Exceptions

In this lecture we will talk about:

- Exception
  - Checked and Unchecked
  - Try/Catch blocks
  - Throws clause
- More practice reading and writing files

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:

  - Computer science has a number of concentrations.
    - General concentration is the most flexible, and even allows students to double major or minor pretty easily.
    - Software Engineering
    - Computing Systems
    - Human Centered Computing
    - Networks and Security
    - Artificial Intelligence
    - Computer Science Education.
  - Minors:
    - Minor in Computer Science - choose your own adventure minor
    - Minor in Machine Learning - popular with stats/math, and engineering
    - Minor in Bioinformatics - Biology + Computer Science

Exceptions

- Sometimes things do not go as expected
- Good programming, handles *most* (not all) errors - gracefully (via code)

- However, where should this error be handled at?
  - Which class, which "layer" in the program.
  - Example: Accessing a file from the operating system
    - You request the file from the OS
    - The OS either sends you the file, OR
    - It tells you the file isn't there
  - Who should handle that issue?
  - Usually, the application requesting the file!
    - As compared to the SDK/API that helps you access the file

What Are Exceptions?

- Classes / Objects!
  - They contain information about the error that is happening
  - They have a special interface that they all share in common
- What about `try/catch` and `throws`
  - Those are commands that use those objects!
  - And mostly what we deal with

try/catch

- `try{}` - Says "try this block of code"
- `catch(Exception x)` {} - run this block of code if there is an error
- `finally {}` - always run this block of code error or not (often can be omitted, won't be used much in this class)

Let's look at some code!

In [22]:
```
try {
    FileInputStream stream = new FileInputStream("data/notes2.md");
    Scanner scn = new Scanner(stream); // this line will not run if an exception is th
    while(scn.hasNextLine()) {
        System.out.println(scn.nextLine());
    }
}catch(FileNotFoundException ex) { //It extends IOException and makes the most sense i
    System.err.println("File not file! " + ex.getMessage());
}
```

```
File not file! data\notes2.md (The system cannot find the file specified)
```

Class Discussion: Combining it together

Look through the following code.

- At your table, outline what the code does
- This includes thinking about a flow diagram of what method is called when
- Think about the try catch, what happens if a file is not entered correct?
- How is the program handling the potential errors?

In [23]:

```
public class CommentStrip {
    private String filename;
    public static final String OUT_DIR = "out";
    private ArrayList<String> comments =  new ArrayList<>();

    public CommentStrip(String filename) {
        this.filename = filename;
    }
    public ArrayList<String> getComments() { return comments;}
    public String getDirectory() {
        int last = filename.lastIndexOf("/");
        if(last >= 0) return filename.substring(0, last);
        return ""; // assumed else
    }
    public String getFile() {
        int last = filename.lastIndexOf("/");
        if(last >= 0) return filename.substring(last+1);
        return filename;
    }
    public String getSaveFile() {
        return String.format("%s/%s/%s", getDirectory(), OUT_DIR, getFile());
    }
    public void stripAndSave() {
        String cleanedFile = openAndRemove();
        if(!cleanedFile.isEmpty()) saveOut(cleanedFile);
    }
    private String openAndRemove() {
        FileInputStream fn;
        String cleaned = "";
        try {
            fn = new FileInputStream(filename);
        }catch(FileNotFoundException ex) {
            System.err.println("File not found! " + ex.getMessage());
            return ""; // exit the method early
        }
        Scanner sn = new Scanner(fn); // why is this allowed outside the try catch?
        while(sn.hasNextLine()) {
            cleaned += cleanLine(sn.nextLine()) + "\n";
        }
        return cleaned;
    }
    private String cleanLine(String line) {
        int comIndex = line.indexOf("//");
        if(comIndex >= 0) {
            String comment = line.substring(comIndex);
            comments.add(comment);
            line = line.substring(0, comIndex);
        }
        return line;
    }
    private void saveOut(String contents) {
        try {
            PrintWriter writer = new PrintWriter(new FileOutputStream(getSaveFile()));
            writer.print(contents);  // why does this have to be in the try catch?
            writer.close();
        }catch(IOException ex) {
            System.err.println("Error writing file! " + ex.getMessage());
        }
```

```
    }
}
```

In [24]:
```
CommentStrip strip = new CommentStrip("data/Myprogram.java");
strip.stripAndSave();
System.out.println(strip.getComments());
```

```
[// this is the first comment, // in the main program, //another comment, // and a co
mment at the end]
```

Throws

- Sometimes you don't want to handle the exception in that particular block of code.
- You can "throw" it for someone else to handle
- Adding throws at the end of a method signature allows you to continue to toss it!

Hot Potato - Eventually someone needs to catch it!

