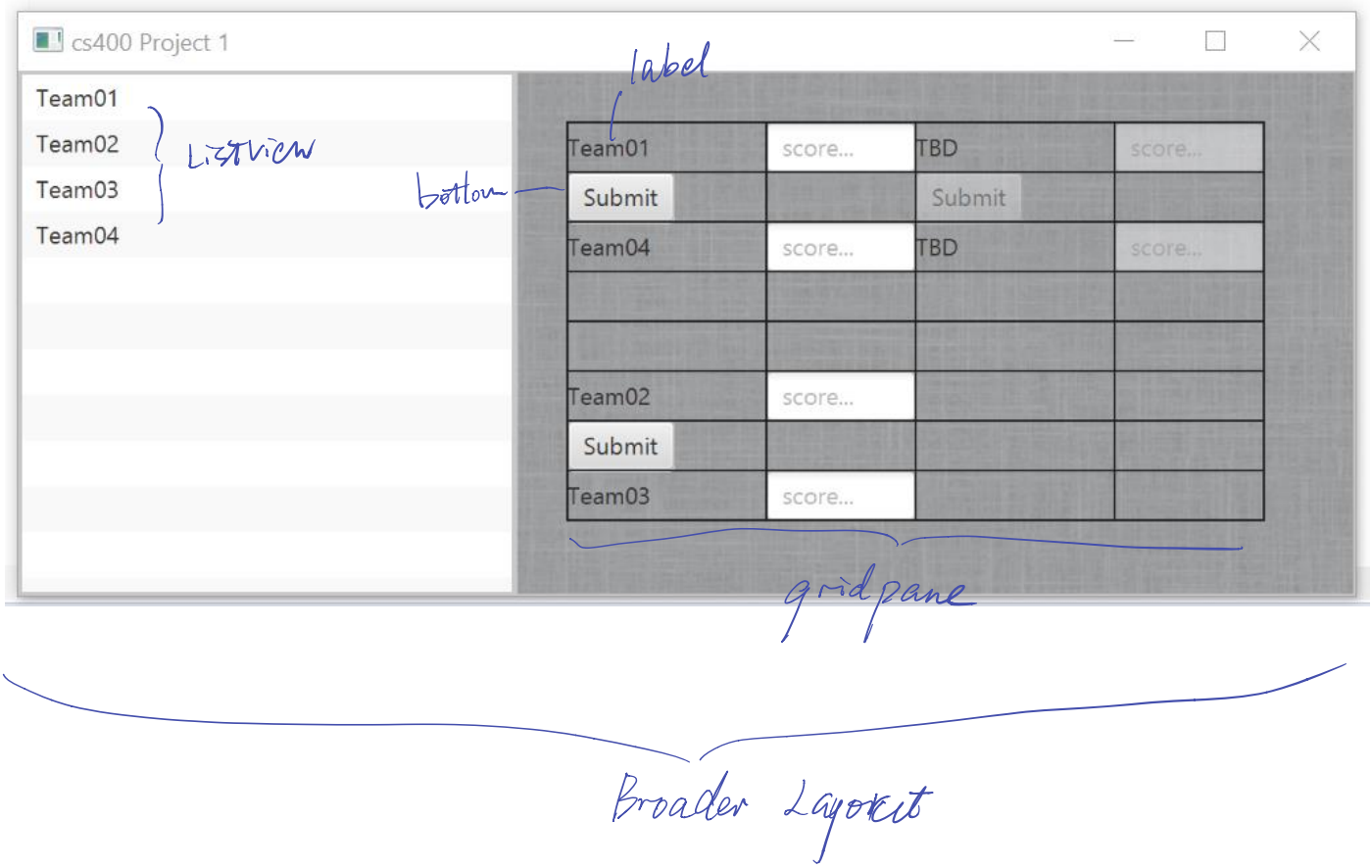
gPane.add(VBox, 3, 3) // col 3 row 3

. ——— - (... 0, 1) // col 0 row 1

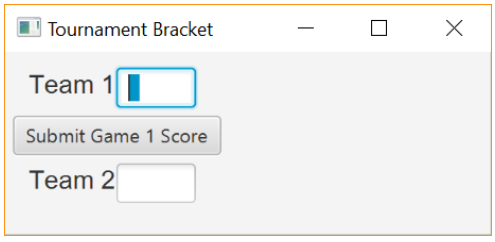


Design a Scene

Label the type of each UI control



Write code to create this scene



JavaFX: Add functionality to UI Controls

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

EventListener – monitors the control and triggers an ActionEvent

EventHandler – excutes codes based on ActionEvent that occurred

In JavaFx, EventHandlers - listen and execute

javafx.event.EventHandler has one method

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

lambda expressions

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;
    MyHandler(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
   Button button = new Button("click me")

