SWEN20003
Object Oriented Software Development

Event Driven Programming

Shanika Karunasekera
karus@unimelb.edu.au

University of Melbourne

The Road So Far

- Java Foundations
  - A Quick Tour of Java
- Object Oriented Programming Foundations
  - Classes and Objects
  - Arrays and Strings
  - Input and Output
  - *Software Tools and Bagel*
  - Inheritance and Polymorphism
  - Interfaces and Polymorphism
- Advanced Object Oriented Programming and Software Design
  - Modelling Classes and Relationships
  - Generics
  - Collections and Maps
  - Design Patterns
  - Exceptions
  - Software Design and Testing

Lecture Objectives

After this lecture you will be able to:

- Describe the event-driven programming paradigm
- Describe where it can be applied, and how
- Implement basic event-driven programs in Java

How do programs work?

```
public static void main(String args[]) {

    Scanner scanner = new Scanner(System.in);

    String firstName = scanner.nextLine();
    String lastName = scanner.nextLine();

    String fullName = String.format("%s %s",
                                    firstName, lastName);

    System.out.format("Welcome %s!\n", fullName);
}
```

How do programs work?

Step 1: Create variable `scanner`

How do programs work?

Step 1: Create variable `scanner`

Step 2: Instantiate `Scanner` object and allocate to `scanner` variable

How do programs work?

Step 1: Create variable `scanner`

Step 2: Instantiate `Scanner` object and allocate to `scanner` variable

Step 3: Create variable `firstName`

How do programs work?

Step 1: Create variable `scanner`

Step 2: Instantiate `Scanner` object and allocate to `scanner` variable

Step 3: Create variable `firstName`

Step 4: Call `readLine` method in `scanner`

How do programs work?

Step 1: Create variable `scanner`

Step 2: Instantiate `Scanner` object and allocate to `scanner` variable

Step 3: Create variable `firstName`

Step 4: Call `readLine` method in `scanner`

Step 5: Allocate result to `firstName`

How do programs work?

Step 1: Create variable `scanner`

Step 2: Instantiate `Scanner` object and allocate to `scanner` variable

Step 3: Create variable `firstName`

Step 4: Call `readLine` method in `scanner`

Step 5: Allocate result to `firstName`

Step 6: ...

Step 7: ...

How do programs work?

Keyword

Sequential Programming: A program that is run (more or less) top to bottom, starting at the beginning of the `main` method, and concluding at its end.

How do programs work?

Keyword

Sequential Programming: A program that is run (more or less) top to bottom, starting at the beginning of the `main` method, and concluding at its end.

- Useful for “static” programs
- Constant, unchangeable logic
- Execution is the same (or very similar) each time

How do programs work?

Keyword

Sequential Programming: A program that is run (more or less) top to bottom, starting at the beginning of the `main` method, and concluding at its end.

- Useful for "static" programs
- Constant, unchangeable logic
- Execution is the same (or very similar) each time

Let's make our programs more "dynamic"...

Event-Driven Programming

Keyword

Event-Driven Programming: Using *events* and *callbacks* to control the flow of a program's execution based on changes to the program *state*.

Event-Driven Programming

Keyword

Event-Driven Programming: Using *events* and *callbacks* to control the flow of a program's execution based on changes to the program *state*.

Have we seen similar behaviour before?

Event-Driven Programming

Keyword

Event-Driven Programming: Using *events* and *callbacks* to control the flow of a program's execution based on changes to the program *state*.

Have we seen similar behaviour before?

- Exception handling → error is the example of events

when error occurs, java calls the catch block

$\Rightarrow$ catch block is your callbacks

error state occur

↓

event generated

↓

callback (catch block)

& handle the event

Event-Driven Programming

Keyword

Event-Driven Programming: Using *events* and *callbacks* to control the flow of a program's execution based on changes to the program *state*.

Have we seen similar behaviour before?

- Exception handling
- Observer pattern

↓ event : sth changing in the subject

callback : notify() method

Event-Driven Programming

Keyword

Event-Driven Programming: Using *events* and *callbacks* to control the flow of a program's execution based on changes to the program *state*.

Have we seen similar behaviour before?

