CS1083 Week 5: Exceptions

Chapter 11

David Bremner

2018-02-01

Exceptions

Why?

- Universal error reporting method
- Does not interfere with return values of methods
- Provides a stack trace

Exceptions are objects

- Create an exception

```
RunTimeException RE=new RunTimeException("Oops.");
```

- Throw it

```
        throw RE;
```

Why?

- Universal error reporting method
- Does not interfere with return values of methods
- Provides a stack trace

Exceptions are objects

- Create an exception

```
RunTimeException RE=new RunTimeException("Oops.");
```

- Throw it

```
      throw RE;
```

Throwing example

```
public class RETest{
  public static void crash(){
    RuntimeException RE=new RuntimeException("Oops");
    throw RE;
  }
  public static void main(String []args){ crash(); }
}
```

RETest

Throwing example

```
public class RETest{
  public static void crash(){
    RuntimeException RE=new RuntimeException("Oops");
    throw RE;
  }
  public static void main(String []args){ crash(); }
}
```

Stack trace:

```
Exception in thread "main"
               java.lang.RuntimeException: Oops
        at RETest.crash(RETest.java:3)
        at RETest.main(RETest.java:7)

Process RETest exited abnormally with code 1
```

Throwing example

Stack trace:

```
Exception in thread "main"
               java.lang.RuntimeException: Oops
        at RETest.crash(RETest.java:3)
        at RETest.main(RETest.java:7)

Process RETest exited abnormally with code 1
```

- Why is the line number in the stack trace 3 instead of 4?

What to throw?

- java.lang.Exception
    - java.lang.RuntimeException

- ArithmeticException
- ClassCastException

- IllegalArgumentException
    - IllegalThreadStateException
    - NumberFormatException

What to throw?

- java.lang.Exception
    - java.lang.RuntimeException

        - ArithmeticException
        - ClassCastException

        - IllegalArgumentException
            - IllegalThreadStateException
            - NumberFormatException

What to throw?

- java.lang.Exception
    - java.lang.RuntimeException

- ArithmeticException
- ClassCastException

- IllegalArgumentException
    - IllegalThreadStateException
    - NumberFormatException

What to throw?

- IndexOutOfBoundsException
    - ArrayIndexOutOfBoundsException
    - StringIndexOutOfBoundsException

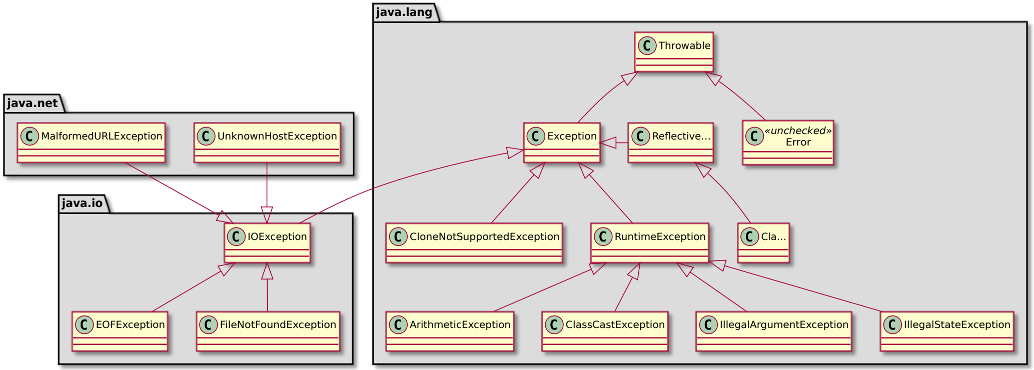
- NullPointerException

What to throw?

- IndexOutOfBoundsException
    - ArrayIndexOutOfBoundsException
    - StringIndexOutOfBoundsException

- NullPointerException

Exceptions and packages



Creating your own exception class

ImageFormatException

```
public class ImageFormatException extends RuntimeException
    public ImageFormatException() {  super();   }
    public ImageFormatException(String msg) { super(msg); }
}
```

And using it

Creating your own exception class

ImageFormatException

```
public class ImageFormatException extends RuntimeException
    public ImageFormatException() {  super();   }
    public ImageFormatException(String msg) { super(msg); }
}
```

And using it

```
if (!sc.next().equals("P1"))
    throw new ImageFormatException("PBM␣Format␣error");
```

PBM2

Catching Exceptions

```
try{
                         ⋮
  throw new RuntimeException("Oops.")
  //execution stops here
                         ⋮
}

catch (RuntimeException e){
  // and resumes here.
}
```

- When an exception is thrown, execution stops in that method, and resumes in the *smallest* enclosing catch block.

