

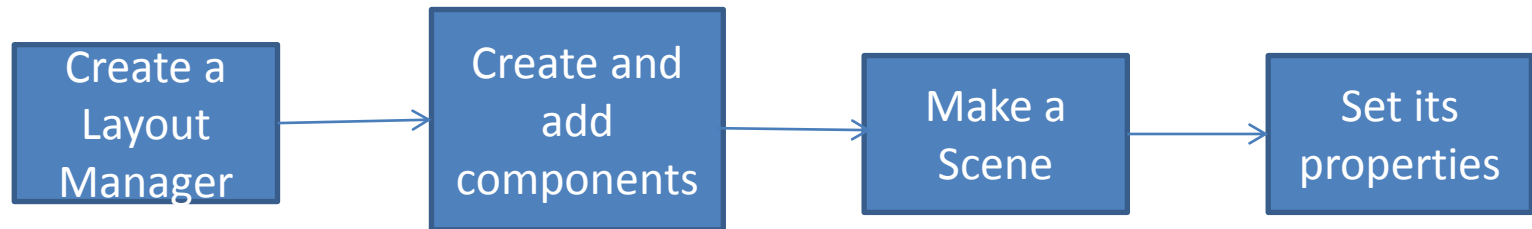
Lec 22

Events

JavaFX - GUIs

- Stage: Top level Java FX Component
- Scene: A container for all the items in the scene
- Border, Hbox, VBox, FlowPane, GridPane: Helper classes that govern where components appear in the Scene

A very rough guide to creating a simple GUI:



You don't always have to do this in this order. Some of the steps can have substeps (e.g., creating a Layout Manager to organize other components)

What happens if “deposit \$20” is clicked

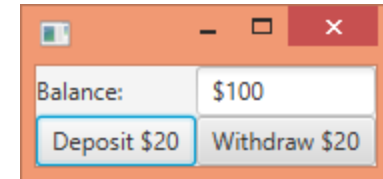
```
private int balance = 100;

public void start(Stage primaryStage) {
    Button deposit = new Button("Deposit $20");
    Button withdraw = new Button("Withdraw $20");
    Label label = new Label("Balance: ");
    TextField balanceField = new TextField ("$" + balance);

    balanceField.setPrefWidth(80);

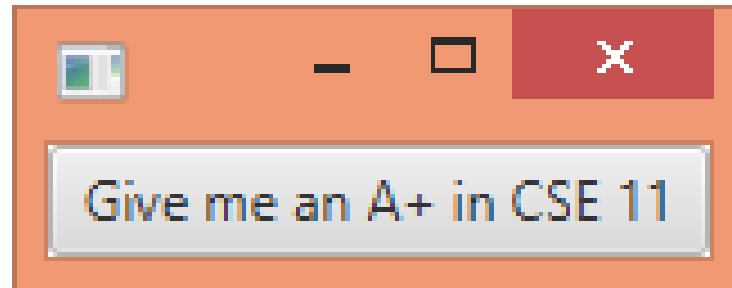
    GridPane pane = new GridPane();
    pane.add(label,0, 0);
    pane.add(balanceField,1, 0);
    pane.add(deposit,0, 1);
    pane.add(withdraw,1, 1);

    Scene scene = new Scene(pane);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```



- A) Balance is increased to 120, but not shown
- B) Program exits
- ➔ C) Nothing
- D) Java prints a message

```
public void start(Stage primaryStage) {  
    Button a = new Button("Give me an A+ in CSE 11");  
    Scene scene = new Scene(a);  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```



Event Handlers

1. Tell the node (e.g. button, text field, etc) to listen
2. Add code to execute when event happens

Events

- <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Button.html>
 - Button click
 - Drag
 - Mouse enters/moves/exits
 - Key pressed/released/typed
 - Scroll

1) Tell node to listen

```
Button b = new Button("ok");  
b.setOnAction(param);  
b.setOnMouseEntered(param);  
b.setOnKeyTyped(param);
```

Where `param` is the code/class that will execute when the event happens.

