Graphical interfaces and event-driven programming

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 12

GUIs and events

- How do we design graphical user interfaces?

GUIs and events

- How do we design graphical user interfaces?
- Multiple applications simultaneously displayed on screen

GUIs and events

- How do we design graphical user interfaces?
- Multiple applications simultaneously displayed on screen
- Keystrokes, mouse clicks have to be sent to appropriate window

GUIs and events

- How do we design graphical user interfaces?
- Multiple applications simultaneously displayed on screen
- Keystrokes, mouse clicks have to be sent to appropriate window
- In parallel to main activity, record and respond to these `events`
  - Web browser renders current page
  - Clicking on a link loads a different page

Keeping track of events

- Remember coordinates and extent of each window

Keeping track of events

- Remember coordinates and extent of each window
- Track coordinates of mouse

Keeping track of events

- Remember coordinates and extent of each window
- Track coordinates of mouse
- OS reports mouse click at (x, y)
  - Check which windows are positioned at (x, y)
  - Check if one of them is "active"
  - Inform that window about mouse click

Keeping track of events

- Remember coordinates and extent of each window
- Track coordinates of mouse
- OS reports mouse click at (x, y)
  - Check which windows are positioned at (x, y)
  - Check if one of them is "active"
  - Inform that window about mouse click
- Tedious and error-prone

Keeping track of events

- Remember coordinates and extent of each window
- Track coordinates of mouse
- OS reports mouse click at (x, y)
  - Check which windows are positioned at (x, y)
  - Check if one of them is "active"
  - Inform that window about mouse click
- Tedious and error-prone
- Programming language support for higher level events

Keeping track of events

- Remember coordinates and extent of each window
- Track coordinates of mouse
- OS reports mouse click at (x, y)
  - Check which windows are positioned at (x, y)
  - Check if one of them is "active"
  - Inform that window about mouse click
- Tedious and error-prone
- Programming language support for higher level events
  - Run time support for language maps low level events to high level events

Keeping track of events

- Remember coordinates and extent of each window
- Track coordinates of mouse
- OS reports mouse click at (x, y)
  - Check which windows are positioned at (x, y)
  - Check if one of them is "active"
  - Inform that window about mouse click
- Tedious and error-prone
- Programming language support for higher level events
  - Run time support for language maps low level events to high level events
  - OS reports low level events: mouse clicked at (x, y), key `'a'` pressed

Keeping track of events

- Remember coordinates and extent of each window
- Track coordinates of mouse
- OS reports mouse click at (x, y)
  - Check which windows are positioned at (x, y)
  - Check if one of them is "active"
  - Inform that window about mouse click
- Tedious and error-prone
- Programming language support for higher level events
  - Run time support for language maps low level events to high level events
  - OS reports low level events: mouse clicked at (x, y), key `'a'` pressed
  - Program sees high level events: Button was clicked, box was ticked ...

Better PL support for events

- Programmer directly defines components such as windows, buttons, . . . that "generate" high level events

Better PL support for events

- Programmer directly defines components such as windows, buttons, . . . that "generate" high level events
- Each event is associated with a listener that knows what to do
  - e.g., clicking `Close window` exits application

Better PL support for events

- Programmer directly defines components such as windows, buttons, . . . that "generate" high level events
- Each event is associated with a listener that knows what to do
  - e.g., clicking `Close window` exits application
- Programming language has mechanisms for
  - Describing what types of events a component can generate
  - Setting up an association between components and listeners

Better PL support for events

- Programmer directly defines components such as windows, buttons, ... that "generate" high level events
- Each event is associated with a listener that knows what to do
  - e.g., clicking `Close window` exits application
- Programming language has mechanisms for
  - Describing what types of events a component can generate
  - Setting up an association between components and listeners
- Different events invoke different functions
  - Window frame has `Maximize`, `Iconify`, `Close` buttons
- Language "sorts" out events and automatically calls the correct function in the listener

An example

- A `Button` with one event, press button

An example

- A `Button` with one event, press button
- Pressing the button invokes the function `buttonpush(..)` in a listener

```
interface ButtonListener{
  public abstract void buttonpush(...);
}

class MyClass implements ButtonListener{
  ...
  public void buttonpush(...){
     ...       // what to do when
               // a button is pushed
  }
}
```

