

Principles of Programming Languages

So, what was I supposed to learn here?

Other programming paradigms

- Event-driven programming
- Parallel programming
- Spatial programming

Lecture 27, CS314, parts copyrighted by Prof. Ryder

1

Principles of Programming Languages

Why don't we all program in machine language?

- Languages are tools to make programming easier
- Each language is based on a different “world model”
 - ⇒ you need to match your problem with the best language to solve it.

Lecture 27, CS314, parts copyrighted by Prof. Ryder

2

Principles of Programming Languages

What is the best language for my problem?

- **Programming abstraction**
 - imperative, functional, logic, OO, event-driven, ...
- **Language “characteristics”**
 - **safety**: can I do “bad” things, and will I know?
 - **performance**: time, space, power ...
 - **portability**: platform independent
 - **flexibility**: eager vs. lazy, interpreted vs. compiled

Lecture 27, CS314, parts copyrighted by Prof. Ryder

3

Principles of Programming Languages

Lessons to learn:

- **Each language chooses a particular tradeoff**
 - ⇒ there is no free lunch
- **Some language features are more expensive than others**
- **Languages and compilers are an active research area**

Lecture 27, CS314, parts copyrighted by Prof. Ryder

4

Event-driven Programming

- **What is it?**
- **Event model with handling of events**
- **Examples**
 - **ATM machine**
- (Lecture notes source: **Programming Languages**, A. Tucker, R. Noonan. McGraw Hill, 2001)

Lecture 27, CS314, parts copyrighted by Prof. Ryder

5

Event-driven Program

- **Designed to not terminate**
 - **Runs for arbitrary length of time**
- **Reacts to arbitrary sequences of events during execution**
 - **Computation modeled as *interaction***
 - **“Computation is a community of persistent entities coupled together by their ongoing interactive behavior...Beginning and end, when present, are special cases that can often be ignored”, L.Stein, “Challenging the computational metaphor: implications for how we think”, *Cybernetics and Systems*, 20, 6, 1999.**

Lecture 27, CS314, parts copyrighted by Prof. Ryder

6

Event-driven Programs

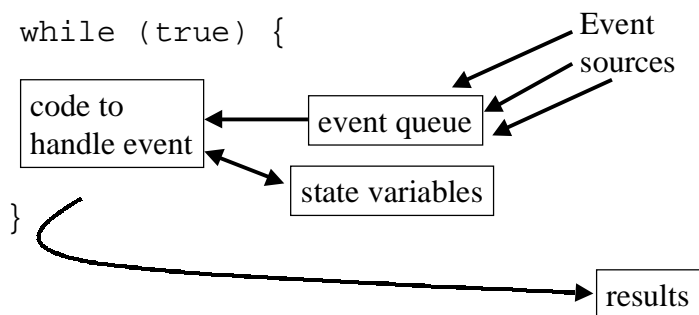
- **Examples**
 - GUI mouse and screen interface
 - Web-based applications (e.g., online registration systems, online shopping sites)
 - Embedded control systems (e.g., in planes)
- **Language support in Java, Visual Basic, Tcl/Tk**

Lecture 27, CS314, parts copyrighted by Prof. Ryder

7

Event-driven Programs

- **Conceptual model of an event-driven program**

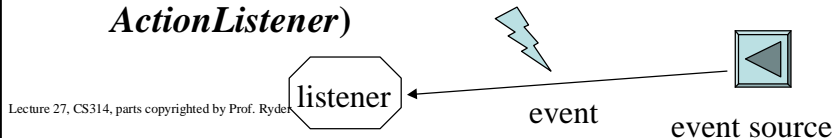


Lecture 27, CS314, parts copyrighted by Prof. Ryder

8

Java Events

- **Events** are subclasses of abstract class *AWTEvent* (e.g., *ActionEvent*)
- **Event sources** are objects that are instances of subclasses of abstract class *Component* (e.g., *Button*)
 - **Listeners** in program recognize when a particular **event** has occurred on the object which is the event source
 - A **listener** handles the **event** (e.g., implements *ActionListener*)



Listening for Events in Java

- **How does *listening* for events work?**
 - When user pushes a mouse button, an event occurs
 - An object that implements the appropriate interface (e.g., *ActionListener*) is registered as an *event listener* on the appropriate *event source* (e.g., *Button*)
 - Each event is represented by an object that has information about the event and the event source
 - Listener/event source relation can be many-to-many
 - One listener can register with several event sources
 - Multiple listeners can register with the same event source

ATM Application

- **Transaction-based interactions with user**
- **Minimal ATM**
 - **Actions:** *deposit, withdrawal, balance?, end session*
 - **Shows:** *Account number, amount, message*
- **Consider the state variables**
 - *Account, type, amount, message, balance*
 - *Balance* forces program to interact with database, not just a user

Lecture 27, CS314, parts copyrighted by Prof. Ryder

11

ATM

- **Event 1 - enter account number through card swipe**
 - **Handled by:** Check number is valid, set *balance*, ask user to select transaction
- **Event 2 - user presses a button**
 - **Handled by:** Check a valid account has been entered, save *type* of button pressed;
 - If deposit/withdrawal, issue *message* 'enter amount'
 - If balance?, display *balance*
 - If end session, clear the account info in state variables

Lecture 27, CS314, parts copyrighted by Prof. Ryder

12

ATM

- **Event 3 - User enters amount**
 - **Handled by: Check either deposit or withdrawal is type of transaction;**
 - **If deposit, add to *balance***
 - **If withdrawal**
 - **If *balance* is larger than *amount*, subtract *amount* from *balance***
 - **Otherwise, issue *message* ‘insufficient funds’**
 - **Otherwise, issue *message* ‘please select transaction desired’**

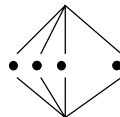
Lecture 27, CS314, parts copyrighted by Prof. Ryder

13

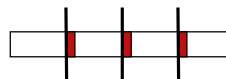
Parallel Programming

- **Shared-memory vs. distributed memory**
- **task parallel vs. data parallel**

```
doall i=1, 100
  A(i) = B(i+1) + C(i+1)
```



```
distribute A, B, C onto procs(4)
send(left, (B(1), C(1)))
receive(right, (B(26), C(26)))
for i=1, 25
  A(i) = B(i+1) + C(i+1)
```

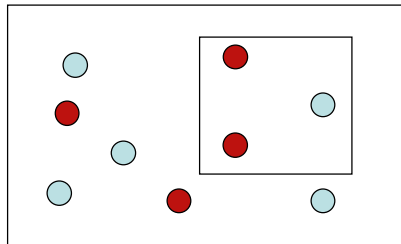


Lecture 27, CS314, parts copyrighted by Prof. Ryder

14

Spatial Programming

- **NES: network of embedded systems**
- **volatile and unreliable target platform**
- **location within physical space is important**
- **nodes provide different services**



Lecture 27, CS314, parts copyrighted by Prof. Ryder

15