2) Handle action event

```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

public class Deposit implements EventHandler<ActionEvent>{

    public void handle(ActionEvent e) {
        //your code here
    }
}
```

Events

- <ActionEvent>
- <MouseEvent>
- <DragEvent>
- <TouchEvent>

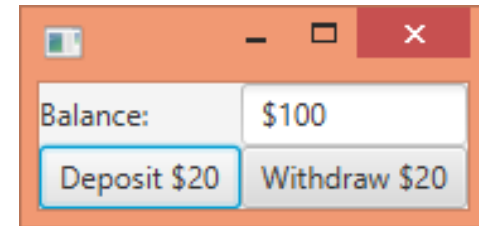
What happens if “deposit \$20” is clicked

```
private int balance = 100;

public void start(Stage primaryStage) {
    Button deposit = new Button("Deposit $20");
    Button withdraw = new Button("Withdraw $20");
    Label label = new Label("Balance: ");
    TextField balanceField = new TextField ("$" +
                                           balance);

    Deposit d = new Deposit();
    deposit.setOnAction(d);

    //same as before to set up GUI
}
```



- A) Balance is increased to 120,
- B) Program exits
- C) Nothing
-  D) Java prints “add \$20”

```
public class Deposit implements EventHandler<ActionEvent>{
    public Deposit() {
        //constructor if needed
    }

    public void handle(ActionEvent e) {
        System.out.println("add $20");
    }
}
```

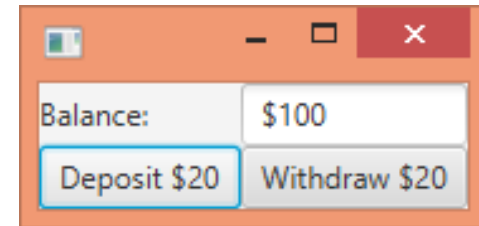
What happens if user enters 200 into the balanceField

```
private int balance = 100;

public void start(Stage primaryStage) {
    Button deposit = new Button("Deposit $20");
    Button withdraw = new Button("Withdraw $20");
    Label label = new Label("Balance: ");
    TextField balanceField = new TextField ("$" +
                                           balance);

    TField tf = new TField();
    balanceField.setOnKeyTyped(tf);

    //same as before to set up GUI
}
```



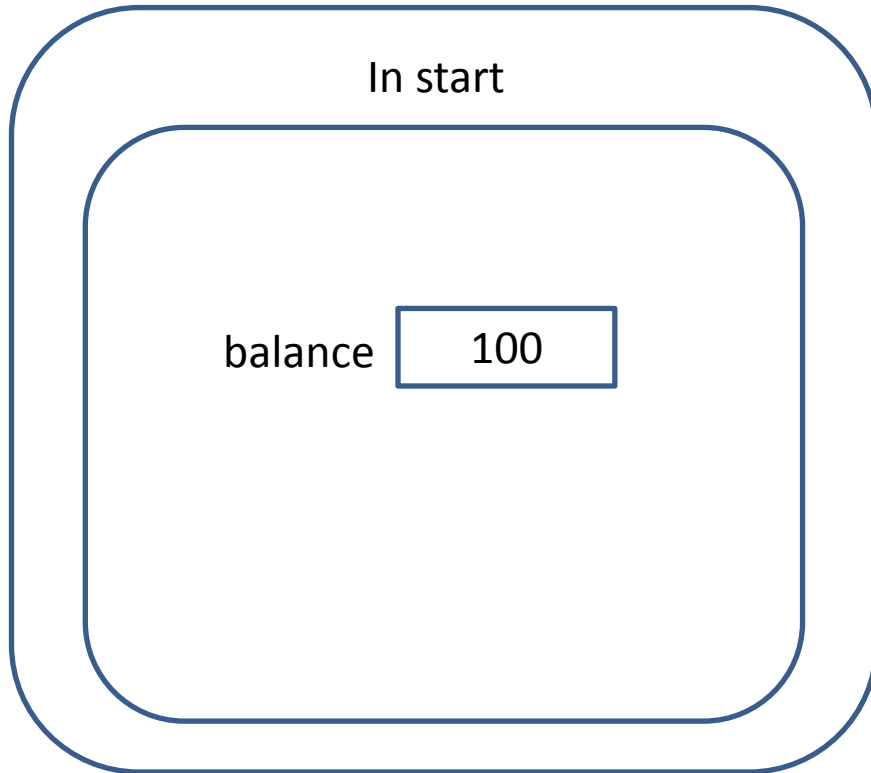
- A) Balance is increased to 200
- B) Program exits
- C) Nothing
- D) Java prints "don't change"
- E) (D) 3 times



```
public class TField implements EventHandler<KeyEvent>{

    public void handle(KeyEvent e) {
        System.out.println("don't change");
    }
}
```

