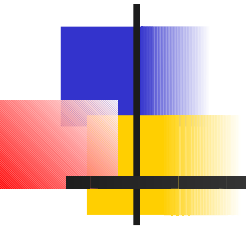
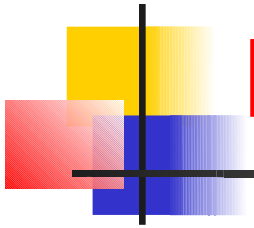


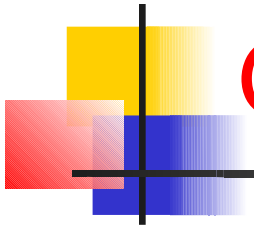
Graphical User Interfaces

Corky Cartwright
Stephen Wong
Department of Computer Science
Rice University



Human Computer Interaction

- Original computer interfaces
  - Sequences of numbers/characters:
    punch cards
    paper tape
    console switches and lights
    line printers
    magnetic tape
    typewriter terminals (teletypes)
  - Original interactive interfaces were based on streams of characters
  - Well suited to some applications because it accommodates the input of arbitrary program text
    Unix shell (bash)
    Windows command line

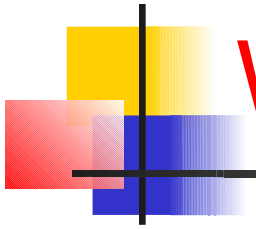


Graphical interaction

- Labeled buttons
- Text boxes for character input
- Mouse actions for selection, cutting, pasting, tracking/drawing
- What is the programmatic interface for graphical interaction (*i.e.*, what does the program receiving graphical input see)?

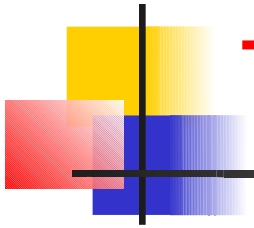
  A sequence of events

- What are events?



What are Events?

- In an OO language, they are objects describing graphical inputs. Generated by low-level code in the VM supporting the OO language. The low-level code processes hardware interrupts. Intuition: messages.
- A GUI (graphical user interface) library defines and supports the event system.
- The nitty gritty systems level code supporting the event library is based on interrupt-handling in the operating system (Comp 221, Comp 421)
- From the perspective of high-level language programming, events are are simply the elements of a much richer input stream than ASCII characters.
- Very close connection between text processing and event processing



Text processing template

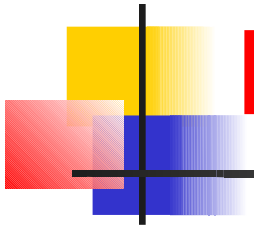
- ```
  while (! input.endOfFile) {
    read(input);
    process(input);
  }
  ```
- Sometimes end-of-file is handled as an exception:

  ```
  try {
   while (true) {
     read(input);
     process(input);
   }
  }
  catch(EOFException e) { ... }
  ```
- If a Java program tries to read past end-of-file, the read operation throws a java.io.??.
  ```

Event processing template

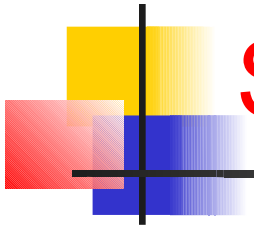
```
while (true) {
   get next event;
   process event;
}
```

- Processing an event may terminate application. The event may be “close this application”; code simply performs any necessary clean-up and breaks.
- In Java, the GUI library includes a separate thread that processes events using a loop as shown above.
  The GUI library presumes that all event processing is done in the event-handling loop. In fact, it assumes that *all method calls on GUI components (unless the method contract states otherwise) are called from the event-processing thread*.



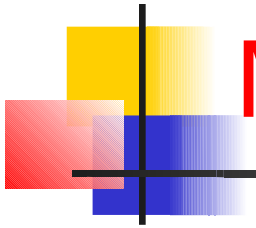
Key Intuitions

- A program driven by a GUI simply processes a stream of events.
- Key question: what about pre-emption? In an internet browser, you can do other things while a page is being loaded. How can an event be processed without holding subsequent event processing pending?
- Answer: spawn an independent thread to process the event if it cannot be done quickly.
- HUGE problem: concurrent (multi-threaded) programming is HARD and full of hidden gotcha's.



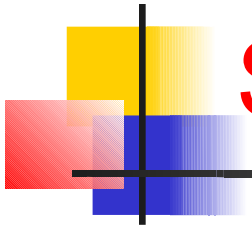
Simple GUI Applications

- No pre-emption/concurrency. Embedded application (often called he model) is unaware of the GUI.
- How is such an application organized? The “pure” model-view-controller (MVC) pattern.
- Pure MVC:
  - On startup, the controller (which should be an object) runs. (How can we avoid making the controller an object in Java? Use “static” code, typically the main() method.)
  - The controller creates the model (which should be structured as an object) and the GUI (called the “view”) which is also organized as an object (*e.g.* a JFrame)
  - The controller links the view to the model by attaching listeners (commands) to GUI elements (buttons, etc) that are run when that GUI element is activated. The listener code performs some action on the model. The model is passive; it does not talk to the GUI.
  - The controller activates the GUI and immediately terminates.



Motivation for MVC

- We partition GUI applications using the MVC (model-view-controller) pattern so that new views can potentially be created without changing the logic (model) of the underlying application.
- This is a specific example of the general design concept called *de-coupling*: partition an application into independent components that interact only through explicitly declared interfaces. Unfortunately, it is easy to violate this discipline in Java and other languages. In Java, a file can import (or directly access using fully qualified path names) any public class that it wants.



Simple GUI Threading

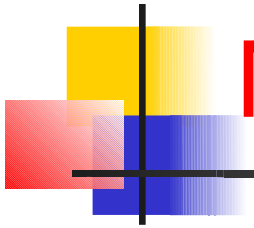
- How do GUI events get recognized and processed?
- A GUI system *requires* a dedicated thread that performs the event loop we showed earlier:

```
while (true) {
  get next event;
  process event;
}
```

- How do we avoid perils of multi-threading? The event thread is started by the GUI library when the GUI is activated (typically by making a GUI component visible). Prior to that time, no event handling thread exists.
- In the pure MVC model, the controller dies immediately after activating the GUI so there is never more than one thread executing in the program!

Example: a click counter

- Explicated in the OO Design Notes



More Complex GUIs

- In some cases the model must be aware of the GUI and explicitly call methods in it, *e.g.*, DrJava or a game playing program.
  - The model includes a reference to the GUI specified by an interface with limited GUI functionality. The model implements this interface. The controller passes a reference to the GUI when it sets up the model.
  - The limited GUI interface should be developed as part of the model and include only what the model needs.
- Sophisticated "view" components may spawn asynchronous threads to process an event that would otherwise lock-up the GUI. This coding practice (explicit concurrency) is treacherous because special protocols must be used in to access shared data and the methods **`invokeLater`** and **`invokeAndWait`** must be used to move code fragments (expressed using the **`Runnable`** interface) to the event-handling thread in the GUI.