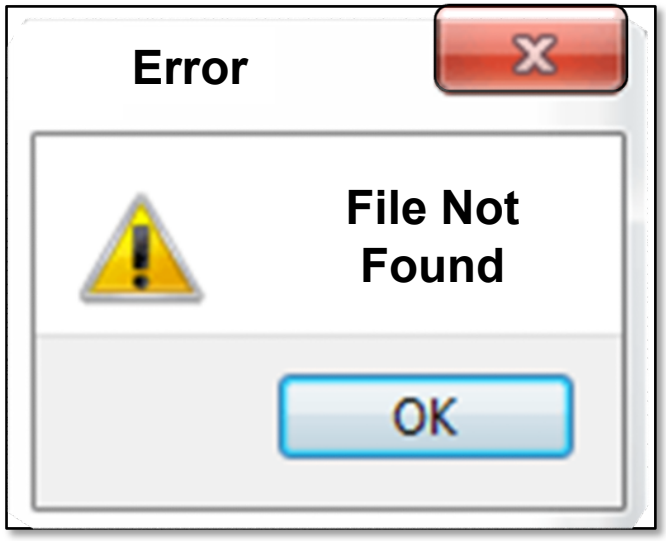
Java Exception Handling

- Apps occasionally encounter problems that just can't be ignored

See en.wikipedia.org/wiki/Murphy's_law

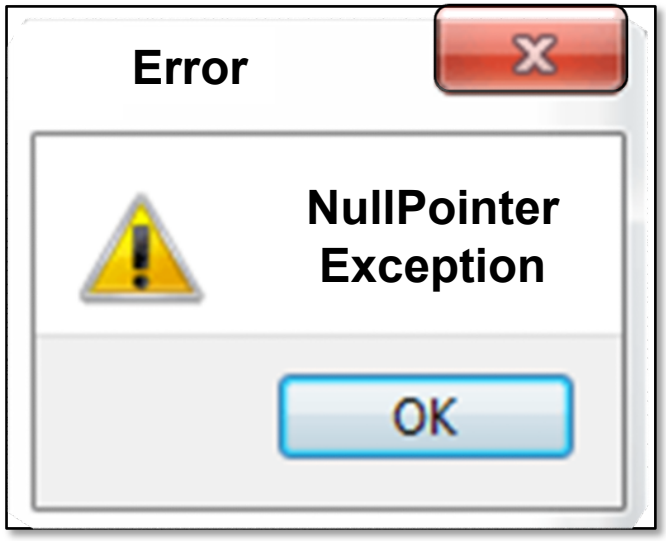
Java Exception Handling

- Apps occasionally encounter problems that just can't be ignored, e.g.
  - Attempting to open a file that's not available



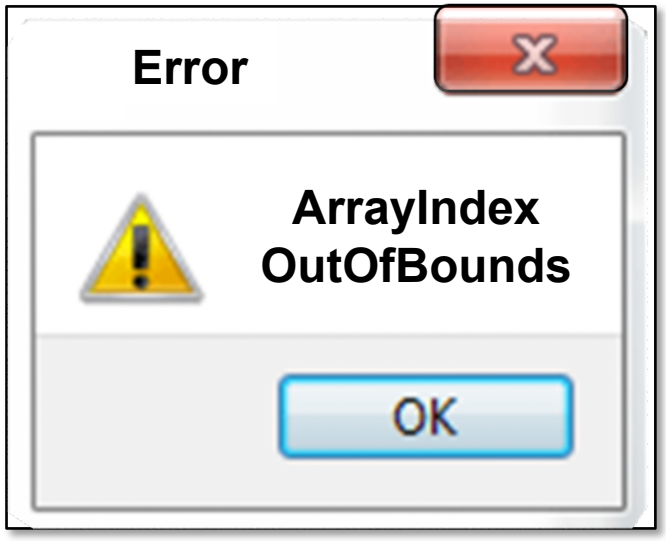
Java Exception Handling

- Apps occasionally encounter problems that just can't be ignored, e.g.
  - Attempting to open a file that's not available
  - Trying to call a method or access a field via a null reference



Java Exception Handling

- Apps occasionally encounter problems that just can't be ignored, e.g.
  - Attempting to open a file that's not available
  - Trying to call a method or access a field via a null reference
  - Indexing before or after the valid range of an array



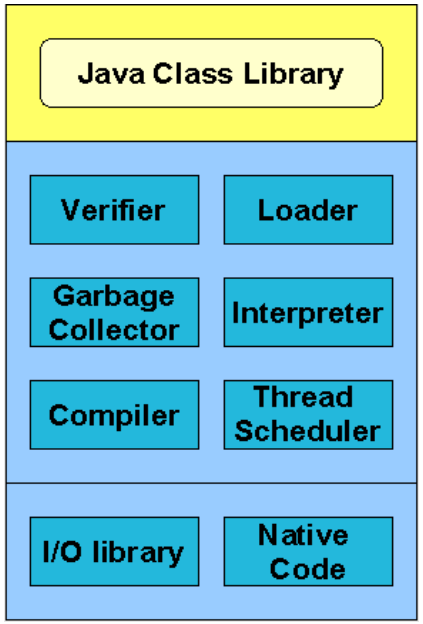
Java Exception Handling

- Exception handling separates control flow paths of "normal" processing versus "anomalous" processing

