Exceptions and File I/O

3 Feb 2011
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Exceptions for error handling

- Recall that exceptions are used for indicating runtime errors
  - Incorrect user input or parameters
  - No memory, disk space, permissions, etc.
- When an exception is thrown:
  - Execution of the current block is terminated
  - Search for the nearest exception handler
    - **Search enclosing blocks ({})**
    - **Search down the call-stack (what code invoked the current function)**

Exceptions in Java

- In Java, use try-throw-catch
- Make an instance of java.lang.Exception, throw it:
  - The Exception constructor can take a string param: this is stored with the exception

```
try {
    if (ID <= 0)
        throw new Exception("Invalid ID!");
} catch (Exception e) {
    ...
}
```

- Can have several catch blocks, for different kinds of exceptions (first matching one is used)

The caught exception object

- **} catch (Exception e) { ...**

- A reference to the caught exception object is in e
  - Can use this to unpack auxiliary data
- Get the auxiliary data with .getMessage() method on the caught exception object inside the handler:
  - **System.out.println(e.getMessage());**

Custom Exception classes

- Create your own type of exceptions:

```
public class StudentError extends Exception
```

- Need at least 2 constructors: no arg, 1 string arg
  - Pass the string msg up to superclass constr.:

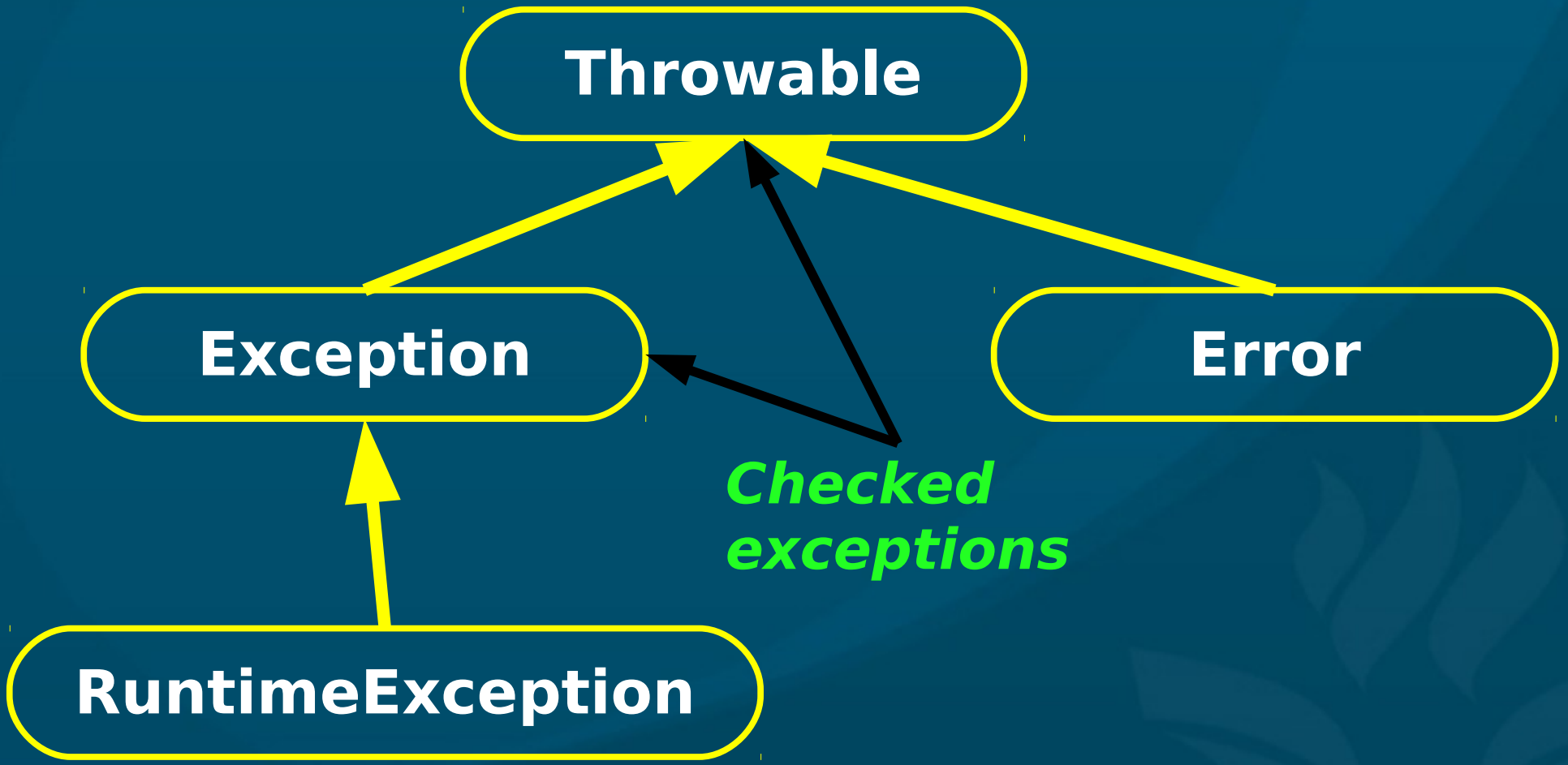
```
public StudentError( String msg ) { super(msg); }
public StudentError()
    { super("Error with student!"); }
```

