

## Objectives

- Jar files
- Exceptions
  - Wrap up
  - Why Exceptions?
- Files
- Streams

## JAR FILES

## Jar (Java Archive) Files

- Archives of Java files
- Package code into a neat bundle to distribute
  - Easier, faster to download
  - Easier for others to use
- **jar** command: create, view, and extract Jar files
  - Works similarly to **tar**  
`jar cf myapplication.jar *.class`
- Run it using java  
`java -jar myapplication.jar`

Oct 3, 2016

Sprenkle - CSCI209

3

## Creating Jar Files in Eclipse

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

Oct 3, 2016

Sprenkle - CSCI209

4

## Review

- What are the components of the Java Collections framework?
- For each interface, name one implementation
- What are the benefits of the Java Collections framework?
- What is the preferred way to use Java Collections?
- How do we use exceptions?
- What are the different types of exceptions?

Oct 3, 2016

Sprenkle - CSCI209

5

## Practice

```
public void setBirthday(int month, int day) {  
}
```

- How should we implement this method?
  - Rule of thumb: Handle error checking first

Sept 30, 2016

Sprenkle - CSCI209

6

## CATCHING EXCEPTIONS

Sept 30, 2016

Sprenkle - CSCI209

7

### Try/Catch Block

- The simplest way to catch an exception
- Syntax:

```
try {  
    code;  
    more code;  
}  
catch (ExceptionType e) {  
    error code for ExceptionType;  
}  
catch (ExceptionType2 e) {  
    error code for ExceptionType2;  
}  
...  
...
```

Python equivalent?

Sept 30, 2016

Sprenkle - CSCI209

8

## Try/Catch Block

```
try {
    code;
    more code;
}
catch (ExceptionType e) {
    error code for
    ExceptionType
}
```

- Code in **try** block runs first
- If **try** block completes without an exception, **catch** block(s) are not executed
- If **try** code generates an exception
  - A **catch** block runs
  - Remaining code in **try** block is not executed
- If an exception of a type other than **ExceptionType** is thrown inside **try** block, method exits immediately\*

Sept 30, 2016

Sprenkle - CSCI209

9

## Try/Catch Block

```
try {
    code;
    more code;
}
catch (ExceptionType e) {
    error code for
    ExceptionType
}
catch (ExceptionType2 e) {
    error code
    for ExceptionType2
}
```

- You can have more than one **catch** block
  - To handle > 1 type of exception
- If exception is not of type **ExceptionType1**, falls to **ExceptionType2**, and so forth
  - Run the first matching **catch** block

Can catch any exception with **Exception e** but won't have customized messages

Sept 30, 2016

Sprenkle - CSCI209

10

## Try/Catch Example

```
public void read(BufferedReader in) {
    try {
        boolean done = false;
        while (!done) {
            String line=in.readLine();
            // above could throw IOException!
            if (line == null)
                done = true;
        }
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Prints out stack trace to method call  
that caused the error

Sept 30, 2016

Sprenkle - CSCI209

11

## Try/Catch Example

```
public void read(BufferedReader in) {
    try {
        boolean done = false;
        while (!done) {
            String line=in.readLine();
            // above could throw IOException!
            if (line == null)
                done = true;
        }
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

More precise **catch** may help pinpoint error  
But could result in messier code

Sept 30, 2016

Sprenkle - CSCI209

12

## The finally Block

- Optional: add a **finally** block after all **catch** blocks

- Code in **finally** block **always** runs after code in **try** and/or **catch** blocks

- After **try** block finishes or, if an exception occurs, after the **catch** block finishes

- Allows you to clean up or do maintenance before method ends (one way or the other)

- E.g., closing files or database connections

```
try {
    ...
}
catch (Exception e) {
    ...
}
finally { ←
```

FinallyTest.java

Sept 30, 2016

Sprenkle - CSCI209

13

## Practice: try/catch/finally Blocks

```
try {
    statement1;
    statement2;
}
catch (EOFException e) {
    statement3;
    statement4;
}
finally {
    statement5;
}
```

- Which statements run if:

- Neither **statement1** nor **statement2** throws an exception
  - statement1** throws an **EOFException**
  - statement2** throws an **EOFException**
  - statement1** throws an **IOException**

Sept 30, 2016

Sprenkle - CSCI209

14

## What to do with a Caught Exception?

- Dump the stack after the exception occurs
  - What else can we do?
  