See en.wikipedia.org/wiki/Exception_handling

Java Exception Handling

- Java exception handling is supported by an extensible hierarchy of classes & a set of virtual machine mechanisms

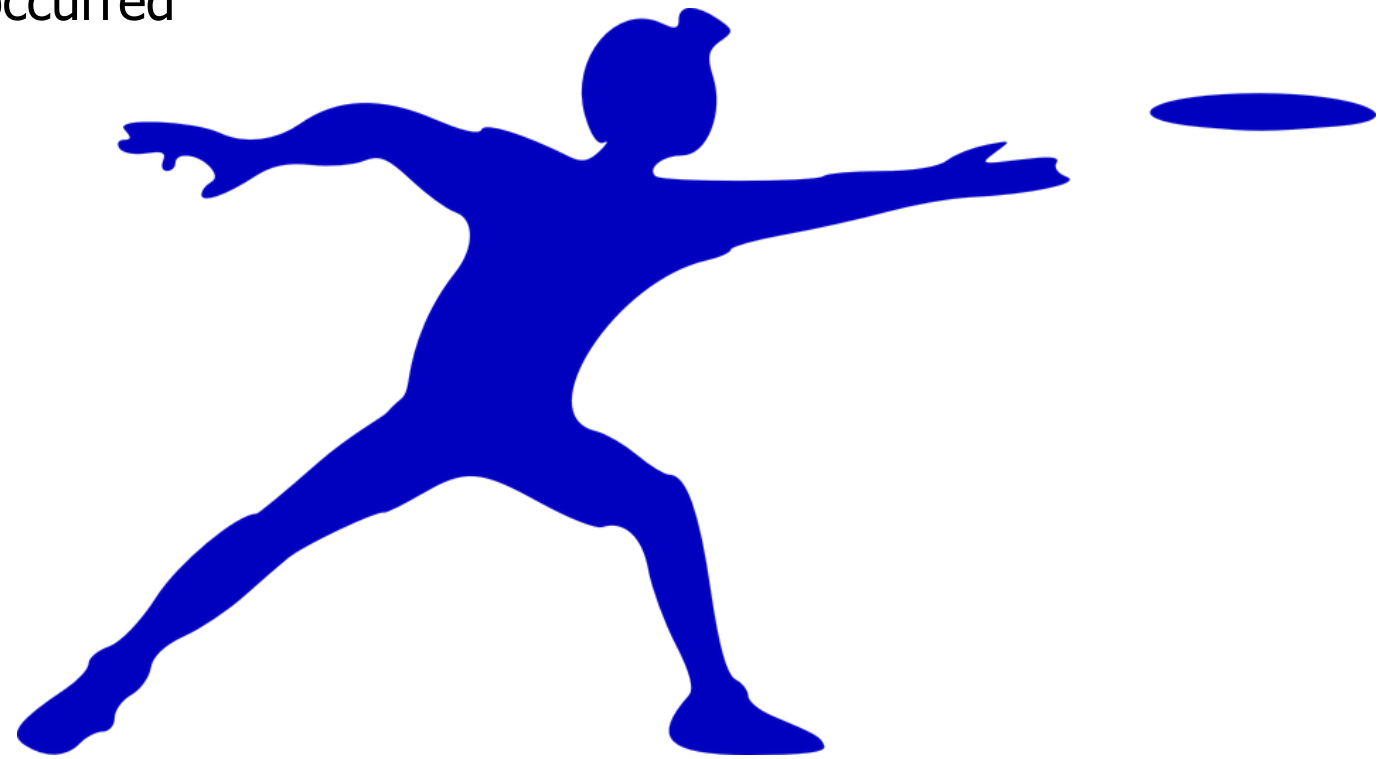


- java.lang.**Exception**
  - java.lang.**CloneNotSupportedException**
  - java.lang.**InterruptedException**
  - java.lang.**ReflectiveOperationException**
    - java.lang.**ClassNotFoundException**
    - java.lang.**IllegalAccessException**
    - java.lang.**InstantiationException**
    - java.lang.**NoSuchFieldException**
    - java.lang.**NoSuchMethodException**
  - java.lang.**RuntimeException**
    - java.lang.**ArithmeticException**
    - java.lang.**ArrayStoreException**
    - java.lang.**ClassCastException**
    - java.lang.**EnumConstantNotPresentException**
    - java.lang.**IllegalArgumentException**
      - java.lang.**IllegalThreadStateException**
      - java.lang.**NumberFormatException**
    - java.lang.**IllegalMonitorStateException**
    - java.lang.**IllegalStateException**
    - java.lang.**IndexOutOfBoundsException**
      - java.lang.**ArrayIndexOutOfBoundsException**
      - java.lang.**StringIndexOutOfBoundsException**
    - java.lang.**NegativeArraySizeException**
    - java.lang.**NullPointerException**
    - java.lang.**SecurityException**
    - java.lang.**TypeNotPresentException**
    - java.lang.**UnsupportedOperationException**

See eskatos.wordpress.com/2009/09/30/how-is-exception-handling-implemented-in-jvms

Java Exception Handling

- If an error condition occurs, an "exception" can be thrown to notify that a problem occurred



Java Exception Handling

- If an error condition occurs, an "exception" can be thrown to notify that a problem occurred
  - Appropriate actions when catching an exception include interacting with the user, logging, or exiting the app

