COMP 202 – Week 12

- We can now further explore two related topics: exceptions and input / output streams

- This week we focus on exceptions:
  - the try-catch statement
  - exception propagation
  - exception hierarchy
  - creating and throwing exceptions
  - I/O streams
  - reading and writing text files

Exceptions

- An *exception* is an object that describes an unusual or erroneous situation

- Exceptions are *thrown* by a program, and may be *caught* and *handled* by another part of the program

- A program can therefore be separated into a normal execution flow and an *exception execution flow*

- An *error* is also represented as an object in Java, but usually represents a unrecoverable situation and should not be caught

Exception Handling

- A program can deal with an exception in one of three ways:
  - ignore it
  - handle it where it occurs
  - handle it an another place in the program
- The manner in which an exception is processed is an important design consideration

Exception Handling

- If an exception is ignored by the program, the program will terminate and produce an appropriate message
- The message includes a *call stack trace* that indicates on which line the exception occurred
- The call stack trace also shows the method call trail that lead to the execution of the offending line
- See `Zero.java`
- See `Zero2.java`

The `try` Statement

- To process an exception when it occurs, the line that throws the exception is executed within a *try block*
- A try block is followed by one or more *catch* clauses, which contain code to process an exception
- Each catch clause has an associated exception type
- When an exception occurs, processing continues at the first catch clause that matches the exception type
- See `ProductCodes.java`

The `finally` Clause

- A try statement can have an optional clause designated by the reserved word `finally`
- If no exception is generated, the statements in the finally clause are executed after the statements in the try block complete
- Also, if an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause complete

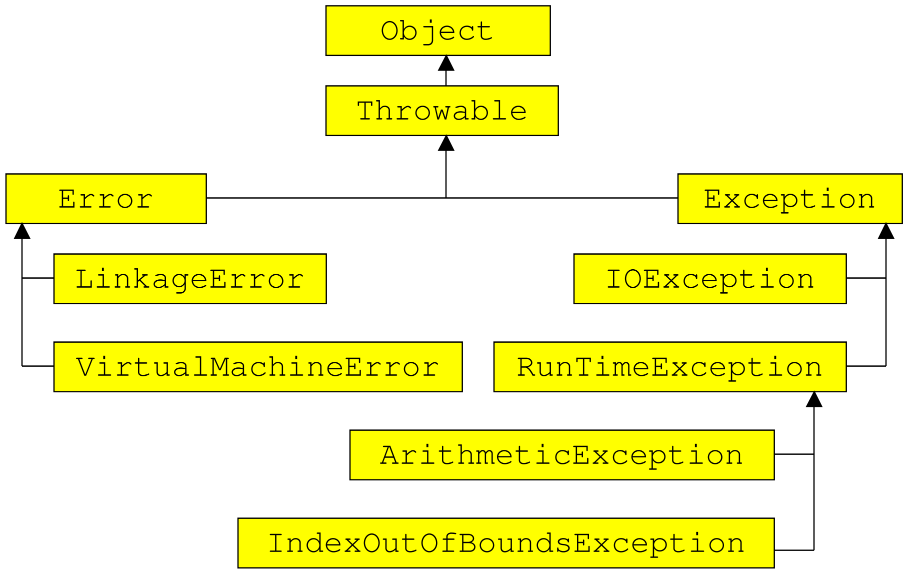
Exception Propagation

- If it is not appropriate to handle the exception where it occurs, it can be handled at a higher level
- Exceptions *propagate* up through the method calling hierarchy until they are caught and handled or until they reach the outermost level
- A try block that contains a call to a method in which an exception is thrown can be used to catch that exception
- See `WildernessIndex.java`
- See `WorldZoom.java`

The `throw` Statement

- A programmer can define an exception by extending the appropriate class
- Exceptions are thrown using the `throw` statement
- See `Ball.java`
- See `BaseBall.java`
- See `FootBall.java`
- See `LetsPlayCatch.java`
- Usually a throw statement is nested inside an if statement that evaluates the condition to see if the exception should be thrown

Part of the Throwable class hierarchy



Checked Exceptions

- An exception is either *checked* or *unchecked*
- A checked exception can only be thrown within a try block or within a method that is designated to throw that exception
- The compiler will complain if a checked exception is not handled appropriately
- An unchecked exception does not require explicit handling, though it could be processed that way

- See `Professor.java`
- See `PredatorsAreNotEatenException.java`
- See `YouShouldNot…Exception.java`
- See `WhoYouShouldAndShouldNotEat.java`

I/O Streams

- A stream is a sequence of bytes that flow from a source to a destination
- In a program, we read information from an input stream and write information to an output stream
- A program can manage multiple streams at a time
- The `java.io` package contains many classes that allow us to define various streams with specific characteristics

I/O Stream Categories

- The classes in the I/O package divide input and output streams into other categories
- An I/O stream is either a
  - *character stream*, which deals with text data
  - *byte stream*, which deal with byte data
- An I/O stream is also either a
  - *data stream*, which acts as either a source or destination
  - *processing stream*, which alters or manages information in the stream

Standard I/O

- There are three standard I/O streams:
  - *standard input* – defined by `System.in`
  - *standard output* – defined by `System.out`
  - *standard error* – defined by `System.err`
- We use `System.out` when we execute `println` statements
- `System.in` is declared to be a generic `InputStream` reference, and therefore usually must be mapped to a more useful stream with specific characteristics

The Standard Input Stream

- We've used the standard input stream to create a `Scanner` object to process input read interactively from the user:

```
Scanner scan =  new Scanner (System.in);
```

- The `Scanner` object converts bytes from the stream into characters, and provides various methods to access those characters (by line, by word, by type, etc.)

Text Files

- **Information can be read from and written to text files by declaring and using the correct I/O streams**

- **We can read from a file using the file as the input stream for our scanner object:**

```
Scanner scan = new Scanner (new File("test.txt"));
```

- **We can write to a text file using the `FileWriter` class in the `java.io` package**

- See `MyWorld.java`
- See `Country.java`