

Objectives

- Unix File System, Permissions
- Exceptions

Sept 24, 2008

Sprenkle - CS209

1

Review

- Compare and contrast abstract classes and interfaces
- When should a class be abstract?
- When should you create/use an interface?
- What is the keyword for defining your class to implement an interface?

Sept 24, 2008

Sprenkle - CS209

2

Notes on Assignment 6

- In Eclipse ...
- When warnings are ok
 - Towards no warnings
- Other useful Eclipse features
 - Format: Control-Shift-F
 - No indentation/formatting issues

Sept 24, 2008

Sprenkle - CS209

3

Analysis of equals methods

```
public boolean equals(Object o){
    if(((Birthday) o).getDate() != this.getDate())
        return false;

    if( ((Birthday) o).getMonth() != this.getMonth())
        return false;
    return true;
}
```

```
public boolean equals(Object o) {
    Birthday other = (Birthday) o;
    if (this.month == other.month && this.day ==
        other.day)
        return true;
    else
        return false;
}
```

Sept 24, 2008

Sprenkle - CS209

4

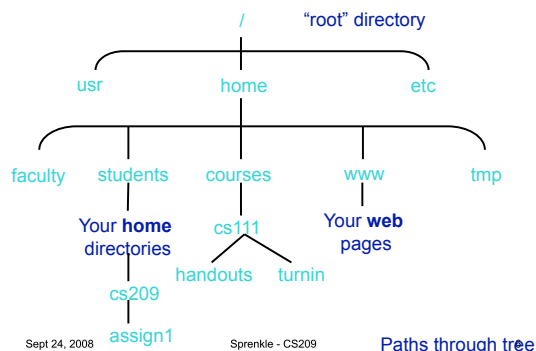
Game Demonstration

Sept 24, 2008

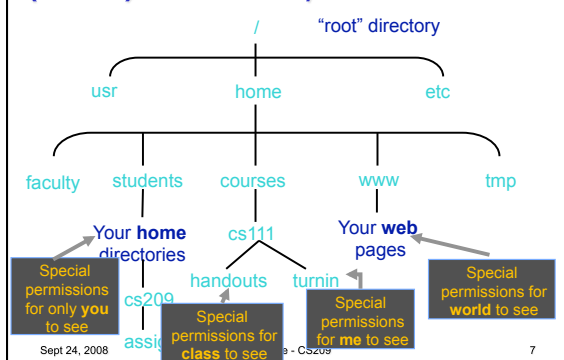
Sprenkle - CS209

5

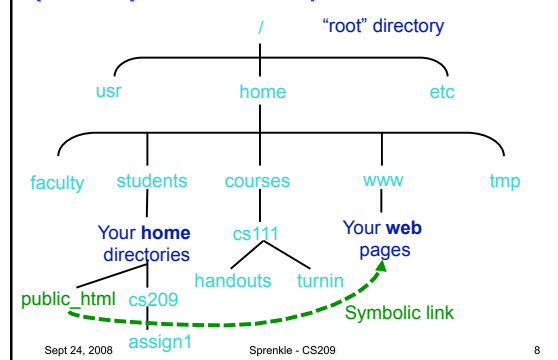
(Partial) Linux File System



(Partial) Linux File System



(Partial) Linux File System



Unix File Structure/Permissions

From your home directory

```
> ls -l
public_html may be in different color than most entries

> ls public_html          Note: no / at end

> ls -l public_html

> ls -l /home/courses/cs209/
```

Sept 24, 2008

Sprengle - CS209

9

Unix Permissions

- Categories: **owner**, **group**, **others**
- Permissions: read, write, execute

```
[sprengle@hopper courses]$ ls -l /home/courses/cs209/handouts/
total 16
drwxr-x--- 3 sprengle cs209 4096 2008-09-16 22:09 ./
drwxr-x--- 5 sprengle cs209 4096 2008-09-03 11:20 ../
drwxr-xr-x 2 sprengle faculty 4096 2008-09-17 10:16 assign4/
-rw-r--r-- 1 sprengle faculty 616 2008-09-04 15:56 First.java
```

permissions owner group size date modified file name

Sept 24, 2008

Sprengle - CS209

10

Unix Permissions

- Categories: **owner**, **group**, **others**
- Permissions: read, write, execute

```
[sprengle@hopper courses]$ ls -l /home/courses/cs209/handouts/
total 16
drwxr-x--- 3 sprengle cs209 4096 2008-09-16 22:09 ./
drwxr-x--- 5 sprengle cs209 4096 2008-09-03 11:20 ../
drwxr-xr-x 2 sprengle faculty 4096 2008-09-17 10:16 assign4/
-rw-r--r-- 1 sprengle faculty 616 2008-09-04 15:56 First.java
```

permissions owner group size date modified file name

- What are the permissions on the file First.java?
- In the permissions, how can we distinguish between an executable file and directory?
- What does it mean for a file to be executable?

