Exception Handling

- Exceptions are error conditions -- not bugs
  - a bug is a defect in the code
  - an error condition is a predictable failure that occurs under certain circumstances
- Exceptions are the way a program can handle gracefully an error that would cause a crash.

Exception Handling

- The Java language provides many exception handlers for programmer errors
  - Java runtime system detects an exceptional event
  - an Exception object is constructed containing info about exception type
  - runtime system throws an exception
  - runtime system looks for code to handle the exception
  - if no handler found, program terminates

Exceptions Classes Hierarchy

- Object
  - Throwable
    - Exception
      - RuntimeException
      - ClassNotFoundException
      - IOException
      - FileNotFoundException
      - ...EOFException...
      - ArithmeticException
      - NumberFormatException
      - ArrayIndexOutOfBoundsException
      - NullPointerExpection
      - Handling exceptions in extending classes is optional
      - FileNotFoundException
      - IOException
      - EOFException
      - ClassNotFoundException
      - RuntimeException
      - NumberFormatException
      - ArrayIndexOutOfBoundsException
      - NullPointerException
      - Handling exceptions in extending classes is optional
Throwing Exceptions

• If a method needs to be able to throw an exception, it has to
  – declare the exception(s) thrown in the method signature
  – include a throw-statement in the method

Ref: http://tutorials.jenkov.com/java-exception-handling/basic-try-catch-finally.html

Writing a Method That Throws an Exception

```java
public void deposit(double anAmount) throws IllegalArgumentException {
    if (anAmount <= 0.0) throw new IllegalArgumentException();
    balance = balance + anAmount;
}
```

The keyword ‘throw’ must be followed by an instance of a class that extends `Throwable`.

Because `Exception` extends `Throwable`, an instance of any `Exception` class can be thrown.

Can throw any type of exception from your code, as long as your method signature declares it.

Ref: http://tutorials.jenkov.com/java-exception-handling/basic-try-catch-finally.html

Methods that throw exceptions

• Exceptions are regular Java classes that extend `java.lang.Exception` or any of the other built-in exception classes.
• If a method declares that it throws an exception A, then it is also legal to throw subclasses of A.

Ref: http://tutorials.jenkov.com/java-exception-handling/basic-try-catch-finally.html

Catching Exceptions

• If a method calls another method that throws checked exceptions, the calling method is forced to either pass the exception on, or catch it.
• Catching the exception is done using a try-catch block.

Ref: http://tutorials.jenkov.com/java-exception-handling/basic-try-catch-finally.html
Exception Handling

- Example: Java throws an exception when an invalid number is entered

```java
public class HandleException {
    public static void main ( String[] args ) {
        Scanner in = new Scanner(System.in);
        System.out.println ( "Enter your age: ");
        int num = in.nextInt();
        System.out.println("you entered " + num);
    }
}
```

Exception Handling by JVM

- Dialogue when number entered is not valid:

```
Enter your age: 30
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at ioDemos.IODemo.getAge(IODemo.java:21)
    at ioDemos.IODemo.main(IODemo.java:13)
```

- Java displays a stack of method calls – the "call stack"; main is the bottom of the stack
- the exception occurred in the method on the top

Exception Handling By Programmer

```java
public class HandleException {
    public static void main ( String[] args ) {
        Scanner in = new Scanner(System.in);
        System.out.println ( "Enter your age: ");
        int age=0;
        try {
            int age = in.nextInt();
        } catch (InputMismatchException e) {
            System.out.println(age + " is not a valid number" );
            System.out.println("Setting age to -1");
            age = -1;
        } System.out.println( age + " stored in age as " + age );
    }
}
```

Runtime Exceptions

- You must know when a method might throw an exception and what type of exception it throws.
- Read the documentation to find out! For example:

```
*** From Double class:
* Return a floating-point number represented by the String
* argument; If numberAsString does not represent a valid
* number, this method will throw a number format
* exception.
*/
public static double parseDouble(string numberAsString) throws NumberFormatException
```
Scanner class exceptions

- int nextInt()
  - If the next token is not an integer, InputMismatchException is thrown.
- long nextLong()
  - If the next token is not an integer, InputMismatchException is thrown.
- float nextFloat()
  - If the next token is not a float or is out of range, InputMismatchException is thrown.
- double nextDouble()
  - If the next token is not a float or is out of range, InputMismatchException is thrown.
- String next()
  - Finds and returns the next complete token from this scanner and returns it as a string; a token is usually ended by whitespace such as a blank or line break. If no token exists, NoSuchElementException is thrown.
- String nextLine()
  - Returns the rest of the current line, excluding any line separator at the end.

Handling an I/O Exception

- If an I/O connection fails, Java throws an IOException. This is the method for a URL connection:
  ```java
  public URLConnection() throws IOException
  ```
- Here's the usage:
  ```java
  java.net.URL url = new java.net.URL("http://www.sice.umkc.edu/");
  java.net.URLConnection conn;
  try {
    conn = url.openConnection();
  } catch (java.io.IOException e) {
    //an error has occurred; log an error, print a message or
    //throw another exception
  }
  ```

Checked Exceptions

- A checked exception (e.g. FileNotFoundException) must be caught or forwarded by putting the phrase throws FileNotFoundException on the header of the method in which the instantiation occurs and the header of any method that calls the method in which the instantiation occurs.
Unchecked Exceptions

- The parseDouble method does not catch the exception!!
  - It's Unchecked – doesn’t need to be handled
- If the programmer does not put the method call in a try block and explicitly catch the exception, the program simply terminates (as shown in earlier example).
- many unchecked exception classes extend the RuntimeException class.

try-finally clause

- JVM can exit block of code many ways:
  1. Executing lines without error to end of ‘}’
  2. Encountering ‘break’, ‘continue’ or ‘return’
  3. Jumping to ‘catch’ clause if error occurred
  4. Terminating the thread if no ‘catch’ exists
- Need way to exit gracefully – to express that something happened no matter how a block of code is exited
- Use try-finally clause

To use a try-finally clause:

```java
try {
    // Block of code with multiple exit points
} finally {
    // Block of code that is always executed
    // when the try block is exited, no matter
    // how the try block is exited
}
```


Multiple Catch Statements

- If there are multiple categories of exceptions that you want to catch, each catch is a separate catch block

```java
try {
    //some code
}
catch (NullPointerException e1) {System.out.println("caught null ptr"); }  
catch (IllegalArgumentException e2) {System.out.println("caught illegal arg"); } 
catch (Exception e3) { }  //This is a “catch-all” -- catches any type of exception that can be thrown
finally {
    //this code will be executed even if an exception did not occur
    //commonly used to clean up internal state (like closing a file)
}
```
• If a finally clause is present with a try, its code is executed after all other processing in the try is complete. This happens no matter how completion was achieved -- whether through an exception or through a control flow statement such as return or break.