Java Exception Handling

- Any code can report an exception via "throw"

```
public class Vector<E> {
  ...
  public Vector(int initialCapacity,
                ......) {
    if (initialCapacity < 0)
      throw new
        IllegalArgumentException
          ("Illegal Capacity: "
           + initialCapacity);
  ...
  }
  ...
```

See docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html

Java Exception Handling

- Any code can report an exception via "throw"

```
public class Vector<E> {
  ...
  public Vector(int initialCapacity,
                ......) {
    if (initialCapacity < 0)
      throw new
        IllegalArgumentException
          ("Illegal Capacity: "
           + initialCapacity);
  ...
  }
  ...
```

Java Exception Handling

- Any code can report an exception via "throw"

```
public class Vector<E> {
  ...
  public Vector(int initialCapacity,
                ......) {
    if (initialCapacity < 0)
      throw new
        IllegalArgumentException
          ("Illegal Capacity: "
           + initialCapacity);
  ...
  }
  ...
```

Java Exception Handling

- Any code can report an exception via "throw"

```
public class Vector<E> {
  ...
  public int lastIndexOf(Object o,
                         int index){
    if (index >= mCount)
      throw new
      IndexOutOfBoundsException
        (index + " >= "+ mCount);
    ...
  }
```

Java Exception Handling

- Any code can report an exception via "throw"

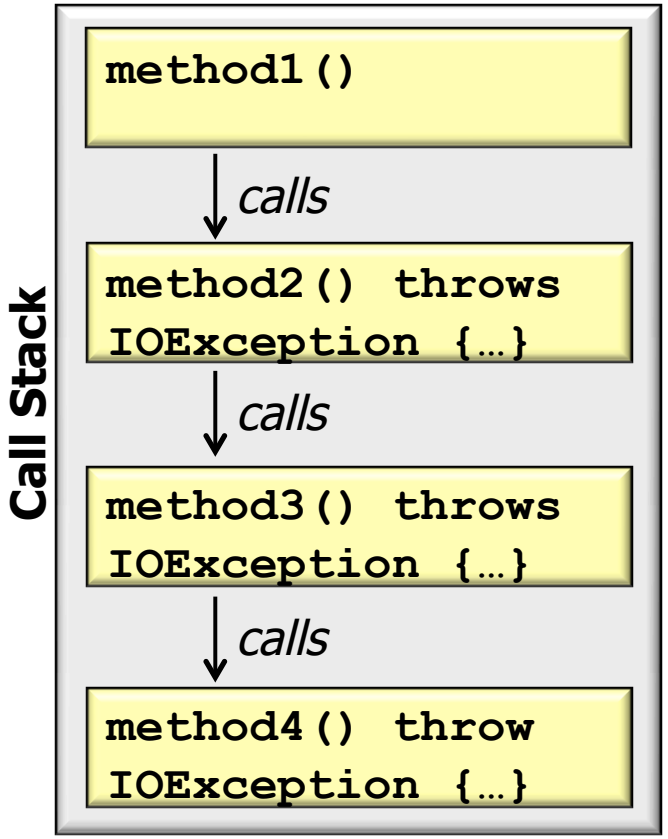
```
public class Vector<E> {
  ...
  public int lastIndexOf(Object o,
                         int index){
    if (index >= mCount)
      throw new
      IndexOutOfBoundsException
        (index + " >= "+ mCount);
    ...
  }
```

Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown

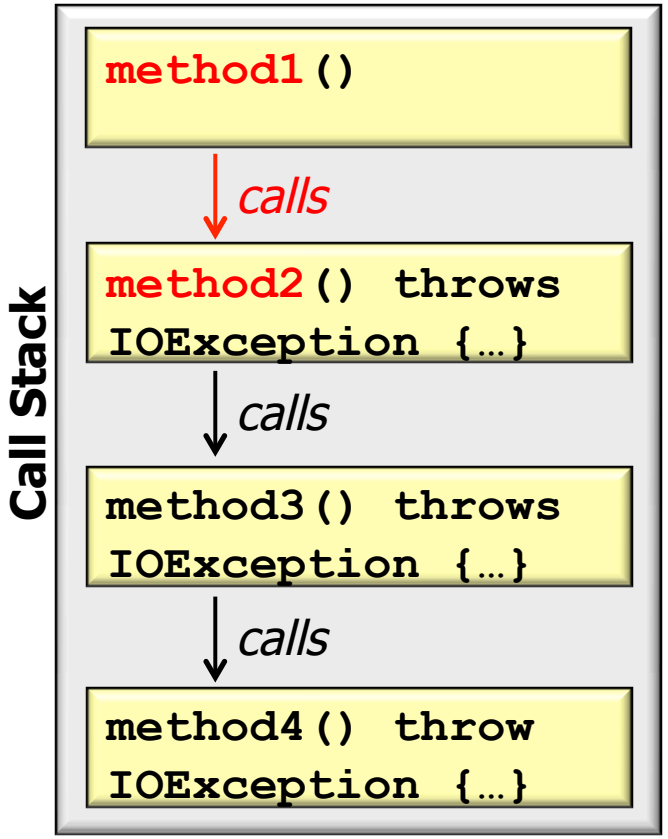
Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown



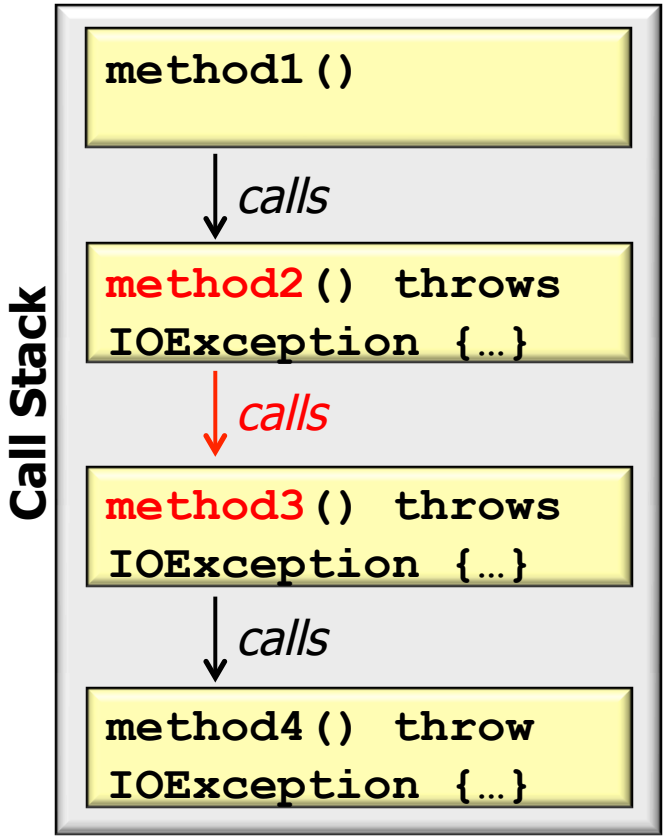
Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown



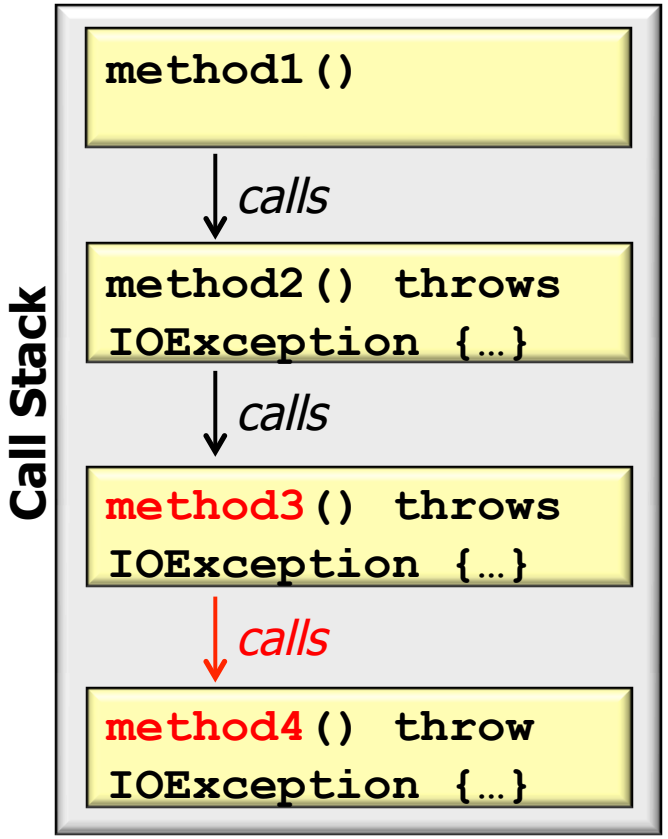
Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown



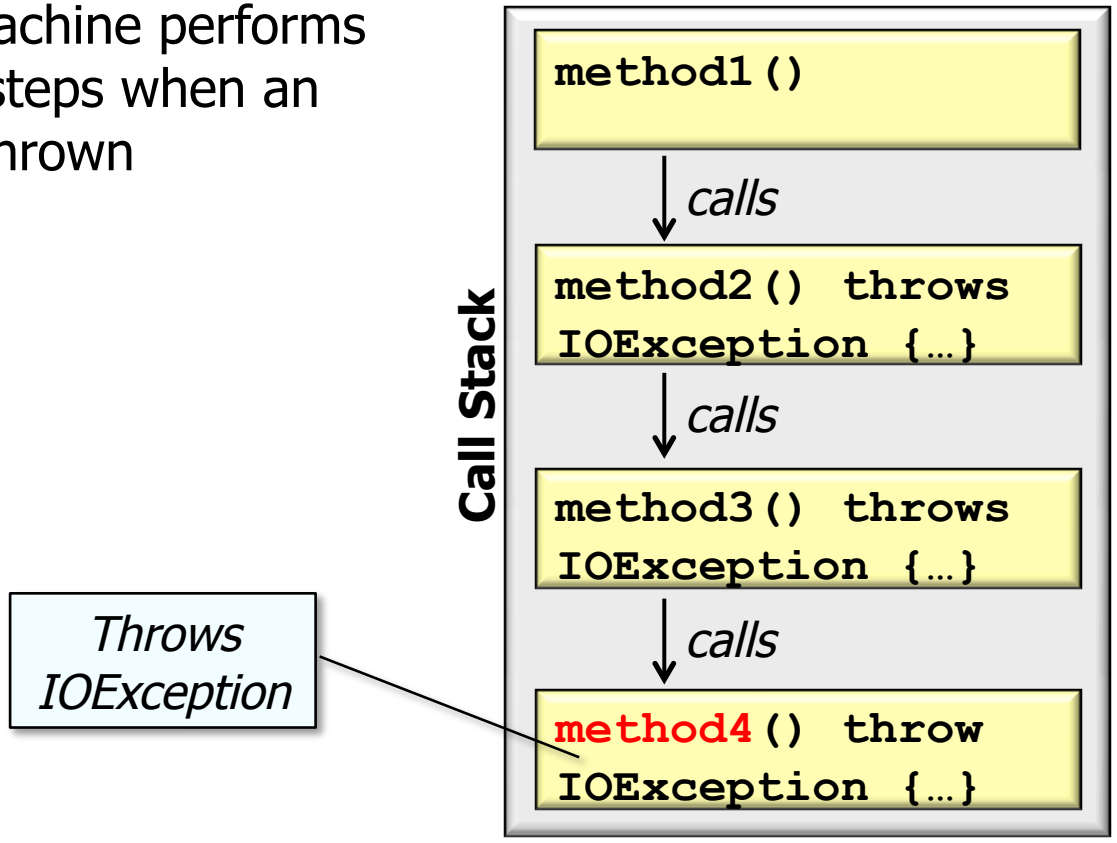
Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown



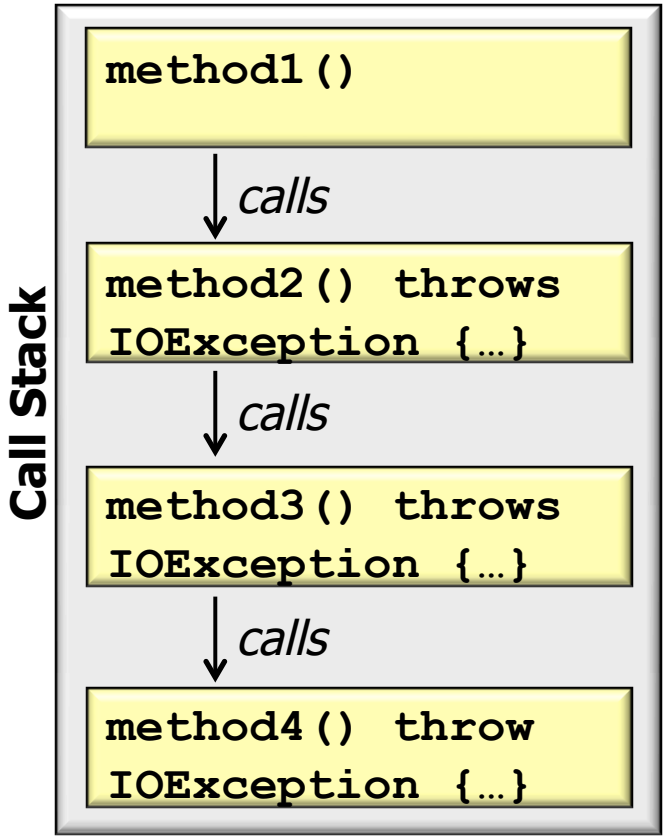
Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown



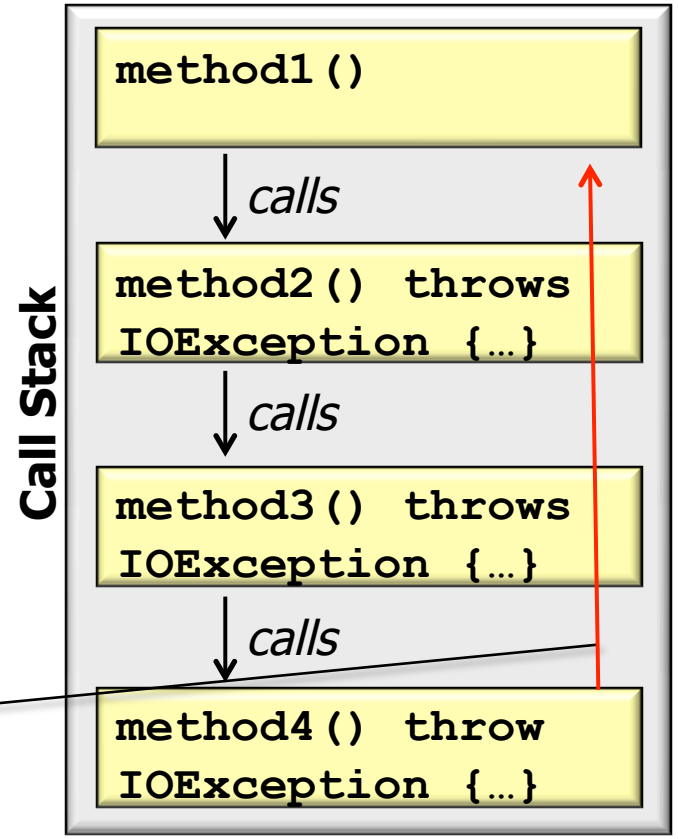
Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown
  1. Normal program execution stops