BankAcct object (scope)



Deposit object (scope)



No connection between BankAcct object
and Deposit object!

```

public class BankAcct extends Application {
    private int balance = 100;
    private Button Deposit;
    private TextField balanceField;

    public void start(Stage primaryStage) {
        deposit = new Button("Deposit $20");
        Label label = new Label("Balance: ");
        balanceField = new TextField ("$" + balance);
        Button withdraw = new Button("Withdraw $20");

        Deposit d = new Deposit();
        deposit.setOnAction(d);

        // rest of code
    }

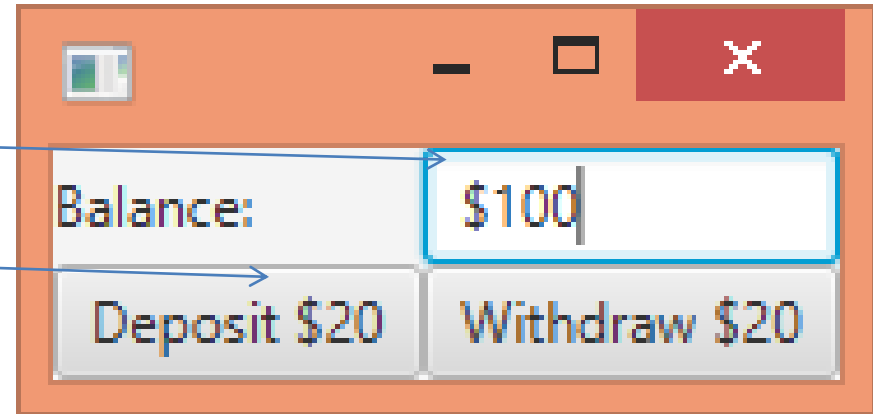
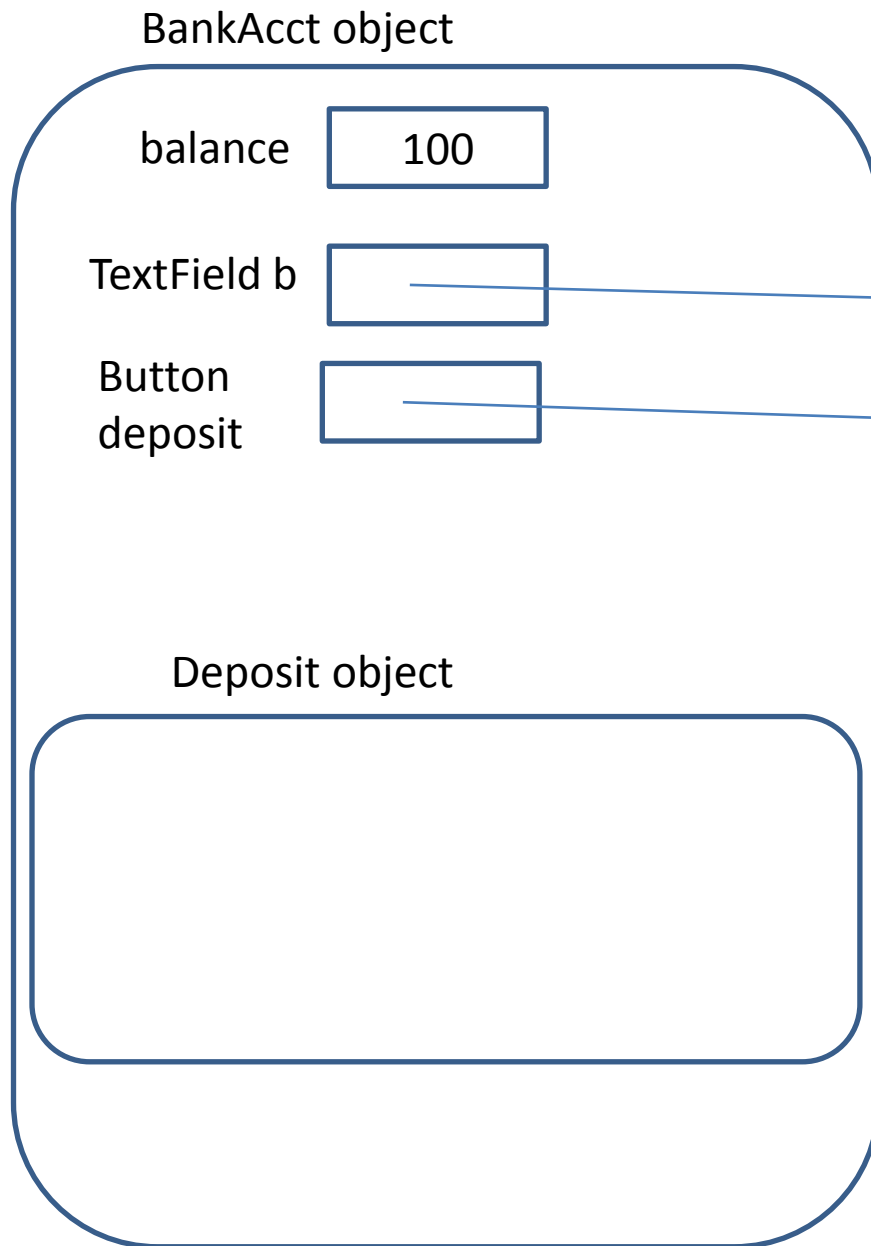
    public static void main(String[] args) {
        launch(args);
    }

    public class Deposit implements EventHandler<ActionEvent>{
        public void handle(ActionEvent e) {
            balance += 20;
            balanceField.setText(balance + "");
        }
    }
}

