Exceptions and Error Handling

Jim Wilson

@hedgehogjim | blog.jwhh.com | jimw@jwhh.com

What to Expect in This Module

The role of exceptions

The try/catch/finally statement

Exceptions crossing method boundaries

Throwing exceptions

Custom exception types

Error Handling with Exceptions

Error handling needs to be implicit in application development

The traditional approach of checking error codes/flags is too intrusive

Exceptions provide a non-intrusive way to signal errors

try/catch/finally provides a structured way to handle exceptions

The try block contains the "normal" code to execute

Block executes to completion unless an exception is thrown

The catch block contains the error handling code

Block executes only if matching exception is thrown

The finally block contains cleanup code if needed

Runs in all cases following try or catch block

Error Handling with Exceptions

```
int i = 12;
int j = 2;

try {                      0
  int result = i / (j - 2);
  System.out.println(result);
} catch(Exception e) {
  System.out.println(
     "Error: " + e.getMessage());
  e.printStackTrace();
}
```

```
Error: / by zero
```

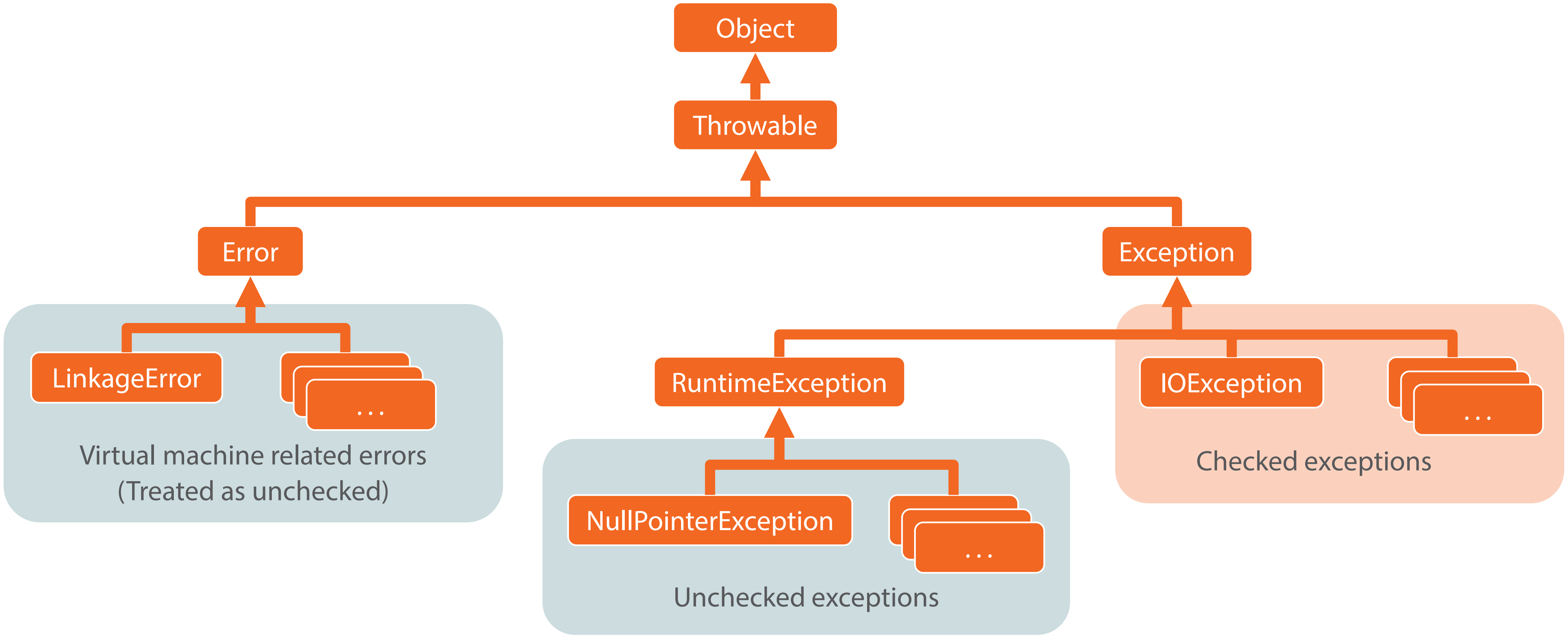
Error Handling with Exceptions

```
BufferedReader reader = null;
int total = 0;
try {
  reader =
    new BufferedReader(new FileReader("C:\\Numbers.txt"));
  String line = null;
   while ((line = reader.readLine()) != null)
     total += Integer.valueOf(line);
   System.out.println("Total: " + total);
} catch(Exception e) {
    System.out.println(e.getMessage());
} finally {
   try {
     if(reader != null)
        reader.close();
   } catch(Exception e) {
       System.out.println(e.getMessage());
   }
}
```