- Exception handling
- Observer pattern
- Graphical User Interfaces (GUI) → event happens depend on user's action

Event-Driven Programming

- Event-driven programming is a programming style that uses signal-and-response approach to programming

Event-Driven Programming

- Event-driven programming is a programming style that uses signal-and-response approach to programming
- This style of programming abstraction is often provided by a development framework: e.g. GUI development frameworks such as Java GUI Framework JavaFX, Web Development Frameworks

Event-Driven Programming

- Event-driven programming is a programming style that uses signal-and-response approach to programming
- This style of programming abstraction is often provided by a development framework: e.g. GUI development frameworks such as Java GUI Framework JavaFX, Web Development Frameworks
- The programming style is essentially asynchronous: programmer deals with events that are generated at unknown times

→ not going by steps

Event-Driven Programming

- Event-driven programming is a programming style that uses signal-and-response approach to programming
- This style of programming abstraction is often provided by a development framework: e.g. GUI development frameworks such as Java GUI Framework JavaFX, Web Development Frameworks
- The programming style is essentially asynchronous: programmer deals with events that are generated at unknown times]

Event-Driven Programming

Keyword

State: The properties that define an object or device; for example, whether it is "active".

Event-Driven Programming

Keyword

State: The properties that define an object or device; for example, whether it is “active”.

Keyword

Event: Created when the *state* of an object/device/etc. is altered.

Event-Driven Programming

Keyword

State: The properties that define an object or device; for example, whether it is "active".

Keyword

Event: Created when the *state* of an object/device/etc. is altered.

Keyword

Callback/Listener: A method triggered by an event.

- A given component may have any number of listeners
- Each listener may respond to a different kind of event, or multiple listeners may respond to the same event
- Listeners must register with the event generator in advance

$$\begin{tikzcd}
\text{event generated} \arrow[r] & \text{subject} \arrow[d, "\text{"notify"}"] \\
& \text{callback of listener}
\end{tikzcd}$$

Event-Driven Programming

- The sending of an *event* is called *firing the event*
- An *event handler* is a method in the *listener* that specifies what will happen when events of various kinds are received by it

Event-Driven Programming

Event-driven programming is very different from most programming seen up until now:

- So far, programs have consisted of a list of statements executed in order
- When that order changed, whether or not to perform certain actions (such as repeat statements in a loop, branch to another statement, or invoke a method) was controlled by the logic of the program

Event Handlers

- In event-driven programming, objects are created that can fire events, and listener objects are created that can react to the events
- The next thing that happens depends on the next event
- The program itself no longer determines the order in which things can happen - instead, the events determine the order
- In particular, methods are defined that will never be explicitly invoked in any program written by the developer Instead, methods are invoked automatically when an event signals that the method needs to be called

Event-Driven Programming

We will learn event driven programming through writing a simple Graphical User Interface (GUI) application.

We will use `JavaFX`, the java GUI development framework to demonstrate an example of event driven programming.

You are not expected to know how to program in `JavaFX` - the following examples are only to demonstrate the event driven programming paradigm.

Event-Driven Programming

When we use a GUI, what *events* might we respond to?

Event-Driven Programming

When we use a GUI, what *events* might we respond to?

- Mouse
- Keyboard
- Touch
- Controller (e.g. XBox controller)

Event-Driven Programming

When we use a GUI, what *events* might we respond to?

- Mouse
- Keyboard
- Touch
- Controller (e.g. XBox controller)

Assuming we could respond to those events, what *features* could we add?

Event-Driven Programming

When we use a GUI, what *events* might we respond to?

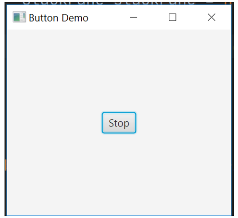
- Mouse
- Keyboard
- Touch
- Controller (e.g. XBox controller)

Assuming we could respond to those events, what *features* could we add?

- Mouse: Menu buttons, GUI controls, click-to-move
- Keyboard: movement, "shooting", special powers
- Touch/Controller: similar to keyboard

Event-Driven Programming - Example

Write a java program to display the following GUI window with a single button.



