Introduction to Java

Handout-3a

Exceptions

- The purpose of exceptions
- How to cause an exception (implicitely or explicitly)
- How to handle (“catch”) an exception within the method where it occurs
- Handling groups of related exceptions
- How to handle exceptions if not handled in the method where it was thrown

Exceptions (ii)

- How and why methods declare the exceptions that can propagate out of them
- Other

Exceptions (iii)

- Exceptions change the flow of control when some important or unexpected event, usually an error, occurs
  - Cope with error or die gracefully

Exceptions (iv)

Note	Java	Other languages
An error condition that happens at run-time	Exception	Exception
Causing an exception to occur	Throwing	Raising
Capturing an exception that has just occurred and executing statement to resolve it	Catching	Handling
The block that does this	Catch clause	Handler
The sequence of method calls that brought control to the point where the exception happened	Stack trace	Call chain

Exceptions (v)

- Explicitly: use the keyword *throw*
- Implicitly: carry out some invalid or illegal operation
- If provided, control is transferred to section of code that handles exception
  - Can be in same method or caller method
  - If no catch clause found anywhere in the call chain, then program exits

Exceptions (vi)

- The general form of throw statement

`throw` *`ExceptionObject`*

- The *ExceptionObject* is an object of a class that extends the class java.lang.Exception

Exceptions (vii)

Ex:

```
class Melon {
  public static void main(String[] a) {
    int i=1, j=0, k;

    k = i/j;  // Division-by-zero
              // exception
}
```

Exceptions (viii)

- All exceptions are run-time events
  - Run-time library code
    - Irrecoverable (e.g. NullPointerException, SecurityException, ArrayIndexOutOfBoundsException)
    - You don't have to make provisions to catch
  - User defined
    - Less severe, can recover sometimes (e.g. file not found, can prompt user for new file name)
    - You must provide code to handle

Exceptions (ix)

- User defined

Ex:

```
class OutOfGas extends Exception {}

class Car {
  …
  if ( fuel < 0.1 ) throw new OutOfGas();
}
```

Exceptions (x)

- Any method that throws a user_defined exception must either catch or declare it as part of the method interface
- Exceptions *don't reduce* the amount of work needed to handle errors. They just provide a well-localized place to collect and process errors

Exceptions (xi)

- Handling exceptions within the method where it's thrown

```
try block // There must at least one (or both)
          // of the choices below

[ catch (arg) block ] // Zero or many of these
[ finally block ] // Zero or one of these
                  // If present it will be always
                  // executed
```

Exceptions (xii)

- A handler can can catch several related exceptions if the exception objects have the same superclass

Exceptions (xiii)

```
class Grumpy extends Exception {}
class TooHot extends Grumpy {}
class TooTired extends Grumpy {}
class TooCold extends Grumpy {}

try {
  if ( temp > 75 ) throw (new TooHot());
  if ( sleep < 8 ) throw (new TooTired());
}
catch (Grumpy g) {
  if ( g instanceOf TooHot )
    { System.out.println("caught too hot"); return }
  if (g instanceOF TooTired )
    {System.out.println("caught too tired"); return }
}
```

Exceptions (xiv)

- Exception propagation
  - If none of the catch clauses match the exception, then the *finally* clause is executed (if one exists)
  - The flow of control abruptly leaves the the method and a premature return is done to the method that called. If that call was in the scope of a try statement, then it looks for a matching exception
  - This continues until a matching exception block is found or until the top of the call chain is found (when execution ceases with a message)

Exceptions (xv)

- Methods must either catch the exceptions that it throws or declare it
- This is to let know anyone who writes a call to that method, that an exception may come back instead of the normal return

```
modifiersAndReturnType methodName (params) throws e1, e2 {}

Ex:
byte readByte() throws IOException;
```

Exceptions (xvi)

```
class OutOfGas extends Exception {
  OutOfGas(String s) { super(s); }
}
…
try {
  if (j,1) throw new OutOfGas(“try the gas tank”);
}
Catch (outOfGas o) {
  System.out.println(o.getMessage());
}
…
// At run-time will print “try the gas tank”
```