C:\Numbers.txt

```
5
12
6
4
```

Exception Class Hierarchy



Typed Exceptions

Exceptions can be handled by type

Each exception type can have a separate catch block

Each catch is tested in order from top to bottom

First assignable catch is selected

Start catch blocks with most specific exception types

Error Handling with Exceptions

```java
BufferedReader reader = null;
int total = 0;
try {
  reader =
    new BufferedReader(new FileReader("C:\\Numbers.txt"));
  String line = null;
  while ((line = reader.readLine()) != null)
     total += Integer.valueOf(line);
  System.out.println("Total: " + total);
} catch(Exception e) {
   System.out.println(e.getMessage());
} finally {
  try {
    if(reader != null)
       reader.close();
  } catch(Exception e) {
      System.out.println(e.getMessage());
  }
}
```

C:\Numbers.txt

```
5
12
6
4
```

Error Handling with Exceptions

```java
BufferedReader reader = null;
int total = 0;
try {
  . . .
} catch(Exception e) {
   System.out.println(e.getMessage());









} finally {
   . . .
}
```

C:\Numbers.txt

```
5
12
6
4
```

Error Handling with Exceptions

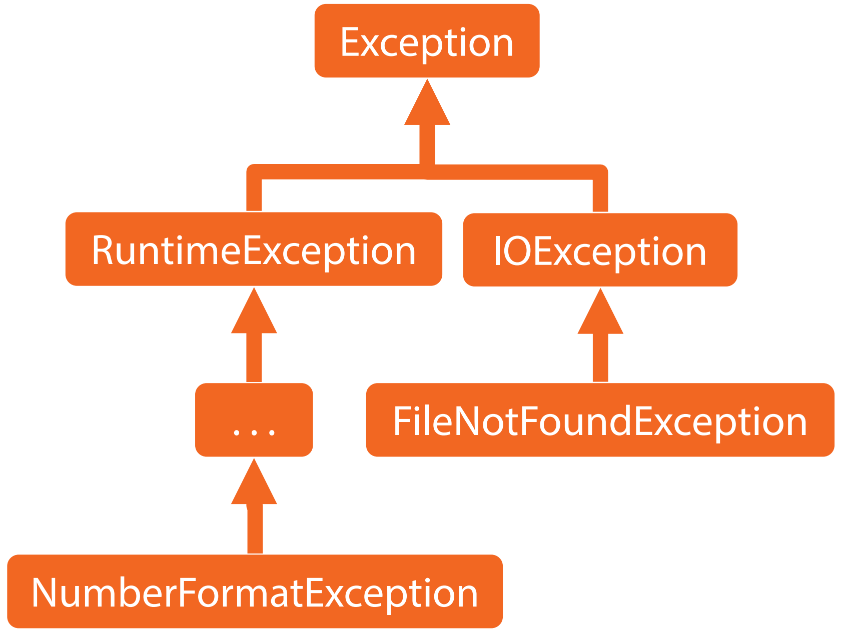
```
BufferedReader reader = null;
int total = 0;
try {
  . . .
} catch(Exception e) {
   System.out.println(e.getMessage());
} catch(NumberFormatException e) {
   System.out.println(“Invalid value: “ +
     e.getMessage());




} finally {
   . . .
}
```

C:\Numbers.txt

```
5
12
6
4
```

Error Handling with Exceptions

```
BufferedReader reader = null;
int total = 0;
try {
  . . .
} catch(NumberFormatException e) {
   System.out.println(“Invalid value: “ +
     e.getMessage());
} catch(FileNotFoundException e) {
   System.out.println(“Not found: “ +
     e.getMessage());
} catch(IOException e) {
   System.out.println(“Error interacting with file: “ +
     e.getMessage());
} finally {
   . . .
}
```



