Examples of Exceptions

- **ArrayIndexOutOfBoundsException**
  ```java
  int a[] = new int[10];
  for( int i=0; i <= a.length ; ++i ) a[i] = i;
  ```

- **NumberFormatException**
  ```java
  String s = "7.3";
  int n = Integer.parseInt( s );
  ```

Runtime Errors

- Errors happened when the program is running, e.g.
  - Access an element outside array boundary
  - Read a file which doesn't exist
  - User input errors
- **Runtime error handling**
  - Error codes
  - Exceptions

Error Codes

```java
// n >= 1
int factorial( int n )
{
    // return an error code
    if( n < 1 ) return -1;
    int fact = 1;
    for( int i=1; i <= n ; ++i ) fact *= i;
    return fact;
}
```

```
int f = factorial(x);
if( f == -1 ) // error handling
{
    System.err.println( "error!" );
    System.exit(1);
}
else // regular code
{
    // do something with f
}
```

Exception - throw

```java
int factorial( int n ) throws LessThanOneException
{
    if( n < 1 ) throw new LessThanOneException();
    int fact = 1;
    for( int i=1; i <= n ; ++i ) fact *= i;
    return fact;
}
```

Exception - try and catch

```java
try // regular code
{
    int f = factorial(n);
    // do something with f
}
catch( LessThanOneException e ) // error handling
{
    System.err.println( "error!" );
    System.exit(1);
}
```
Use Exceptions

- When an error occurs, throw an exception object
- A method which throws an exception should be declared so using throws ExceptionClassName
- Enclose regular code in a try block
- Enclose error handling code in a catch block

Exception Class

- Pre-defined in Java
- All exception classes must inherit from this class
- Important methods
  - Exception()
  - Exception(String msg)
  - String getMessage()
  - void printStackTrace()

User Defined Exception Class

```java
public class LessThanOneException extends Exception {
    public LessThanOneException() { super(); }
    public LessThanOneException(String msg) {
        super(msg);
    }
}
```

Throw Multiple Exceptions

```java
public void foo() throws ExceptionA, ExceptionB, ExceptionC {
    throw new ExceptionA("Bad thing A");
    throw new ExceptionB("Bad Thing B");
    throw new ExceptionC("Bad Thing C");
}
```

Catch Multiple Exceptions

```java
try {
    foo();
} catch (ExceptionA ea) {
    // do something
} catch (ExceptionB eb) {
    // do something
} catch (ExceptionC ec) {
    // do something
} // Or, if we don't need to distinguish exactly which type of exception occurs
```

A More Complex Example

```java
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```
Handling Errors with Error Code

```c
errorCodeType readFile()
errCode = 0;
open file;
if (fileIsOpen())
  denote the length of the file;
finalize file length;
allocate such memory;
if (allocateMemory())
  read the file into memory;
if (readFailed (errCode = -1))
  else (errCode = -2);
else (errCode = -3);
else (errCode = 0);
(errCode = -4);
else (errCode = -5);
return errCode;
```

Handling Errors with Exceptions

```c
readFile()
try
{
on file;
determine bytes;
allocate that much memory;
read the file into memory;
close file;
catch (fileOpenFailed) (do something);
catch (sizeDefinitionalFailed) (do something);
catch (memoryAllocationFailed) (do something);
catch (result failed) (do something);
catch (fileClosedFailed) (do something);
}
```

Advantages of Exceptions

- Separate error handling code with from regular code
- Group error types and error differentiation
- Propagate errors up the call stack