- Can also add your own auxiliary data (attributes) and constructors, set/get methods, etc.
  - **int studentID;**

The catch-or-declare rule

- A method may encounter exceptions:
  - Directly thrown: throw new StudentError(...)
  - Or thrown by functions it calls: nextInt()
- For checked exceptions, the method must either:
  - Catch the exception and handle it, or
  - Declare that this method may raise an exception, and "pass the buck":
    - **public void setID(int ID) throws StudentError {**
      ...

Exception class hierarchy



Exceptions raised by Scanner

- Using Scanner to read console input:
  - **import java.util.Scanner;**
  - **Scanner kbd = new Scanner(System.in);**
- Expecting an integer:
  - **int num = kbd.nextInt();**
- If Scanner can't convert the input to the desired type, it raises an InputMismatchException
- This can be caught, so you can try again

java.io classes

- Object holding pathname information: File
- Formatted text I/O:
  - Scanner, PrintWriter
- Byte-based streams:
  - FileInputStream, FileOutputStream
- Object-based I/O (Serializable):
  - ObjectInputStream, ObjectOutputStream
- Standard streams:
  - System.in (an InputStream),
    System.out, System.err (both PrintStreams)

File methods

- File is essentially a wrapper around a filename string. Constructor:
  - **File oFile = new File(“output.txt”);**
- Check if exists, can read/write:
  - **if (oFile.exists() && oFile.canRead())**
- Check file type:
  - **If (oFile.isFile() || oFile.isDirectory())**
- Get parent directory:
  - **oFile.getParent()**
- Get just the filename: oFile.getName()

Formatted text stream I/O

- java.io.PrintWriter: output formatted text

```
PrintWriter output =
        new PrintWriter( oFile );
output.println( “Hello, World!” );
```

  - Methods as with System.out

- java.util.Scanner: read text from stream

```
Scanner input =
        new Scanner( new File( “in.txt” ) );
        // or: new Scanner( System.in );
id = input.nextInt();
```

- Remember to close() when you're done

File I/O exceptions

- An instance of the class FileNotFoundException is raised if the file cannot be opened:

```
try {
    out = new PrintStream( “out.txt” );
} catch ( FileNotFoundException e ) {
    System.err.println( “No write permissions!” );
}
```

- Scanner raises:
  - InputMismatchException if wrong type, or
  - NoSuchElementException if input is exhausted.
- EOFException when the end of file is reached
- These are all subclasses of IOException

Object-based I/O

- Use FileInputStream / FileOutputStream to open a file for binary I/O

```
fos = new FileOutputStream( “output.db” )
```

- Wrap the stream in an ObjectInputStream / ObjectOutputStream to use object serialization

```
oos = new ObjectOutputStream( fos );
```

- Use readObject/writeObject to do the I/O:

```
oos.writeObject( myobj );
```

  - readObject() returns a generic Object, so need to cast it back to the original type:

```
myobj = (MyObj) ios.readObject();
```

Serializable objects

- Serialization is converting an object to a representation that can be written to a stream
- The Serializable interface is a tag:
  - Interface with no methods
  - Used to identify what objects are serializable
- Primitive types are serializable
- Arrays of serializable objects are serializable
- A class can be tagged as serializable if all its non-transient instance variables are serializable
    - Vars declared transient are skipped in serialization

Customizing serialization

- Serializable objects: just tag as Serializable
  - all the work of reading/writing is done for you!
- You may override writeObject() and readObject():
  - Specify your own format to use in writing out
    - e.g., use writeInt() etc.
  - Default functionality is in defaultWriteObject()
- See CustomDataExample.java

Summary of I/O classes

- Formatted text I/O:
  - Create a File object (pathname)
  - Write: create a PrintWriter, call .print()
  - Read: create a Scanner, call .next*()
- Object-based I/O:
  - Create a File object (pathname)
  - Write: create a FileOutputStream
    - Create ObjectOutputStream: .writeObject()
  - Read: create a FileInputStream
    - Create ObjectInputStream: .readObject()

Random-access files

- Sequential files are hard to modify in-place
  - Must erase and rewrite entire file
- Random-access files:
  - file = new RandomAccessFile(“user.db”, “rw”);
  - Overwrites existing bytes (can append to end)
- Can be used in place of FileInputStream / FileOutputStream, e.g., to do object-based I/O
- File position pointer:
  - file.seek(num_bytes);
  - Seek to position relative to start