Catching Exceptions

```
try{
                         ⋮
  throw new RuntimeException("Oops.")
  //execution stops here
                         ⋮
}

catch (RuntimeException e){
  // and resumes here.
}
```

- When an exception is thrown, execution stops in that method, and resumes in the *smallest* enclosing catch block.

Catching Exceptions

```
try{
                              ⋮
  throw new RuntimeException("Oops.")
  //execution stops here
                              ⋮
}

catch (RuntimeException e){
  // and resumes here.
}
```

- When an exception is thrown, execution stops in that method, and resumes in the *smallest* enclosing catch block.

Enclosing handlers I

Propagate

```
public class Propagate{
    static void a(){
        throw new RuntimeException("A");
    }









}
```

Enclosing handlers I

```
public class Propagate{
    static void a(){
        throw new RuntimeException("A");
    }
    static void b(){  a();  }






}
```

Enclosing handlers I

Propagate

```
public class Propagate{
    static void a(){
        throw new RuntimeException("A");
    }
    static void b(){  a();  }
    public static void main(String[] args){






    }
}
```

Enclosing handlers I

Propagate

```
public class Propagate{
    static void a(){
        throw new RuntimeException("A");
    }
    static void b(){  a();  }
    public static void main(String[] args){
        try{ b(); }




    }
}
```

Enclosing handlers I

Propagate

```
public class Propagate{
    static void a(){
        throw new RuntimeException("A");
    }
    static void b(){  a();  }
    public static void main(String[] args){
        try{ b(); }
        catch (RuntimeException e){
          System.out.println(e.getMessage());
        }
    }
}
```

Enclosing exception handlers II

```
public class Propagate2{
    static void a(){
        throw new RuntimeException("A"); }
```

Enclosing exception handlers II

```
public class Propagate2{
    static void a(){
        throw new RuntimeException("A"); }
    static void b(){
        try { a(); }
        catch (RuntimeException e)
            {  System.out.println("b"); }
    }
```

Enclosing exception handlers II

```
public class Propagate2{
    static void a(){
        throw new RuntimeException("A"); }
    static void b(){
        try { a(); }
        catch (RuntimeException e)
            {  System.out.println("b"); }
    }
    public static void main(String[] args){
        try{ b(); }
        catch (RuntimeException e){
            System.out.println("main"); }
    }
```

Catching more than one exception

```java
public class Catch{
    public static void main(String[] args){









    } // Order of catches?
}
```

Catching more than one exception

```
public class Catch{
    public static void main(String[] args){
        try{throw new ImageFormatException("yuck!");}







    } // Order of catches?
}
```

Catching more than one exception

```
public class Catch{
    public static void main(String[] args){
        try{throw new ImageFormatException("yuck!");}
        catch (ImageFormatException e){
            System.out.println("caught it");
        }




    } // Order of catches?
}
```

Catching more than one exception

```
public class Catch{
    public static void main(String[] args){
        try{throw new ImageFormatException("yuck!");}
        catch (ImageFormatException e){
            System.out.println("caught it");
        }
        catch (RuntimeException e){
            System.out.println(e.getMessage());
        }
    } // Order of catches?
}
```

The finally clause

- The finally clause is executed *whether or not* there is an exception

```
public class Finally{
  public static void main(String[] args){
    try{
      //nothing
    }
    finally{
      System.out.println("finally");
    }
  }
}
```

The finally clause

- The finally clause is executed *whether or not* there is an exception

```
public class Finally{
  public static void main(String[] args){
    try{
      //nothing
    }
    finally{
      System.out.println("finally");
    }
  }
}
```

More finally

```
public class Finally2{
  public static void main(String[] args){
    try{
      throw new RuntimeException("boom");
    }
    finally{
      System.out.println("finally");
    }
  }
}
```