When the user clicks the "Stop" button, the program should print to the console:

```
I am the stop button.
```

Event-Driven Programming - Example

Following is a JavaFX program to display the button.

```java
// imports required for JavaFX classes are not shown

public class DisplayButton extends Application {
    private Button stopButton;
    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
        primaryStage.setTitle("Button Demo"); → set title for the page
        stopButton = new Button();
        stopButton.setText("Stop");
        StackPane stackPane = new StackPane();
        stackPane.getChildren().add(stopButton);
        primaryStage.setScene(new Scene(stackPane, 300,275));
        primaryStage.show();  → make screen appear
        System.out.println("Displaying window");
    }
    public static void main(String[] args) {
        launch(args);  → run start method
    }
}
```

Event-Driven Programming

Now we will add code to handle the "mouse click event" and print a message. We will first develop the event handler code.

interface

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

public class StopButtonHandler  implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent actionEvent) {
        System.out.println("I am the stop button.");
    }
}

```

Event-Driven Programming - Example

We will now add the event handling code.

Model view Controller Paradigm

all the logic we deal with should be seperated from the view

⇓

decoupling

```
// imports required for JavaFX classes are not shown

public class DisplayButton extends Application {
    private Button stopButton;
    @Override
    public void start(Stage primaryStage) throw Exception{
        Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
        primaryStage.setTitle("Button Demo");
        stopButton = new Button();
        stopButton.setText("Stop");
        stopButton.setOnAction(new StopButtonHandler());
        StackPane stackPane = new StackPane();
        stackPane.getChildren().add(stopButton);
        primaryStage.setScene(new Scene(stackPane, 300,275));
        primaryStage.show();
        System.out.println("Displaying window");
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Event-Driven Programming

What makes this approach better?

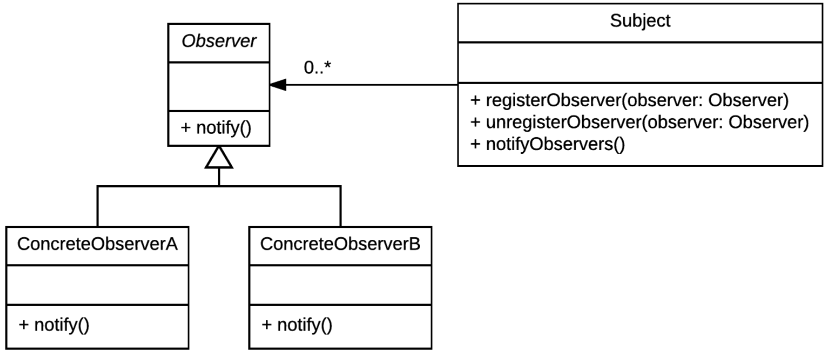
Event-Driven Programming

What makes this approach better?

Using an event-driven approach allows us to:

- Better *encapsulate* classes by hiding their behaviour
- Avoid having to explicitly send information about the input; instead it is automatically passed as part of the *callback*
- Easily add/remove behaviour to classes
- Easily add/remove additional *responses*

Observer Pattern Reminder



Event-Driven Programming

Real-world examples:

- Graphical User Interfaces (GUIs)
- Web development/Javascript
- Embedded Systems/Hardware

Event-driven programming is a very powerful technique, and is normally supported through a software framework such as the JavaFX GUI development framework we looked at.

Software Development Frameworks

- An Object-oriented application development framework is a set of cooperating classes that represent reusable designs
- A framework:
  - normally consists of a set of abstract classes (that are partially complete) and interfaces
  - the partially complete classes can be customized to meet application needs
- Examples of software development frameworks you have seen so far: Java I/O Framework, Java Collections/Maps Frameworks, JavaFX GUI Development Framework, Bagel

Software Development Frameworks - Key Features

- **Extensibility:**
  - Consists of a set of abstract classes and interfaces to be extended and specialized
  - The changeable aspects are represented as hook methods ?
- **Inversion of Control:**
  - With a conventional library the application controls the flow of execution application acts as the master
  - With an application development framework, often the flow of execution often resides in the framework - the framework acts as the master
- **Design Patterns as building blocks:**
  - Most frameworks use design patterns as building blocks

Lecture Objectives

After this lecture you will be able to:

- Describe the event-driven programming paradigm
- Describe where it can be applied, and how
- Implement basic event-driven programs in Java