In [25]:
```
public class CommentStrip {  // rewriting this class to use throws!
    private String filename;
    public static final String OUT_DIR = "out";
    private ArrayList<String> comments =  new ArrayList<>();

    public CommentStrip(String filename) {
        this.filename = filename;
    }
    public ArrayList<String> getComments() { return comments;}
    public String getDirectory() {
        int last = filename.lastIndexOf("/");
        if(last >= 0) return filename.substring(0, last);
        return ""; // assumed else
    }
    public String getFile() {
        int last = filename.lastIndexOf("/");
        if(last >= 0) return filename.substring(last+1);
        return filename;
    }
    public String getSaveFile() {
        return String.format("%s/%s/%s", getDirectory(), OUT_DIR, getFile());
    }
    public void stripAndSave() {
        try {
            String cleanedFile = openAndRemove();
            saveOut(cleanedFile); // no need to check for an non-existent file now
        }catch(FileNotFoundException ex) {
            System.err.println("File not found! " + ex.getMessage());
        }
    }
    private String openAndRemove() throws FileNotFoundException{
        FileInputStream fn;
        String cleaned = "";
        fn = new FileInputStream(filename);
        Scanner sn = new Scanner(fn); // won't run if the file is not found
        while(sn.hasNextLine()) {
            cleaned += cleanLine(sn.nextLine()) + "\n";
        }
        return cleaned;
```

```
    }
    private String cleanLine(String line) {
        int comIndex = line.indexOf("//");
        if(comIndex >= 0) {
            String comment = line.substring(comIndex);
            comments.add(comment);
            line = line.substring(0, comIndex);
        }
        return line;
    }
    private void saveOut(String contents) {
        System.out.println("INFO: only runs if a file is found");
        try {
            PrintWriter writer = new PrintWriter(new FileOutputStream(getSaveFile()));
            writer.print(contents);  // why does this have to be in the try catch?
            writer.close();
        }catch(IOException ex) {
            System.err.println("Error writing file! " + ex.getMessage());
        }
    }
}

CommentStrip strip = new CommentStrip("data/MyprogramDoesNotexist.java");
strip.stripAndSave();
System.out.println(strip.getComments());
```

```
File not found! data\MyprogramDoesNotexist.java (The system cannot find the file spec
ified)
[]
```

Advanced: Creating Your Own Exceptions

- You can create and throw your own exceptions (often called "raise" in other languages)
- In java, you have to **extend** the *Exception* class to do that
  - Ensures certain methods are implements for try/catch/throw/throws
- Won't use much in this class, but worth knowing
- Especially useful if you are developing an SDK/API

In [26]:
```
public class MyCoolException extends Exception {
    public MyCoolException(String msg) {
        super(msg);
    }
    public String getMessage() {
        return "SUPER COOL: " + super.getMessage();
    }
}
public static void doSomething(boolean type) throws MyCoolException,Exception { // not
    if(type) throw new MyCoolException("This is a personal message");
    throw new Exception("General exceptions can have messages");
}
public static void test(boolean type) {
    try {
        doSomething(type);
    }catch(MyCoolException ex) {
        System.err.println(ex.getMessage());
    }catch(Exception ey) { // chained catch statements
        System.err.println(ey.getMessage());
```

```
    }
}
```

In [27]:
```
test(true);
test(false);
```

```
SUPER COOL: This is a personal message
General exceptions can have messages
```

Exception Types

- Exceptions fall into two categories checked and unchecked

Checked Exceptions

- FileNotFoundException
- IOException
- Any exception you create

The program won't compile unless you "handle" the exception in some way.

- try/catch
- throws

They assume the situation is serious (and common) enough, that you will need to handle it. You also can't avoid them with good programming habits, as they are often things you can't control (interaction with the OS).

Unchecked Exceptions

A few common ones:

- NullPointerException
  - What happens when you declare an object, but don't create the object.
- IndexOutOfBoundsException
  - Happens when you try to access a location that does not exist on a sequence (String, Lists, Arrays)
- NumberFormatException
  - Happens when you try to make a number for something that is not a number
    - `int myInt = Integer.parseInt("10,")` (Common with OB1 errors)
- ArithmeticException
  - Happens when you divide by 0 or other illegal operations

All these exceptions can be **avoided** with good programming. As such, Java doesn't require try/catch, but you **can** use a try catch if you want.

> Discussion:
> Given the following code, what is printed.

In [28]:
```
public static void exceptionTester(String input, int loc) {
    try {
        System.out.print(input.length());
        System.out.print(input.charAt(loc));
        System.out.print(input.length() / loc);
        System.out.print(Integer.parseInt(input) * loc);
    }catch(NullPointerException ex) {
        System.err.println("Null Pointer!");
    }catch(IndexOutOfBoundsException ey) {
        System.err.println("Index out of bounds! " + ey.getMessage());
    }catch(ArithmeticException ez) {
        System.err.println("Divide by 0!");
    }catch(Exception e) { // admittedly, this is often considered bad style
        System.err.print("catches all exceptions that aren't first caught:: ");
        System.err.println(e.getClass().toString() + " " + e.getMessage());
    }

}
```

In [29]:
```
exceptionTester(null, 0);
exceptionTester("2022", 4);
exceptionTester("2022", 0);
exceptionTester("y2022", 1);
```

```
Null Pointer!
4
Index out of bounds! String index out of range: 4
42
Divide by 0!
525
catches all exceptions that aren't first caught:: class java.lang.NumberFormatExcepti
on For input string: "y2022"
```

In Class Practice File Reading and Writing

For the rest of this class you will experiment with reading CSV files.

- A Comma Separated Value (CSV) file is a common way for to store data.
- It is essentially a spreadsheet/excel file!

Your task will be to download a CSV file (any CSV file), and then

1. Write a program to read the lines in the file
2. Modify the program to read only a couple of the items in each line
3. Have the program write out a program that only writes certain cells to the file

This is intentionally open ended, as the goal is simply to 'play' with file reading and writing. At the table, take things in steps, figure out and discuss what you know, and continue to build up to *something* until the end of the class session.

Here is a list of CSV files CORGIS: The collection of really great, interesting, situated datasets

- Note: some datasets are harder than others, you should avoid ones with quotation marks in them!