- Generally, two options:
  1. Catch the exception and recover from it
  2. Pass exception up (throw) to whoever called the method

Oct 3, 2016


Sprenkle - CSCI209

15

## Methods and Exceptions Example

- `BufferedReader` has method `readLine()`
  - Reads a line from a *stream*, such as a file or network connection
  
- Method header:
 

Part of Advertising



```
public String readLine() throws IOException
```
  
- Interpreting the header: `readLine` will
  - return a `String` (if everything went right)
  - throw an `IOException` (if something went wrong)

Oct 3, 2016

Sprenkle - CSCI209

16



## Javadoc Guidelines about @throws

- Advertising: in Javadoc, document under what conditions each exception is thrown
  - @throws tag
- Always report if throw **checked** exceptions
- Report any unchecked exceptions that the caller might reasonably want to catch
  - Exception: `NullPointerException`
  - Allows caller to handle (or not)
  - Document exceptions that are independent of the underlying implementation
- Errors should **not** be documented as they are unpredictable

Oct 3, 2016

Sprenkle - CSCI209

17

## Summary: Methods Throwing Exceptions

- API documentation tells you if a method can throw an exception
  - If so, you **must** handle it
- If your method could possibly throw an exception (by generating it or by calling another method that could), advertise it!
  - If you can't handle every error, that's OK...let whoever is calling you worry about it
  - However, they can only handle the error if you advertise the exceptions you can't deal with

Oct 3, 2016

Sprenkle - CSCI209

18

## Looking at Game.java

- Use of try/catch

Oct 3, 2016

Sprenkle - CSCI209

19

## Discussion: Why Checked and Unchecked Exceptions?

- Why do we have exceptions that the compiler doesn't force the programmer to check?
  - Think about examples of unchecked exceptions (`ArrayOutOfBoundsException`, `NullPointerException`, `ClassCastException`) and when those exceptions can occur

Oct 3, 2016

Sprenkle - CSCI209

20

## Programming with Exceptions

- Exception handling is slow
- Use one big `try` block instead of nesting `try-catch` blocks
  - Speeds up Exception Handling
  - Otherwise, code gets too messy
- Don't ignore exceptions (e.g., `catch` block does nothing)
  - Better to pass them along to higher calls

```
try {
}
catch () {
}
try {
}
catch () {
}
```

```
try {
  try {
  }
  catch () {
  }
}
catch () {
}
```

```
try {
  ...
}
catch () {
}
```

Oct 3, 2016

Sprenkle - CSCI209

## Creating Our Own Exception Class

- Try to reuse an existing exception
  - Match in name as well as semantics
- If you cannot find a predefined Java `Exception` class that describes your condition, implement a new `Exception` class!

Oct 3, 2016

Sprenkle - CSCI209

22

## Creating Our Own Exception Class

```
public class FileFormatException extends IOException {
    public FileFormatException() {
        }
        public FileFormatException(String message) {
            super(message);
        }
        // other 2 standard constructors...
    }
}
```

What happens in this constructor implicitly?

Is this a checked or unchecked exception?

- Can now throw exceptions of type **FileFormatException**

Oct 3, 2016

Sprenkle - CSCI209

23

## Discussion: Benefits of Exceptions

- Been talking about details...
- Why does Java have exceptions as part of the language?
- Why does Java add some features that Python doesn't have?

Oct 3, 2016

Sprenkle - CSCI209

24

## Benefits of Exceptions

- Force error checking/handling
  - Otherwise, won't compile
  - Does not guarantee "good" exception handling
- Ease debugging
  - Stack trace
- Separates error-handling code from "regular" code
  - Error code is in catch blocks at end
  - Descriptive messages with exceptions
- Propagate methods up call stack
  - Let whoever "cares" about error handle it
- Group and differentiate error types

Oct 3, 2016

Sprenkle - CSCI209

25

## FILES

Oct 3, 2016

Sprenkle - CSCI209

26

## java.io.File Class

- Represents a file or directory
- Provides functionality such as
  - Storage of the file on the disk
  - Determine if a particular file exists
  - When file was last modified
  - Rename file
  - Remove/delete file
  - ...