Java Exception Handling

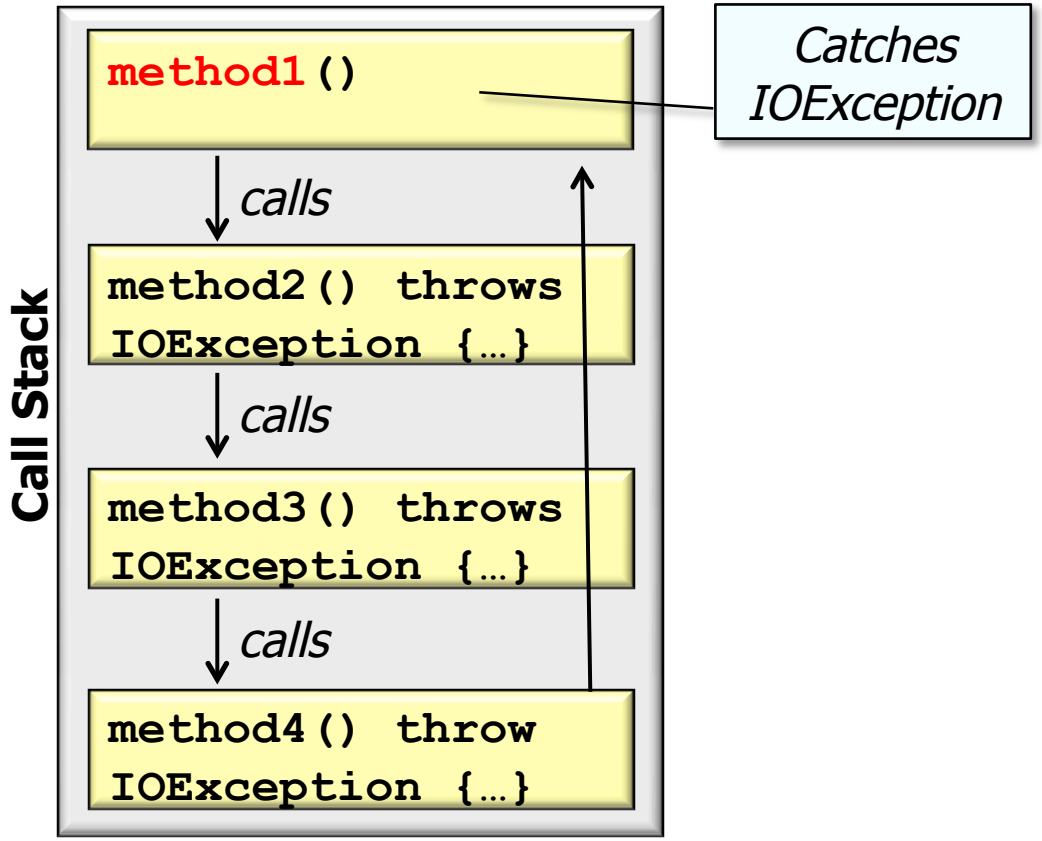
- The virtual machine performs a number of steps when an exception is thrown
  1. Normal program execution stops
  2. Control then transfers to appropriate handler



The virtual machine searches up the runtime call stack to find exception handler

Java Exception Handling

- The virtual machine performs a number of steps when an exception is thrown
  1. Normal program execution stops
  2. Control then transfers to appropriate handler



Java Exception Handling

- A "try block" encloses regions of code that may throw exceptions

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

See docs.oracle.com/javase/tutorial/essential/exceptions/try.html

Java Exception Handling

- A "try block" encloses regions of code that may throw exceptions
  - A try block contains one or more statements that may throw an exception

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

See docs.oracle.com/javase/tutorial/essential/exceptions/try.html

Java Exception Handling

- A "try block" encloses regions of code that may throw exceptions
  - A try block contains one or more statements that may throw an exception

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

Java Exception Handling

- A "try block" encloses regions of code that may throw exceptions
  - A try block contains one or more statements that may throw an exception

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

Java Exception Handling

- One or more "catch blocks" can be associated with a try block

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

See docs.oracle.com/javase/tutorial/essential/exceptions/catch.html

Java Exception Handling

- One or more "catch blocks" can be associated with a try block
  - A catch block defines an exception handler that handles the type of exception indicated by its argument

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

See docs.oracle.com/javase/tutorial/essential/exceptions/catch.html

Java Exception Handling

- One or more "catch blocks" can be associated with a try block
  - A catch block defines an exception handler that handles the type of exception indicated by its argument

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

Java Exception Handling

- One or more "catch blocks" can be associated with a try block
  - A catch block defines an exception handler that handles the type of exception indicated by its argument

```
public class MyClass {
  public MyClass() {
    try {
      Vector<String> v =
        new Vector<>(-1);
    } catch
      (IllegalArgumentException e){
      ...
    }
    ...
```

Java Exception Handling

- A "finally block" *always* runs when the try block exits

```
public class SomeClass {
  ...
  private final ReentrantLock
    lock = new ReentrantLock();
  ...
  public void someMethod() ... {
    final ReentrantLock lock
      = this.lock;
    lock.lock();
    try {
      // ...
    } finally {
      lock.unlock();
    }
   ...
```

See docs.oracle.com/javase/tutorial/essential/exceptions/finally.html

Java Exception Handling

- A "finally block" *always* runs when the try block exits

```
public class SomeClass {
  ...
  private final ReentrantLock
    lock = new ReentrantLock();
  ...
  public void someMethod() ... {
    final ReentrantLock lock
      = this.lock;
    lock.lock();
    try {
      // ...
    } finally {
      lock.unlock();
    }
   ...
```

