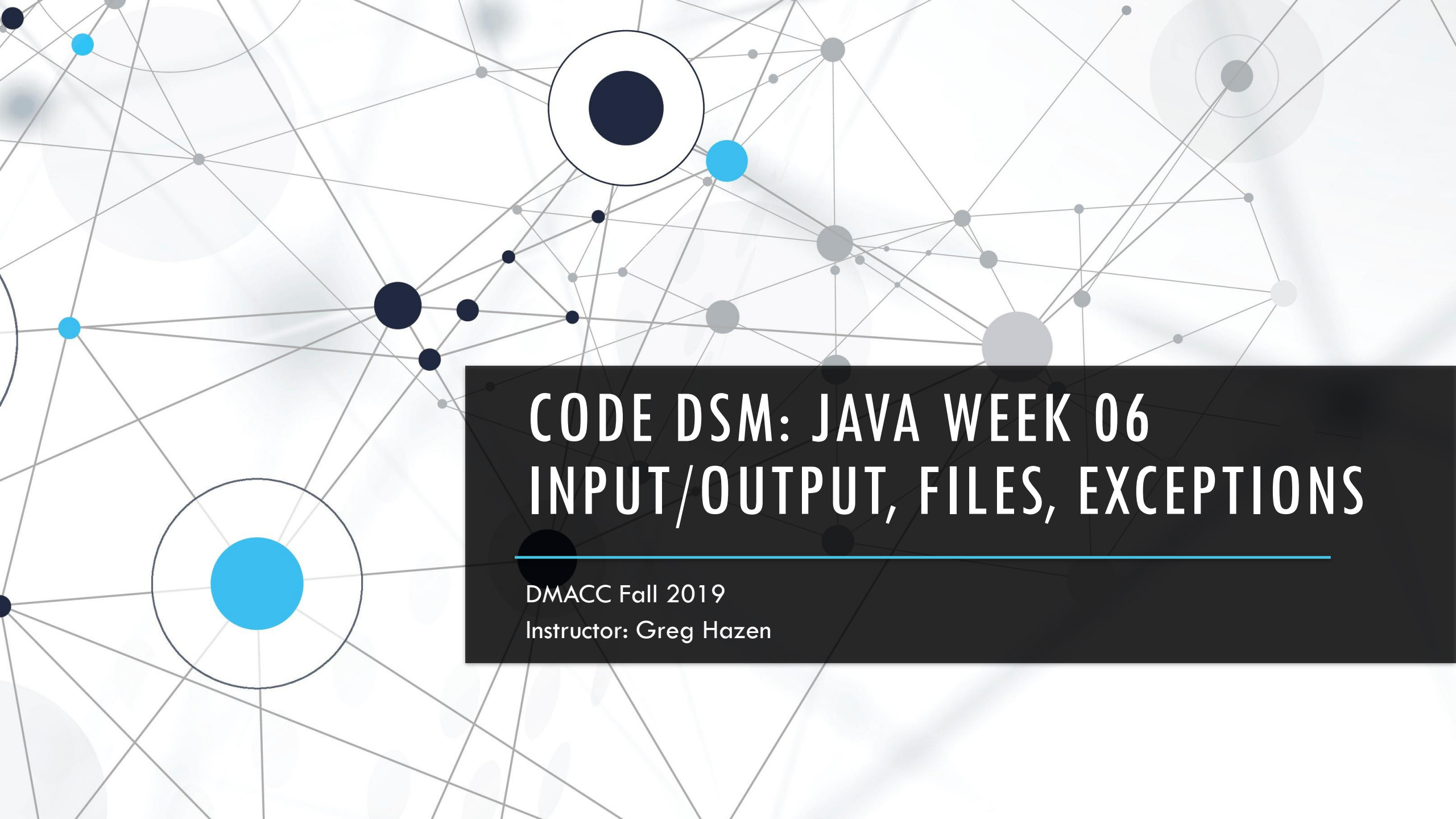

CODE DSM: JAVA WEEK 06 INPUT/OUTPUT, FILES, EXCEPTIONS

DMACC Fall 2019

Instructor: Greg Hazen

AGENDA REVIEW

REVIEW COLLECTIONS

REVIEW WEEK 5 HOMEWORK

AGENDA

EXCEPTIONS

VALIDATION

FILES

COLLECTIONS: ARRAYS

String[] names = new String[3];