```

} Inner Class

Member Inner Classes



Now the Deposit has access to the variables in BankAcct directly.

Do not confuse this with subclasses!

Other ways to write an Event Handler

```
public void start(Stage primaryStage) {  
    Button deposit = new Button("Deposit $20");  
    deposit.setOnAction(new EventHandler<ActionEvent>() {  
  
        // code to handle when deposit is clicked  
        public void handle(ActionEvent e) {  
            balance += 20;  
            balanceField.setText("$"+balance);  
        }  
    });  
  
    // rest of code  
}  
  
public static void main(String[] args) {  
    launch(args);  
}  
}
```



Anonymous
Inner Class

Other ways to write an Event Handler

```
public void start(Stage primaryStage) {  
    Button deposit = new Button("Deposit $20");  
    deposit.setOnAction(e -> {  
        // code to handle when deposit is clicked  
        balance += 20;  
        balanceField.setText(balance + "");  
    });  
  
    // rest of code  
}  
  
public static void main(String[] args) {  
    launch(args);  
}  
}
```

Anonymous
Inner Class via
Lambda Expression

```

public class BankAcct extends Application {
    private int balance = 100;

    public void start(Stage primaryStage) {
        Button deposit = new Button("Deposit $20");
        Button withdraw = new Button("Withdraw $20");
        Label label = new Label("Balance: ");
        TextField balanceField = new TextField ("$" + balance);

        deposit.setOnAction(e -> {
            balance += 20;
            balanceField.setText("$" + balance);
            withdraw.setDisable(false);
        });

        withdraw.setOnAction(e -> {
            balance -= 20;
            balanceField.setText("$" + balance);
            if(balance <= 0) {
                withdraw.setDisable(true);
            }
        });

        // rest of code
    }
}

```

What happens if user hits :
 withdraw X 5, then
 deposit X 1

- A) Balance = 20, withdraw is disabled
- B) Balance = 0, withdraw is disabled
-  C) Balance = 20, withdraw is enabled
- D) Balance = 0, withdraw is enabled

Ways to write an Event Handler

// Separate class or member inner class

```
class DepositHandlerClass implements EventHandler<ActionEvent> {  
    @Override  
    public void handle(ActionEvent e){  
        System.out.println("Adding $20");  
    }  
}
```

// Anonymous Inner Class

```
btnDeposit.setOnAction(new EventHandler<ActionEvent>() {  
    public void handle(ActionEvent e){  
        System.out.println("Adding $20");  
    }  
} );
```

// Lambda expressions, shorter Anonymous Inner Classes

```
btnDeposit.setOnAction( e-> {  
    System.out.println("Adding $20");  
} );
```