Runs regardless of whether an exception is thrown or if the try block returns

Java Exception Handling

- A "try-with-resources" try block declares one or more resources that are closed after block exits

```
public class SomeClass {
   ...
   String readFirstLineFromFile
                              (String path)
         throws IOException {
     try (BufferedReader bufReader =
            new BufferedReader
              (new FileReader(path)))
     { return bufReader.readLine(); }
   }
   ...
```

See docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Java Exception Handling

- A "try-with-resources" try block declares one or more resources that are closed after block exits

```
public class SomeClass {
   ...
   String readFirstLineFromFile
                              (String path)
         throws IOException {
     try (BufferedReader bufReader =
            new BufferedReader
              (new FileReader(path)))
     { return bufReader.readLine(); }
   }
   ...
```

Java Exception Handling

- A "try-with-resources" try block declares one or more resources that are closed after block exits

```
public class SomeClass {
   ...
   String readFirstLineFromFile
                              (String path)
         throws IOException {
     try (BufferedReader bufReader =
            new BufferedReader
              (new FileReader(path)))
     { return bufReader.readLine(); }
   }
   ...
```

Java Exception Handling

- A "try-with-resources" try block declares one or more resources that are closed after block exits

```
public class SomeClass {
   ...
   String readFirstLineFromFile
                              (String path)
         throws IOException {
     try (BufferedReader bufReader =
            new BufferedReader
              (new FileReader(path)))
     { return bufReader.readLine(); }
   }
   ...
```

bufReader is automatically closed when the block exits

Java Exception Handling

- Java exceptions are organized as a class hierarchy

- java.lang.**Exception**
  - java.lang.**CloneNotSupportedException**
  - java.lang.**InterruptedException**
  - java.lang.**ReflectiveOperationException**
    - java.lang.**ClassNotFoundException**
    - java.lang.**IllegalAccessException**
    - java.lang.**InstantiationException**
    - java.lang.**NoSuchFieldException**
    - java.lang.**NoSuchMethodException**
  - java.lang.**RuntimeException**
    - java.lang.**ArithmeticException**
    - java.lang.**ArrayStoreException**
    - java.lang.**ClassCastException**
    - java.lang.**EnumConstantNotPresentException**
    - java.lang.**IllegalArgumentException**
      - java.lang.**IllegalThreadStateException**
      - java.lang.**NumberFormatException**
    - java.lang.**IllegalMonitorStateException**
    - java.lang.**IllegalStateException**
    - java.lang.**IndexOutOfBoundsException**

...

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses

```
public class MyClass {
  public void aMethod() {
    try { ... }
    catch
     (IndexOutOfBoundsException i)
     { ... }
    catch
     (IllegalArgumentException e)
     { ... }
  }
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses

```
public class MyClass {
  public void aMethod() {
    try { ... }
    catch
     (IndexOutOfBoundsException i)
     { ... }
    catch
     (IllegalArgumentException e)
     { ... }
  }
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses

```
public class MyClass {
  public void aMethod() {
    try { ... }
    catch
     (IndexOutOfBoundsException i)
     { ... }
    catch
     (IllegalArgumentException e)
     { ... }
 }
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called

```
public class MyClass {
  public void method1() {
    try { method2(); }
    catch (SomeException e) {
      // do something w/exception
      e.printStackTrace();
    }

  public void method2()
  { method3(); }

  public void method3()
  { throw new SomeException(); }
}
```

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called
  - To handle every possible type of exception, catch Exception

```
public class MyClass {
  public void methodA() {
    ...
  }

  public void methodB() {
    try { methodA(); }
    catch (Exception e) {
      // do something w/exception
    }
  }
```

See developer.android.com/reference/java/lang/Exception.html

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called
  - To handle every possible type of exception, catch Exception
  - To hancle every possible type of error, catch Throwable

```
public class MyClass {
  public void methodA() {
    ...
  }

  public void methodB() {
    try { methodA(); }
    catch (Throwable e) {
      // Every recoverable runtime
      // error and exception will
      // be caught
      e.printStackTrace();
    }
  }
```

See developer.android.com/reference/java/lang/Throwable.html

Java Exception Handling

- Java exceptions are organized as a class hierarchy
  - Handlers can be specified for different exception subclasses
  - If a handler isn't specified for the exception thrown, the next matching handler up the runtime stack get called
  - To handle every possible type of exception, catch Exception
  - To handle every possible type of error, catch Throwable

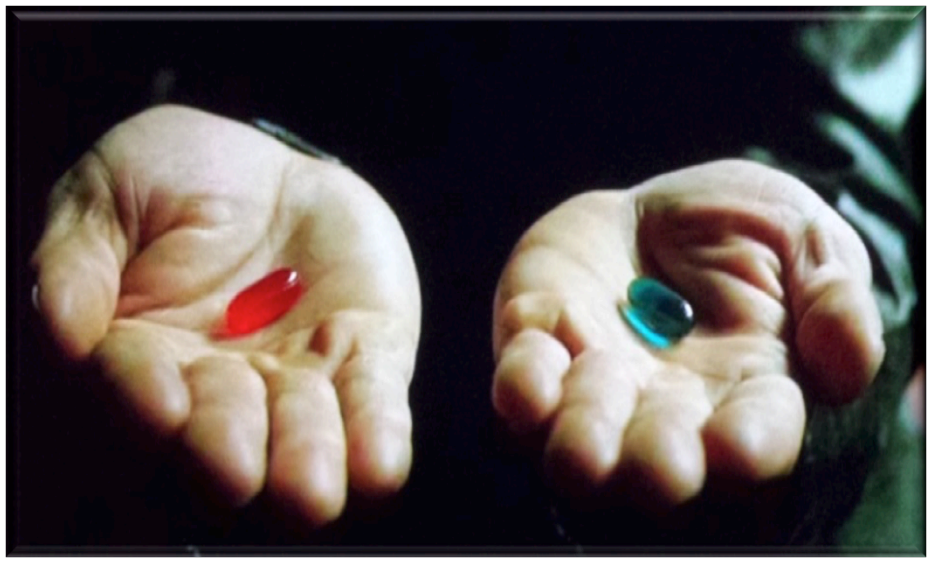
```
public class MyClass {
  public void methodA() {
    ...
  }

