CSE 143 Java

Exceptions

Exception Handling

- Idea: exceptions can represent unusual events (as well as errors) that client code could handle
  - Finite table is full; can't add new element
  - Attempt to open a file failed
- **Problem**: the object that detects the error doesn't know how to handle it (and probably shouldn't)
- **Problem**: the client code could handle the error, but isn't in a position to detect it
- **Solution**: object detecting an error **throws** an exception; client code **catches** and handles it

try-catch

```
try {
      somethingThatMightBlowUp( );
} catch (Exception e) {
      recovery code – e, the exception object, is a "parameter"
}
```

- ➢Execute try block
- ➢If an exception is thrown, catch block can either process the exception, re-throw it, or throw another exception
- ➢Thrown exceptions terminate throwing method and all methods that called it, until reaching a method that catches the exception (has a catch block whose type matches the exception)
- ➢If there is no try/catch, terminate the thread (possibly the program)

try-catch

- Can have several catch blocks

```
try {
      attemptToReadFile( );
} catch (FileNotFoundException e) {
      …
} catch (IOException e) {
      …
} catch (Exception e) {
      …
}
```

- **Semantics: try to match exception parameters in order until one matches**
- **Need to go from more specific to more general (why?)**
- **If no match – exception propagates (gets thrown) to calling method**
- **See example**

Exceptions as Part of Method Specifications

Generally a method ***must*** either handle an exception or declare that it can potentially throw it

```
void readSomeStuff( ) {
      try {
         readIt( );                    // potentially throws an IOException
      catch (IOException e) {
         handle
      }
```

or

```
void readSomeStuff( ) throws IOException {
      readIt( );
}
```

Checked vs Unchecked Exceptions

Unchecked: RuntimeException and all subclasses (e.g. NullPointerException)

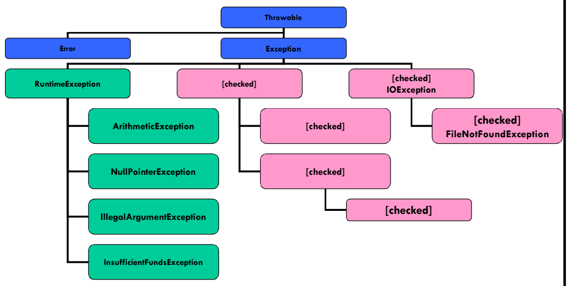
Checked: all others (e.g. IOException)

Rule: a method ***must*** either catch all checked exceptions it might encounter, or declare that it might throw them

No need to declare anything about unchecked exceptions

At least include an @throws in the JavaDocs for ones specifically thrown

Throwable/Exception Hierarchy



finally

```
try {
    …
} catch (SomeException e) {
    …
} catch (SomeOtherException e) {
    …
} finally {
    …
}
```

- Semantics: the finally block is *always* executed, regardless of whether an exception is thrown, or whether we catch an exception or not
- Useful to guarantee execution of cleanup code no matter what