Sept 24, 2008

Sprengle - CS209

11

Changing Permissions

- **chmod** command
 - Syntax: **chmod [options] <mode> <file(s)>**
- Examples:
 - chmod u+x script.sh**
 - chmod a-w readDir**
 - chmod -R ug+r myDir**
 - Recursive**

Shorthand	Meaning
u	User/owner
g	Group
o	Others
a	All
r	Read permission
w	Write permission
x	eXecutable permission

Sept 24, 2008

Sprengle - CS209

12

Changing Ownership, Group

- To change the owner of a file:
 - `chown <owner> <file(s)>`
 - `chown <owner:group> <file(s)>`
 - `-R` recursive option available
- To change the group of a file
 - `chgrp <group> <file(s)>`
 - `-R` recursive option available

Sept 24, 2008

Sprenkle - CS209

13

Interface Object Variables

- Can use an object variable to refer to an object of any class that implements an interface
- Using this object variable, can *only* access the interface's methods
- For example...

```
Object obj;
...
if (obj instanceof Comparable) {
    Comparable comp = (Comparable) obj;
    boolean res = comp.compareTo(obj2);
}
```

Sept 22, 2008

Sprenkle - CS209

14

Errors

- Programs encounter errors when they run
 - Users may enter data in the wrong form
 - Files may not exist
 - Printers run out of paper in the middle of printing
 - Program code always has bugs
- When an error occurs, a program should do one of two things:
 - Revert to a stable state and continue
 - Allow the user to save data and then exit the program gracefully

Sept 24, 2008

Sprenkle - CS209

15

Java Method Behavior

- Normal/correct case: return specified return type
- Error case: does not return anything, **throws** an Exception
 - **Exception**: object that encapsulates the error information

Sept 24, 2008

Sprenkle - CS209

16

Handling Exceptions

- When Exception is thrown, JVM's **exception-handling mechanism** searches for an **exception handler**
- Exception handler: error recovery code
 - Deals with a particular exception

Sept 24, 2008

Sprenkle - CS209

17

Throwable

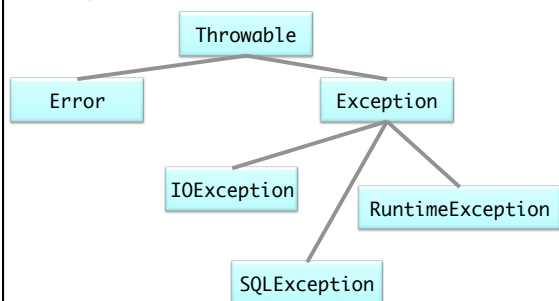
- All exceptions indirectly derive from a class **Throwable**
 - Child classes: **Error** and **Exception**
- Important **Throwable** methods
 - **getMessage()**
 - Detailed message about error
 - **printStackTrace()**
 - Prints out where problem occurred and path to reach that point
 - **getStackTrace()**
 - Get the stack in non-text format

Sept 24, 2008

Sprenkle - CS209

18

Exception Classification



Sept 24, 2008

Sprenkle - CS209

19

Exception Classification: Error

- An internal error
 - JVM-generated in the case of resource exhaustion or an internal problem
 - Out of Memory error (When can that happen in Java?)
- Program's code should not and can not throw an object of this type
- *Unchecked* exception

Sept 24, 2008

Sprenkle - CS209

20

Exception Classifications

- **RuntimeException** something that happens due to a programming error
 - **Unchecked** exception
 - **ArrayOutOfBoundsException**
 - **NullPointerException**
 - **ClassCastException**
- **Checked** exceptions
 - e.g., **IOException**, **SQLException**

} Seen before

Sept 24, 2008

Sprenkle - CS209

21

Exception Classifications

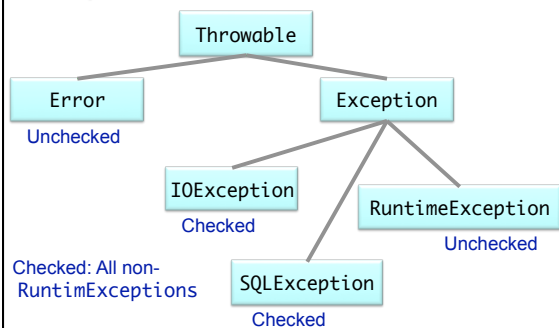
- If something is *programmer's* fault
 - **RuntimeException**
- Otherwise, an **Error** or another **Exception**
- Common: **IOException**
 - Trying to read past the end of a file
 - Trying to open a bad URL
 - File not found
 - ...

