

Objectives

- Java Wrap up
 - Standard Error
 - CLASSPATH
 - Jar Files
- Language Comparison
- Software Development

Oct 17, 2011

Sprenkle - CSCI209

1

Discussion

- Who is Dennis Ritchie?

Oct 17, 2011

Sprenkle - CSCI209

2

Dennis Ritchie



Dennis Ritchie (standing) and Ken Thompson at a PDP-11 in 1972. (Photo: Courtesy of Bell Labs)

- Creator of C programming language
- Built Unix using C with Ken Thompson

Ritchie's running joke was that C had "the power of assembly language and the convenience of ... assembly language."

"UNIX is very simple, it just needs a genius to understand its simplicity."

09

3

Discussion: Site Spoofing



CSCI 325 Spring 2010 Project

Review

- If I only have a certain number of possible valid object values, what should I use?
- Why do we need Comparators?
- How does Java resolve methods that are
 - Overridden
 - Overloaded

Oct 17, 2011

Sprenkle - CSCI209

5

STANDARD ERROR

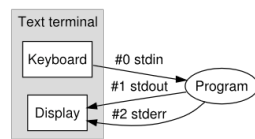
Oct 17, 2011

Sprenkle - CSCI209

6

Standard Streams

- Preconnected streams
 - Standard Out: `stdout`
 - Standard In: `stdin`
 - *Standard Error: `stderr`*
 - For error messages and diagnostics
 - In Java: `System.err`
- Benefits of two output streams
 - Redirect to different places
 - Example: separate log files for info and for errors
 - Look at some web logs



Oct 17, 2011

Sprenkle - CSCI209

7

Redirecting Output

- Recall earlier this semester
 - `> java OlympicScore > score.out`
 - Redirected `stdout` to `score.out`
 - `stderr` would still go to terminal
- To redirect `stderr` to file as well
 - `> java OlympicScore >& score.out`

Oct 17, 2011

Sprenkle - CSCI209

8

CLASSPATH

Oct 17, 2011

Sprenkle - CSCI209

9

Classpath

- Tells the compiler or JVM where to look for user-defined classes and packages
 - Often when using third-party libraries
- Similar to `PYTHONPATH`
- Typically know it needs to be set when there are class not found error messages

Oct 17, 2011

Sprenkle - CSCI209

10

Setting the Classpath

- Can specify classpath in command line


```
javac -cp path/to/myjavaclasses MyClass.java
java -cp path/to/myjavaclasses MyClass
```
- Can specify the classpath environment variable
 - Edit your `.bash_profile` OR
 - Set in terminal


```
CLASSPATH=$CLASSPATH:path/to/myjavaclasses
echo $CLASSPATH
```

← Current value of `CLASSPATH`
- In Eclipse, you can “Configure Build Path” for a project

Oct 17, 2011

Sprenkle - CSCI209

11

JAR FILES

Oct 12, 2011

Sprenkle - CSCI209

12

Jar (Java Archive) Files

- Archives of Java class files
- Package code into a neat bundle to **distribute**
 - Easier, faster to download
 - Easier for others to use
- Optional: include source code

Oct 12, 2011

Sprenkle - CSCI209

13

Using Jars

- Add jar files to CLASSPATH to use classes in jar file
- Add a jar file to the current classpath environment variable (\$CLASSPATH)

```
CLASSPATH=$CLASSPATH:myapplication.jar
echo $CLASSPATH
```

 Current value of CLASSPATH

- Or add to classpath as command-line option


```
java -cp mail.jar:email.jar grading.Email ...
```
- In Eclipse, you can "Configure Build Path"

Oct 12, 2011

Sprenkle - CSCI209

14

Jar/Tar Command

- jar** command: Create, view, extract Jar files
- Common use:
 - `jar cfz archive.jar.gz arch_directory`
 - `jar xfz archive.jar.gz`

Option/Operation	Meaning	Similar to tar
f	The name of the archive file	
c	Create an archive file	
x	Extract the archive file	
v	Verbose	
z	Zip (compress)	
t	Table of contents (list contents)	

Oct

15

Jar file: Metadata

- Jar file includes a special metadata file with the path META-INF/MANIFEST.MF
 - Describe how Jar file is used
 - jar** creates a default metadata file, if not specified

Oct 12, 2011


Sprenkle - CSCI209

16

Jar file: Metadata


- Example metadata file that allows you to execute the JAR with java

```
Manifest-Version: 1.0
Main-Class: MyApplication
```

 Note the newline

- To create the jar file:


```
jar cmf myManifest myapplication.jar *.class
```

 Specifying the metadata file
- Run Main-Class using java


```
java -jar myapplication.jar
```

Oct 12, 2011

Sprenkle - CSCI209

17

Creating Jar Files in Eclipse

- Export → Java → Jar
 - Options to create a MANIFEST.MF file
 - Options to include source files or only class files
- Can submit assignments this way
 - Must include source files**

Oct 12, 2011

Sprenkle - CSCI209

18

LANGUAGE COMPARISON

Oct 17, 2011

Sprenkle - CSCI209

19

Language Comparison

Java

Python

Oct 17, 2011

Sprenkle - CSCI209

20

Language Comparison

Java

- Object-oriented
- Statically typed
- Compiled

Python

- Object-oriented
- Dynamically typed
- Interpreted

Pros and cons of using each?