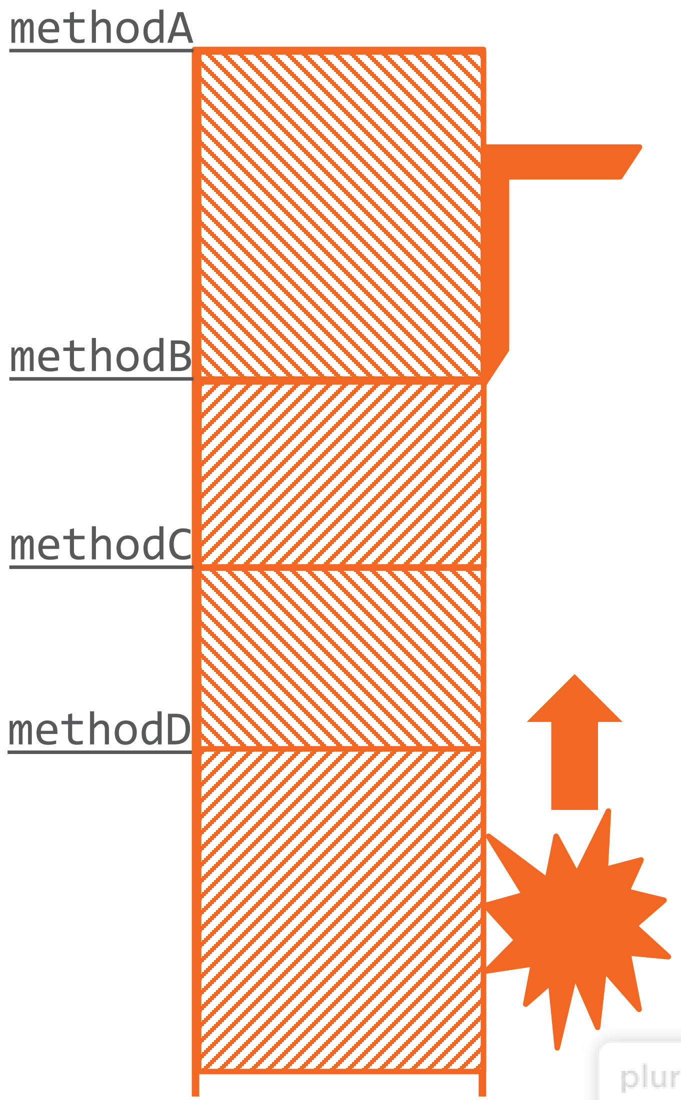
Exceptions and Methods

```
void methodA() {
  . . .
  try {
    methodB();
  } catch(. . .) {
    . . .
  }
}
```

```
void methodB() {
   . . .
   methodC();
}
```

```
void methodC() {
   . . .
   methodD();
}
```

```
void methodD() {
   . . .
   // Does something
   //  that throws an
   //  an exception
}
```



Exceptions and Methods

Exceptions propagate up the call stack

Can cross method boundaries

Exceptions are part of a method's contract

Method is responsible for any checked exceptions that might occur

Catch the exception

Document that the exception might occur

Use the throws clause

Exceptions and Methods

```java
public class Flight {
  int passengers;
  // other members elided for clarity
  public void addPassengers(String filename) throws IOException  {
    BufferedReader reader = null;
    try {
       reader = new BufferedReader(new FileReader(filename));
       String line = null;
       while ((line = reader.readLine()) != null) {
          String[] parts = line.split(" ");
          passengers += Integer.valueOf(parts[0]);
       }
     } finally {
       if(reader != null)
          reader.close();
     }
  }
}
```

C:\PassengerList.txt

```
2   Wilson
4   Rodriguez
7   Smith
4   Sharma
```

Exceptions and Method Overriding

The throws clause of an overriding method must be compatible with the throws clause of the overridden method

Can exclude exceptions

Can have the same exception

Can have a derived exception

```
public class CargoFlight extends Flight {
  // other members elided for clarity

  @Override                                       throws IOException
  public void addPassengers(String filename)                              {
    // ...                                        throws FileNotFoundException
  }
}
```

Throwing Exceptions

Your code can throw exceptions

Use the throw keyword

Must create exception instance before throwing

Be sure to provide meaningful detail

Most exception classes provide a constructor that accepts a String message or other detail

When caused by another exception, include originating exception

All exception classes support initCause method

Many provide a constructor that accepts the originating exception

Creating a Custom Exception Type

You can create your own custom exception types

In most cases better to use existing exception type

Normally inherit directly from Exception class

Makes them checked exceptions

Constructors are often their only members

Most required functionality is inherited

Constructor that accepts required detail

Constructor that accepts required detail and originating exception

Demo

CalcEngine with Exceptions

Summary

- Exceptions provide a non-intrusive way to signal errors
- try/catch/finally provide a structured way to handle exceptions
- Exceptions are caught by type
  - Can have separate catch statement for differing exception types
  - Catch from most specific type to least specific
- Raise exceptions using throw
- Methods must declare any unhandled checked exceptions using throws
- Can create custom exception types
  - Normally inherit from Exception