Sept 24, 2008

Sprenkle - CS209

22

Exception Classification



Sept 24, 2008

Sprenkle - CS209

23

Checked and Unchecked

- **Unchecked**: any exception that derives from **Error** or **RuntimeException**
 - Programmer does not create/handle
 - Try to make sure that they don't occur
- **Checked**: any other exception, e.g., from **IOException**
 - Programmer needs to create and handle **checked** exceptions

Sept 24, 2008

Sprenkle - CS209

24

Types of Unchecked Exceptions

- Derived from the class `Error`
 - Any line of code can generate because it is internal error
 - Don't worry about what to do if this happens
- Derived from the class `RuntimeException`
 - Indicates a bug in the program
 - Fix the bug

Sept 24, 2008

Sprenkle - CS209

25

Checked Exceptions

- Need to be handled by your program
 - Compiler enforced
- Advertise the exceptions that a particular method throws
 - For each method, tell the compiler:
 - What the method returns
 - What could possibly go wrong
- As an example, `java.io.BufferedReader`

Sept 24, 2008

Sprenkle - CS209

26

BufferedReader Class

- Has `readLine()`
 - Reads a line from a stream, such as a file or network connection
- Header:


```
public String readLine() throws
    IOException
```
- `readLine` can
 - return a `String` (if everything went right)
 - throw an `IOException` (if something went wrong)

Sept 24, 2008

Sprenkle - CS209

27

Programmer-Defined Methods

- Advertise only the **checked** exceptions that your method can throw
 - Your method calls a method that throws a checked exception
 - Your method detects an error in its processing and decides to throw an exception

Sept 24, 2008

Sprenkle - CS209

28

Example: Passing an Exception Up

- If we were to write a method that calls the `readLine()` method of `BufferedReader`:

```
public String readData(BufferedReader in)
    throws IOException {
    String str1;
    str1 = in.readLine();
    return str1;
}
```

Sept 24, 2008

Sprenkle - CS209

29

Example: Passing an Exception Up

```
public String readData(BufferedReader in)
    throws IOException {
    String str1;
    str1 = in.readLine();
    return str1;
}
```

Throws the `IOException`

- `readData()` calls a method that can throw an `IOException`
- `readLine()` will throw this exception to our method
 - Assuming we don't want to deal with exceptions, we throw the exception as well
 - Whoever called `readData` will handle exception

Sept 24, 2008

Sprenkle - CS209

30

Example: Throwing An Exception

- If we have a program that is reading a file byte-by-byte, we know in advance how big this file is
- What do we do if we reach an EOF byte while we should still have data to read?
 - Generate our own exception

Sept 24, 2008

Sprenkle - CS209

31

Example: Throwing An Exception

Expected number of bytes

```
public String readBytes(BufferedReader in, int num_bytes)
    throws EOFException {
    while (true) {
        if (char_in == EOF) {
            if (number_read < num_bytes)
                throw new EOFException();
        }
        // ...
    }
}
```

Sept 24, 2008

Sprenkle - CS209

32

Throwing An Exception

```
if (num_read < num_bytes)
    throw new EOFException();
```

- If we encounter an EOF, we make a new object of class EOFException
 - Class derived from IOException
- After making Exception object, we throw it
 - Method ends at this point
 - Calling method handles exception, which says that encountered an EOF before we should have

Sept 24, 2008

Sprenkle - CS209

33

Factorial Alternatives

```
public static double factorial( int x ) {
    if( x < 0 )
        return 0.0;
    double fact = 1.0;
    while( x > 1 ) {
        fact *= x;
        x--;
    }
    return fact;
}
```

Sept 24, 2008

Sprenkle - CS209

34

Factorial Alternatives

```
public static double factorial( int x ) {
    if( x < 0 )
        throw new IllegalArgumentException("x
must be >= 0");
    double fact = 1.0;
    while( x > 1 ) {
        fact *= x;
        x--;
    }
    return fact;
}
```

What are the pros and cons of these approaches?

Sept 24, 2008

Sprenkle - CS209

35

Assignment 6

- Practice on Abstract classes, interfaces, and packages

Sept 24, 2008

Sprenkle - CS209

36