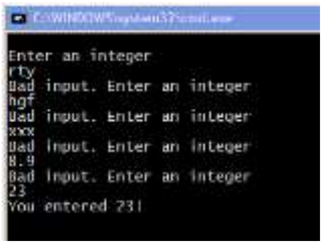
Event driven architecture and Graphical User Interfaces

- In this section:
  - Sequential vs. Event-driven Programming
  - Reacting to the user
  - The concept of *event-driven programs*, and the *event loop*
  - “Event-handlers", methods and listener classes
  - Simple example event based GUI program

Sequential Programming

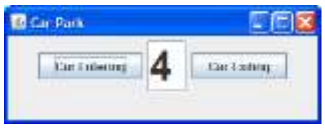
- Your Java programs so far have been *sequential code*
- In sequential programs, *the program is under control*
- The user is required to synchronize with the program:
  - Program tells user it’s ready for more input
  - User enters more input and it is processed
- Examples:
  - Command-line prompts ("DOS", Linux)
  - MATLAB (an interactive mathematical system)
  - Stata (a statistics package)
- *Shouldn’t the program be required to synchronize with the user?*

- Flow of a typical sequential program
  - Prompt the user
  - Read input from the keyboard
  - Parse/analyse the input
  - Evaluate the result
  - Generate output
  - Repeat
- Advantages
  - Simple concept
- Limitations
  - Difficult to implement complex interactions
  - Interaction must proceed according to a pre-defined sequence



```
Enter an integer
rty
Bad input. Enter an integer
hgf
Bad input. Enter an integer
xxx
Bad input. Enter an integer
8.9
Bad input. Enter an integer
23
You entered 23!
```

- In contrast: **Event-driven programming**
  - The user is in control
  - Example:
    An application with two buttons to simulate cars entering and leaving a car park
    The total number of cars in the car park is always displayed in a text field



Demo

Event-driven Programming

- Instead of a user synchronizing with the program, the program synchronizes with, or reacts to, the user
- All communication from user to computer occurs via *events* and the code that handles the events
- An event is an action that happens "to" the system
  - A mouse button pressed or released
  - A keyboard key is hit
  - A window is moved, exposed, resized, closed, etc.
- *The user chooses which events happen and when*
- There are
  - User-initiated events
  - System-initiated events

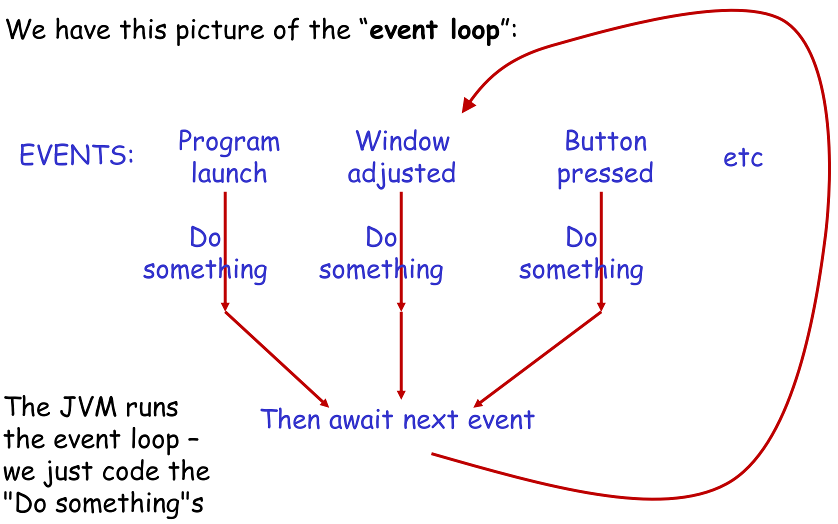
Event-driven program execution

- The basic behaviour of most programs is
  - Start
  - Wait for any event
  - Then to react appropriately to it
  - And back to waiting (for ever!)

The user controls the sequence of events – not the program

- This cycle is called "The Event Loop"
- Most applications are built like this. For example:
  - Microsoft Word
  - Windows itself
- Our GUI Java programs will work exactly like this
- Java's window manager sends event notifications to the program
  - E.g. on a key press, mouse click on GUI button, …
  - Including information about the *source*: e.g. which keyboard key, GUI button, …

We have this picture of the "**event loop**":



The JVM runs the event loop – we just code the "Do something"s

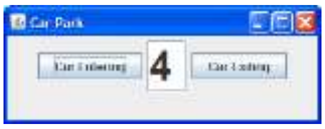
Events and event-handlers

- Each "Do something" is called an "event-handler"
- **Event-handlers are *methods with specific names***
  - **These are called *automatically* by the JVM when a recognised event occurs**
  - The method bodies encode the "appropriate reaction"
- Examples:
  - **`main`** is the event-handler for launching ("pseudo event")
  - **`actionPerformed`** is the handler for GUI button "presses"
  - **`mouseMoved`** is…. (obvious)
- Event-handlers have parameters that the JVM uses to convey details about the event *to the event handler*
  - Example:

```
public void actionPerformed(ActionEvent event)
```

  - Parameter **`event`** indicates which GUI button was clicked

Event handling in the Car Park application



- There are two GUI buttons
  - Java **JButtons**
  - Called **enter** and **exit**
- And one Java **JTextField** called **text**
- Whichever button is clicked, the event handler **actionPerformed** is called
  - It needs to check the *source* of the event...
  - ... to carry out the correct action
  - Either incrementing or decrementing the counter and updating the display

actionPerformed

– called *every time* a button is clicked

```
public void actionPerformed(ActionEvent event)
{
    if (event.getSource() == enter)
    {
        carCount = carCount + 1;
    }
    if (event.getSource() == exit)
    {
        carCount = carCount-1;
    }
    text.setText(Integer.toString(carCount));
}
```

Note: **public**, **event.getSource**, **carCount**, **text.setText**
Further **carCount** is *not* declared in this method

The full Car Park application

- Next lecture:
  - How the GUI is set up on the screen
  - How the event handling is linked up
  - How the whole program is organized

- There are many ways to build a Java GUI program
  - Radically different approaches in different books
  - Some *very complex* in their use of object orientation
  - We will take a simpler approach from Java for Students, by Bell and Parr (in library)

End of section