- names[0] = “Greg”; //Zero-based index
- names.length;

COLLECTIONS: LISTS

ArrayList<String> names = new ArrayList<>();

- names.add("Greg"); //Dynamically add and remove
- names.get(0); //Zero-based index
- names.remove("Greg");

COLLECTIONS: SETS

HashSet<String> uniqueNames = new HashSet<>();

- names.add("Greg"); //Dynamically add and remove
- names.add("Greg"); //Will only keep one "Greg"
- names.remove("Greg");

COLLECTIONS: MAPS

HashMap<String, Double> menu = new HashMap<>();

- menu.put("Fry", 0.75); //Stored in key/value pairs
- menu.put("Fry", 0.50); //Keys are unique, only keeps last value

REVIEW WEEK 5 ASSIGNMENT

Switch to IntelliJ!

EXCEPTIONS

When an error or some other exceptional condition occurs in a program, an ‘exception’ needs to be ‘thrown’

An exception is a special object of type Exception

- Implements the Throwable interface

Exceptions have specific names that indicate their error

- A message and cause are not required, but help programmers troubleshoot issues

EXCEPTIONS

Exceptions come in three varieties:

- Checked
- Unchecked
- Errors

EXCEPTIONS: CHECKED

Created by a condition outside of the code (file doesn't exist or bad database connection…)

Must handle checked exceptions before compile time

If not handled, it must be added to the method header

- public static String readData(String filename) **throws FileNotFoundException**

EXCEPTIONS: UNCHECKED

Created by the code (array IndexOutOfBoundsException)

Are of type RuntimeException

Are NOT checked at compile time to see if they are handled

If not handled, can **optionally** be added to the method header

- public static String readArray(String[] names) **throws IndexOutOfBoundsException**

EXCEPTIONS: ERROR

Represent unrecoverable conditions that force the program to terminate

Cannot be handled

A common error is OutOfMemoryError

HANDLING EXCEPTIONS

Handle exceptions using one or more try-catch blocks

If and only if an exception occurs will the *catch* block be executed. All subsequent code in the *try* block is skipped.

```
try {
        //do something
} catch(Exception e) {
        //handle exception
}
```

HANDLING EXCEPTIONS

Reporting an exception

- Print the stack trace to System.err
- Print a message or cause
- Continue or end the program

Commonly would send details to a file called a log file

- Many tools are available to instigate log files

HANDLING EXCEPTIONS

It's common not to handle the exception right away

Can add exception to method signature and handle higher up where you have more context, for example

- Send email to customers // Knows context of the email
  - Loop through all customers // Knows context of the current customer
    - Query database for information // Exception Thrown

HANDLING EXCEPTIONS

If there's code that must **always** run, use try-catch-finally

catch block executes if and only if exception is thrown in *try*

finally executes every time, even if there's a return in *try*

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

HANDLING EXCEPTIONS

try-catch-finally is useful to release computer resources

- Closing a file
- Closing a database connection

Can also to try-finally without a catch, but not as common

HANDLING EXCEPTIONS

Let's try it in IntelliJ!