An example

- A `Button` with one event, press button
- Pressing the button invokes the function `buttonpush(..)` in a listener
- We have set up an association between `Button b` and a listener `ButtonListener m`

```
interface ButtonListener{
  public abstract void buttonpush(...);
}

class MyClass implements ButtonListener{
  ...
  public void buttonpush(...){
     ...       // what to do when
               // a button is pushed
  }
}

Button b = new Button();
MyClass m = new MyClass();
b.add_listener(m);    // Tell b to notify
                      // m when pushed
```

An example

- A `Button` with one event, press button
- Pressing the button invokes the function `buttonpush(..)` in a listener
- We have set up an association between `Button b` and a listener `ButtonListener m`
- Nothing more needs to be done!

```
interface ButtonListener{
  public abstract void buttonpush(...);
}

class MyClass implements ButtonListener{
  ...
  public void buttonpush(...){
     ...       // what to do when
               // a button is pushed
  }
}

Button b = new Button();
MyClass m = new MyClass();
b.add_listener(m);    // Tell b to notify
                      // m when pushed
```

An example . . .

- Communicating each button push to the listener is done automatically by the run-time system

```
interface ButtonListener{
  public abstract void buttonpush(...);
}

class MyClass implements ButtonListener{
  ...
  public void buttonpush(...){
     ...       // what to do when
               // a button is pushed
  }
}

Button b = new Button();
MyClass m = new MyClass();
b.add_listener(m);    // Tell b to notify
                      // m when pushed
```

An example . . .

- Communicating each button push to the listener is done automatically by the run-time system
- Information about the button push event is passed as an object to the listener

```
interface ButtonListener{
  public abstract void buttonpush(...);
}

class MyClass implements ButtonListener{
  ...
  public void buttonpush(...){
     ...       // what to do when
               // a button is pushed
  }
}

Button b = new Button();
MyClass m = new MyClass();
b.add_listener(m);    // Tell b to notify
                      // m when pushed
```

An example . . .

- Communicating each button push to the listener is done automatically by the run-time system
- Information about the button push event is passed as an object to the listener
- `buttonpush(...)` has arguments
  - Listener can decipher source of event, for instance

```
interface ButtonListener{
  public abstract void buttonpush(...);
}

class MyClass implements ButtonListener{
  ...
  public void buttonpush(...){
     ...       // what to do when
               // a button is pushed
  }
}

Button b = new Button();
MyClass m = new MyClass();
b.add_listener(m);    // Tell b to notify
                      // m when pushed
```

Timer

- Recall `Timer` example

Timer

- Recall `Timer` example
- `Myclass m` creates a `Timer t` that runs in parallel

Timer

- Recall `Timer` example
- `Myclass m` creates a `Timer t` that runs in parallel
- `Timer t` notifies a `Timerowner` when it is done, via a function `timerdone()`

Timer

- Recall `Timer` example
- `Myclass m` creates a `Timer t` that runs in parallel
- `Timer t` notifies a `Timerowner` when it is done, via a function `timerdone()`
- Abstractly, timer duration elapsing is an event, and `Timerowner` is notified when the event occurs
  - In the timer, the notification is done explicitly, manually
  - In the button example, the notification is handled internally, automatically

Timer

- Recall `Timer` example
- `Myclass m` creates a `Timer t` that runs in parallel
- `Timer t` notifies a `Timerowner` when it is done, via a function `timerdone()`
- Abstractly, timer duration elapsing is an event, and `Timerowner` is notified when the event occurs
  - In the timer, the notification is done explicitly, manually
  - In the button example, the notification is handled internally, automatically
- In our example, `Myclass m` was itself the `Timerowner` to be notified

Timer

- Recall `Timer` example
- `Myclass m` creates a `Timer t` that runs in parallel
- `Timer t` notifies a `Timerowner` when it is done, via a function `timerdone()`
- Abstractly, timer duration elapsing is an event, and `Timerowner` is notified when the event occurs
  - In the timer, the notification is done explicitly, manually
  - In the button example, the notification is handled internally, automatically
- In our example, `Myclass m` was itself the `Timerowner` to be notified
- In principle, `Timer t` could be passed a reference to any object that implements `Timerowner` interface

Summary

- Event driven programming is a natural way of dealing with graphical user interface interactions
- User interacts with object through mouse clicks etc
- These are automatically translated into events and passed to listeners
- Listeners implement methods that react appropriately to different types of events