Oct 3, 2016

Sprenkle - CSCI209

27

## Making a File Object

- Simplest constructor takes full file name (including path)
  - If don't supply path, Java assumes current directory (.)

```
File f1 = new File("chicken.data");
```

- Creates a `File` *object* representing a file named "chicken.data" in the current directory
- Does **not** create a file with this name on disk

Oct 3, 2016

Sprenkle - CSCI209

28

## Files, Directories, and Useful Methods

- A `File` object can represent a file **or** a directory
  - Directories are special files in most modern operating systems
- Use `isDirectory()` and/or `isFile()` for type of file `File` object represents
- Use `exists()` method
  - Determines if a file exists on the disk

Oct 3, 2016

Sprenkle - CSCI209

29

## More File Constructors

- String for the path, String for filename

```
File f2 = new File(
    "/csdept/local/courses/cs209/handouts",
    "chicken.data");
```

- File for directory, String for filename

```
File dir = new File(
    "/csdept/local/courses/cs209/handouts");
File f4 = new File(dir, "chicken.data");
```

Oct 3, 2016

Sprenkle - CSCI209

30

## “Break” any of Java’s Principles?

Oct 3, 2016

Sprenkle - CSCI209

31

## Not Portable

- Accessing the file system is inherently not portable
  - In Windows, paths are `c:\\dir`
  - In Unix, paths are `/home/courses/dir`
- Relies on underlying file system/operating system to perform actions

Oct 3, 2016

Sprenkle - CSCI209

32



## Handling Portability Issues

- Static fields in `File` class
  - `static separator`
    - Unix: "/"
    - Windows: "\\\"
  - `static pathSeparator`
    - For separating a list of paths
    - Unix: ":"
    - Windows: ";"
- Use relative paths, with separators
- Use configuration files

Why two "\\\"?

Oct 3, 2016

Sprenkle - CSCI209

33

java.util.

**SCANNER**

Oct 3, 2016

Sprenkle - CSCI209

34

## java.util.Scanner

- New(er) class for handling input
    - Since Java 1.5
  - Many constructors
    - Read from file, input stream, string ...
- ```
Scanner sc = new Scanner(System.in);
```
- Many methods
    - nextXXXX (int, long, line)
    - Skipping patterns, matching patterns, etc.

Oct 3, 2016

Sprenkle - CSCI209

35

## Scanners

- Breaks its input into tokens using a delimiter pattern, which matches whitespace
 

What is "delimiter pattern"?  
 What is "whitespace"?
- Converts resulting tokens into values of different types using nextXXX()
- Can change token delimiter from default of whitespace
- Assumes numbers are input as decimal
  - Can specify a different radix

Oct 3, 2016

Sprenkle - CSCI209

36

## Using Scanners

- Use *nextXXX()* to read from it...

```
long tempLong;

// create the scanner for the console
Scanner sc = new Scanner(System.in);

// read in an integer and a String
int i = sc.nextInt();
String restOfLine = sc.nextLine();

// read in a bunch of long integers
while (sc.hasNextLong()) {
    tempLong = sc.nextLong();
}
```

Oct 3, 2016

Sprenkle - CSCI209

37

## Using Scanner

Simplified version of online example

```
public static void main(String[] args) {

    // open the Scanner on the console input, System.in
    Scanner scan = new Scanner(System.in);

    scan.useDelimiter("\n"); // breaks up by lines, useful for
    // console I/O

    System.out.print("Please enter the width of a rectangle: ");
    int width = scan.nextInt();

    System.out.print("Please enter the height of a rectangle: ");
    int length = scan.nextInt();

    System.out
        .println("The area of your square is " + length * width +
            ".");
}
```

ConsoleUsingScannerDemo.java

Oct 3, 2016

Sprenkle - CSCI209

38

## Output

Read in as one token

```
This program calculates the area of a rectangle.

Please enter the width of a rectangle (as an integer):
the number is 1
Incorrect input.
Please enter the width of a rectangle (as an integer):
1 2
Incorrect input.
Please enter the width of a rectangle (as an integer):
2
Please enter the height of a rectangle (as an
integer): 3
The area of your rectangle is 6.
```

Oct 3, 2016

Sprenkle - CSCI209

39

## Scanners & Exceptions

- Scanners do not throw `IOExceptions`!
  - For a simple console program, `main()` does not have to deal with or throw `IOExceptions`
  - Required with `BufferedReader/InputStreamReader` combination
- Throws `InputMismatchException` when token doesn't match pattern for expected type
  - e.g., `nextLong()` called with next token "AAA"
  - `RuntimeException` (no catching required)

How do you prevent such errors?

Oct 3, 2016

Sprenkle - CSCI209

40

## Looking Ahead

- Exam on Friday
  - [Document online](#)
- Assignment 6 due Wednesday before class