VALIDATE INPUT AND OUTPUT

You've already seen how to use System.in with a Scanner to get user input and print output using System.out

```
Scanner in = new Scanner(System.in);
while (in.hasNext()) {
    System.out.println(in.next());
}
```

VALIDATE INPUT AND OUTPUT

You've also seen converting input from a String to a number using the corresponding object to parse it

- Integer.parseInt(…)
- Double.parseDouble(…)

This will create an exception if you try to parse something that is not a number

VALIDATE INPUT AND OUTPUT

Any time a user is prompted for input, you should validate the input is valid and as expected

- Not doing so leads to security vulnerabilities

We now know how to use parsing to validate input using a try-catch!

```
try {
        Double.parseDouble(…);
} catch(NumberFormatException e) {
        //Invalid Input!
}
```

EXERCISE

Create a main method that asks the user for a menu item and then the price of that item. When the user submits "done", then print out the menu. If the user enters a non-number for the price, print "The price must be a number."

Example without Error:

Food? Burger
Price? 1.5
Food? done
Burger $1.50

Example with Error:

Food? Burger
Price? Fry
The price must be a number.
Price? 1.5

VALIDATE INPUT AND OUTPUT

Similarly, we could have used the methods built into Scanner

```
if(input.hasNextDouble()) {
        price = input.nextDouble();
} else {
        System.out.println("The price must be a number.");
}
```

READING FILES

A text file is a common way to store data on a computer

- Values written in clear text using characters to help divide up the input
- Commonly separate values using New line (\n), Commas or Tabs (\t)

Locating a file used to be highly dependent on the OS

- Broke portability between JVMs
- Java built in OS file path, symbols, and security
- As of Java 7, file path slashes are less sensitive

READING FILES

Files are represented in Java by the 'File' object

- File myFile = new File("a_text_file.txt");

To read a file, create a new File object that uses the OS specific path to the file's location on the computer

The easiest way to read file is to use a File object along with the Scanner object

- Scanner inFile = new Scanner(myFile);

READING FILES

I never get the path right the first time, so here's my way to debug the path

File myFile = new File("./path/to/my_file.txt");

System.out.println(myFile.getAbsolutePath());

READING FILES

You can check if the file exists before trying to read it

```
File myFile = new File("a_text_file.txt");
myFile.exists(); //returns true or false
```

READING FILES

Scanner can read line by line

Use it as an iterator to read all lines in a file

```
File myFile = new File("a_text_file.txt");
Scanner inFile = new Scanner(myFile);
while(inFile.hasNextLine()) {
    String line = inFile.nextLine();
}
```

READING FILES

You must always close a file when you are done using it

- May be unable to access the file elsewhere if it is not properly closed
- To close a file, simply call the close() method on the object accessing the file

```
Scanner inFile = new Scanner(myFile);
….
inFile.close();
```

READING FILES

More advanced usage of files requires more than just the Scanner object

Dealing with files almost always ends up having to use some form of a FileInputStream and a Reader object

- The most common now is a BufferedReader combined with a FileReader

Examples in https://www.baeldung.com/java-read-file

For this class, using the Scanner method described prior is the preferred method of reading a file unless otherwise noted

EXERCISE

- Create a text file in IntelliJ names ‘names.txt’ with several name, one per line
- Create a main method
- Use Scanner to read each name in the file
- Write a message with the following template to System.out: “Hello %s\n”

WRITING FILES

We can use PrintWriter to write lines to a file.

- Behaves identically to System.out

It will create the file if it doesn't already exist

Does need to be closed when finished

Example

```
PrintWriter outFile = new PrintWriter("a_text_file.txt");
outFile.printf("%s, %s\n", lastName, firstName);
outFile.close();
```

WRITING FILES

Again, more advanced usage of writing to files requires more than the PrintWriter

In many cases, you will need to use a FileOutputStream or DataOutputStream

Examples in https://www.baeldung.com/java-write-to-file

For this class, PrintWriter is the preferred way to write to a file unless otherwise noted

EXERCISE

- Reuse your names.txt main method from before
- Now write out your Hello message to another file named ‘hello_names.txt’

ASSIGNMENT

Review assignment for any clarifications

Clone it together

QUIZ

Do NOT use IntelliJ for this week's quiz

SEE YOU NOVEMBER 15TH!

Don’t forget to ask question early in the week