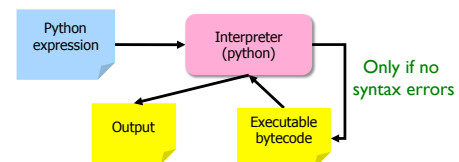
Oct 17, 2011

Sprenkle - CSCI209

21

Python Interpreter

1. Validates Python programming language expression(s)
 - Enforces Python syntax rules
 - Reports syntax errors
2. Executes expression(s)

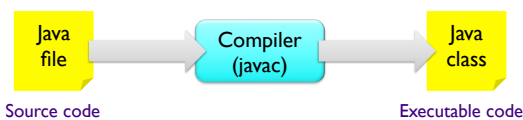


Oct 17, 2011

Sprenkle - CSCI209

22

Java Compiler



- Lexical analysis, parsing, semantic analysis, *code generation*, and *code optimization*
- Code optimization: dead code eliminator, inline expansion, constant propagation, ...

Oct 17, 2011

Sprenkle - CSCI209

23

Compiling

- Translates high-level programming language to machine code or byte code
 - Java: .class → bytecode
- Compiler optimization techniques
 - Generate *efficient* bytecode/machine code
 - Examples: get rid of unused local variables, transform loops, inline method calls
 - In Java: static typing for additional gains
- Can execute generated code multiple times
 - Performance gain
 - Interpreted → have to re-verify the code each time executed

What can we do in Python that we can't do in Java?

Oct 17, 2011

Sprenkle - CSCI209

24

Summary: Compiled vs Interpreted Languages

Compiled

- Spends a lot of time analyzing and processing the program
- Resulting executable is some form of machine-specific binary code
- Computer hardware interprets (executes) resulting code
- Program execution is fast
 - Efficient machine/byte code generation
 - Performance gains

Interpreted

- Relatively little time spent analyzing and processing the program
- Resulting code is some sort of intermediate code
- Another program interprets resulting code
- Program execution is relatively slow
- Faster development/prototyping

Oct 17, 2011

Sprenkle - CSCI209

25

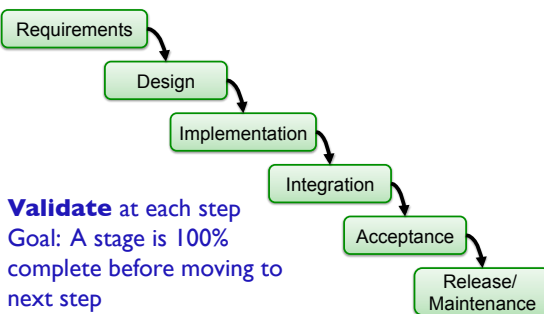
SOFTWARE LIFE CYCLE

Oct 17, 2011

Sprenkle - CSCI209

26

Traditional Software Engineering Process: Waterfall Model

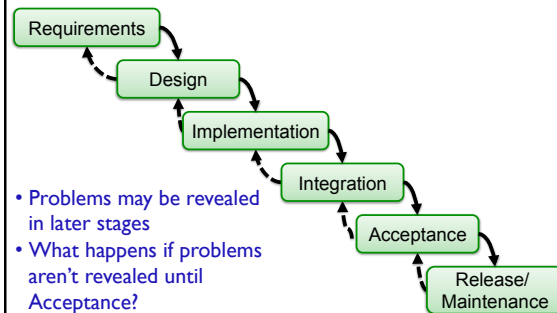


Oct 17, 2011

Sprenkle - CSCI209

27

Feedback in Waterfall Model

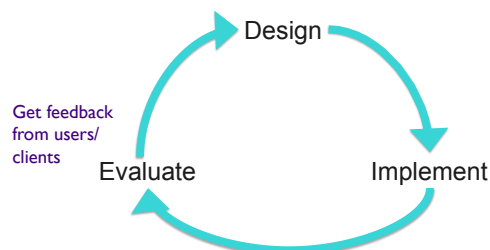


Oct 17, 2011

Sprenkle - CSCI209

28

Iterative Design



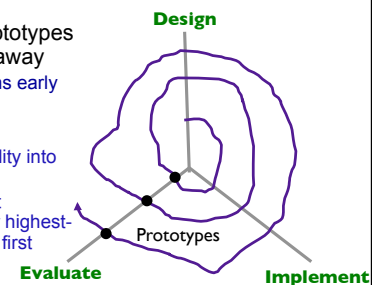
Oct 17, 2011

Sprenkle - CSCI209

29

Spiral Model

- Idea: smaller prototypes to test/fix/throw away
 - Finding problems early costs less
- In general...
 - Break functionality into smaller pieces
 - Implement most depended-on or highest-priority features first



[Boehm 86]

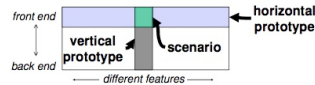
Oct 17, 2011

Sprenkle - CSCI209

30

Prototypes

- Purpose/Dimensions
 - Functionality
 - Interaction
 - Implementation
- Fidelity:
 - Low: omits details
 - High: closer to finished project
 - Multi-dimensional
 - Breadth: % of features covered
 - Only enough features for certain tasks
 - Depth: degree of functionality
 - Limited choices, canned responses, no error handling



From Nielsen,
Usability Engineering

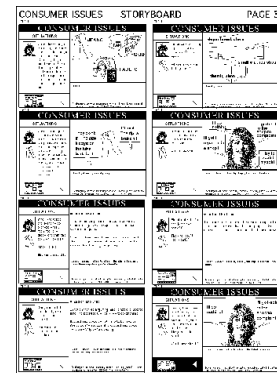
Oct 17, 2011

Sprengle - CSC209

31

Low Fidelity

- Media: Paper
- Examples: storyboard, sketches, flipbook, flow diagram



Oct 17, 2011

Sprengle - CSC209

32

High Fidelity

- Media: Flash, HTML (non-interactive), PowerPoint, Video
- Examples: Mockups, Wizard of Oz

Virtual Peer for
Autistic Children



<http://www.articulab.justinecassell.com/projects/samautism/index.html>

TODO

- Wed: Assignment 8 due
- Wed class: More testing!

Oct 17, 2011

Sprengle - CSC209

34