// Create the handler instance
   myHandler handeler = new MyHandeler(button);

// register the handler as the handler for the object
    Button.setOnAction(handeler);
```

MyFirstJavaFX – Demo Continued

Add a Button Counter

1. Open p5 "MyFirstJavaFX" program
2. Add a field to track the number of times the button has been click
3. Add a label that shows how many times your button has been clicked
4. Add code to your button to increment the button count when button is clicked
5. Run your program and see if your button counter works

Java 8: Functional Interfaces

- exactly one abstract method
- may have static / default methods
- extends java.util.function.Function
- java.lang.comparable is NOT a functional interface (require object)

similar →

java.util.Comparator functional interface

built in

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

- if object1 < object2 "comes before"
- 0 = =
- \+ > "comes after"

➢ **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**

(by position).

```
public class FishComparator implements Comparator<Fish>
{
    @Override
    public int compare(Fish f1, Fish f2) {
        int dx = f1.x - f2.x;
        int dy = f1.y - f2.y;
        int dz = f1.z - f2.z;
        return dx==0 ? dy==0 ? dz : dy : dx;
    }
}
```

conditional expression; true part; false part

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

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

Anonymous Inner Classes

- unnamed
- defined within another class, method; expression
- same code as before
- can create named instances of anno inner class
- Define, use inplace -- good if only 1 instance method

```
// ANON INNER CLASS - class is anonymous, instance has a name
static final Comparator<Fish> FISH_NAME_COMPARATOR =
    new Comparator<Fish>() {
        @Override
        public int compare(Fish f1, Fish f2) {
            return f1.getName().compareTo(f2.getName());
        }
    };
```

anonymous inner class

↑ name of instance

storing

```
/**
 * 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
        new Comparator<Fish>() {
            @Override
            public int compare(Fish f1, Fish f2) {
                return f1.getSpecies().compareTo(f2.getSpecies());
            }
        }

    );

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

unnamed instances

unnamed instances

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

- used when we need a single unnamed instance AND the type can be inferred by the context.
- Pass behavior, not "data"

ValueGenerator example

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

```
interface ValueGenerator {
    int getNext(int prev);
}
```

2. Define a method that uses a `ValueGenerator` instance (provide a context)

```
static int[] getSequence(ValueGenerator vGen, int v0, int n) {
    int[] sequence = new int[n];
    sequence[0] = v0;
    for (int i = 1; i < sequence.length; i++) {
        sequence[i] = vGen.getNext(sequence[i-1]);
    }
}
```

3. Use Lambda Expressions

```
public static void main(String [] args) {
  // what sequence is returned for each instruction?
   int[] s1 = getSequence(x -> {return x+1;}, 0, 3);

   int[] s2 = getSequence(k -> {return k-1;}, -3, 7);

   int[] s3 = getSequence((0) -> {return (int)(Math.random()*50);}, 0, 5);

  System.out.println("seq1="+Arrays.toString(s1)); [0, 1, 2]
  System.out.println("seq2="+Arrays.toString(s2)); [-3, -4, -5, -6, -7, -8, -9]
  System.out.println("seq3="+Arrays.toString(s3)); [ , , , , ]
}
```

random 范围?

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

- lambda expressions must be associated with a(n) Functional Interface
- functional interface defines one abstract method for a lambda expression
- lambda expression defines the behavior of the function for that instance
- convenient (shorter syntax) for when a single unnamed instance is needed (e.g. EventHandler in JavaFX)
- lambda expressions can be saved as a (in a) variable

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); };
```

may span multiple lines

Hash hash2 = (x, y) -> { ... };

Type create variable

```
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);
}
```

```
-1668846783682416820 04 24 94
118488186306385720 84 94 64
496276846567620 84 104 104
3711378104578476620 94 34 54
```