  public void methodB() {
    try { methodA(); }
    catch (Throwable t) {
      // Every recoverable runtime
      // error and exception will
      // be caught
      t.printStackTrace();
    }
  }
```

See developer.android.com/reference/java/lang/Throwable.html#printStackTrace()

Java Exception Handling

- There are two types of exceptions:



Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
    - Methods declaring "checked exceptions" require callers to provide an exception handler

```
public class Vector<E> {
   private void writeObject
     (java.io.ObjectOutputStream s)
     throws java.io.IOException {
     ...
}
...
```

See en.wikibooks.org/wiki/Java_Programming/Checked_Exceptions

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
    - Methods declaring "checked exceptions" require callers to provide an exception handler

```
public class Vector<E> {
   private void writeObject
      (java.io.ObjectOutputStream s)
      throws java.io.IOException {
      ...
   }
   ...
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
    - Methods declaring "checked exceptions" require callers to provide an exception handler

```
public class Vector<E> {
  private void writeObject
    (java.io.ObjectOutputStream s)
    throws java.io.IOException {
    ...
  }
  ...
  void method1(Vector<Integer> v
               ObjectOutputStream s) {
    try { v.writeObject(s); }
    catch (IOException ioe) {
      ...
    }
  }
...
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
    - Methods declaring "checked exceptions" require callers to provide an exception handler

```
public class Vector<E> {
  private void writeObject
    (java.io.ObjectOutputStream s)
    throws java.io.IOException {
    ...
  }
  ...
  void method1(Vector<Integer> v
               ObjectOutputStream s) {
    try { v.writeObject(s); }
    catch (IOException ioe) {
      ...
    }
  }
...
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
  - *Unchecked exceptions*
    - Can be thrown without a method declaring it

```
public class MyClass {
  public void methodA(int index) {
    if (index < 0)
      throw new
      IndexOutOfBoundsException();
    ...
  }

  public void methodB() {
    try { methodA(-1); }
    catch (Exception e) {
      throw new
        RuntimeException(e);
    }
  }
```

See docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
  - *Unchecked exceptions*
    - Can be thrown without a method declaring it

```
public class MyClass {
  public void methodA(int index) {
    if (index < 0)
      throw new
      IndexOutOfBoundsException();
    ...
  }

  public void methodB() {
    try { methodA(-1); }
    catch (Exception e) {
      throw new
        RuntimeException(e);
    }
  }
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
  - *Unchecked exceptions*
    - Can be thrown without a method declaring it

```
public class MyClass {
  public void methodA(int index) {
    if (index < 0)
      throw new
      IndexOutOfBoundsException();
    ...
  }

  public void methodB() {
    try { methodA(-1); }
    catch (Exception e) {
      throw new
        RuntimeException(e);
    }
  }
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
  - *Unchecked exceptions*
    - Can be thrown without a method declaring it

```
public class MyClass {
  public void methodA(int index) {
    if (index < 0)
      throw new
      IndexOutOfBoundsException();
    ...
  }

  public void methodB() {
    methodA(-1);
  }
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
  - *Unchecked exceptions*
    - Can be thrown without a method declaring it

```
public class MyClass {
  public void methodA(int index) {
    if (index < 0)
      throw new
      IndexOutOfBoundsException();
    ...
  }

  public void methodB() {
    methodA(-1);
  }
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
  - *Unchecked exceptions*
    - Can be thrown without a method declaring it

```
public class MyClass {
  public void methodA(int index) {
    if (index < 0)
      throw new
      IndexOutOfBoundsException();
    ...
  }

  public void methodB() {
    try { methodA(-1); }
    catch (Exception e) {
      throw new
        RuntimeException(e);
    }
  }
```

Java Exception Handling

- There are two types of exceptions:
  - *Checked exceptions*
  - *Unchecked exceptions*
    - Can be thrown without a method declaring it

```
public class MyClass {
  public void methodA(int index) {
    if (index < 0)
      throw new
      IndexOutOfBoundsException();
    ...
  }

  public void methodB() {
    try { methodA(-1); }
    catch (Exception e) {
      throw new
        RuntimeException(e);
    }
  }
```

Java Exception Handling

There's still more to learn about Java exceptions!

See docs.oracle.com/javase/tutorial/essential/exceptions