More finally

```
public class Finally2{
  public static void main(String[] args){
    try{
      throw new RuntimeException("boom");
    }
    finally{
      System.out.println("finally");
    }
  }
}
```

```
finally
Exception in thread "main"
        java.lang.RuntimeException: boom
        at Finally2.main(Finally2.java:5)
```

Catch and release

```
public class Finally3{
  public static void main(String[] args){
    try{
      throw new RuntimeException("boom");
    }








  }
}
```

Catch and release

```
public class Finally3{
  public static void main(String[] args){
    try{
      throw new RuntimeException("boom");
    }
    catch (RuntimeException e){
      System.out.println("I got it!");
    }




  }
}
```

Catch and release

```
public class Finally3{
  public static void main(String[] args){
    try{
      throw new RuntimeException("boom");
    }
    catch (RuntimeException e){
      System.out.println("I got it!");
    }
    finally{ System.out.println("finally");  }


  }
}
```

Catch and release

```
public class Finally3{
  public static void main(String[] args){
    try{
      throw new RuntimeException("boom");
    }
    catch (RuntimeException e){
      System.out.println("I got it!");
    }
    finally{ System.out.println("finally");  }
    System.out.println(
                "Exception, what exception?");
  }
}
```

Catching and missing

```
public class Catch2{
    public static void a(){
        throw new RuntimeException("Thrown");  }
```

Catching and missing

```
public class Catch2{
    public static void a(){
        throw new RuntimeException("Thrown");   }
    public static void b(){
        try { a(); }
        catch (ImageFormatException e){
            System.out.println(e.getMessage());
        } }
```

Catching and missing

```
public class Catch2{
    public static void a(){
        throw new RuntimeException("Thrown");   }
    public static void b(){
        try { a(); }
        catch (ImageFormatException e){
            System.out.println(e.getMessage());
        } }
    public static void main(String[] args){
        try{  b(); }
        catch (RuntimeException e){
            System.out.println("Caught");
        }   }
```

Checked and unchecked exceptions

- Any exception class that is not a subclass of `RuntimeException` is *checked*
- *checked* exceptions *must* be dealt with, or the program will not compile.
- An important example of a checked exception is `IOException`

Checked and unchecked exceptions

- Any exception class that is not a subclass of `RuntimeException` is *checked*
- *checked* exceptions *must* be dealt with, or the program will not compile.
- An important example of a checked exception is `IOException`

Checked and unchecked exceptions

- Any exception class that is not a subclass of `RuntimeException` is *checked*
- *checked* exceptions *must* be dealt with, or the program will not compile.
- An important example of a checked exception is `IOException`

IOException Example

```
import java.io.IOException;
public class CheckedException{
  public static void crash(){
    throw new IOException("Compile␣this!");
  }
}
```

IOException Example

```java
import java.io.IOException;
public class CheckedException{
  public static void crash(){
    throw new IOException("Compile␣this!");
  }
}
```

```
CheckedException.java:4: unreported exception
   java.io.IOException; must be caught
                 or declared to be thrown
        throw new IOException("Compile this!");
        ^
1 error
```

Need not be thrown directly

```
public class CheckedException2{
  public static void open(String filename) {
    BufferedReader infile=
        new BufferedReader(
               new FileReader(filename));
  }
}
```

Need not be thrown directly

```
public class CheckedException2{
  public static void open(String filename) {
    BufferedReader infile=
        new BufferedReader(
               new FileReader(filename));
  }
}
```

```
CheckedException2.java:5: unreported exception
    java.io.FileNotFoundException;
      must be caught or declared to be thrown
      BufferedReader infile=
        new BufferedReader(
             new FileReader(filename));
                                  ^
1 error
```

must be caught or declared to be thrown

Caught

```
public static void open(String filename){
  try {
        BufferedReader infile= new BufferedReader(
                new FileReader(filename));
      }
      catch (IOException e){
          System.out.println("Aieeeeee!");
          e.printStackTrace();
          System.exit(1);
      }
}
```

must be caught or declared to be thrown

Caught

```java
public static void open(String filename){
   try {
           BufferedReader infile=  new BufferedReader(
                   new FileReader(filename));
         }
         catch (IOException e){
             System.out.println("Aieeeeee!");
             e.printStackTrace();
             System.exit(1);
         }
}
```

```
Aieeeeee!
java.io.FileNotFoundException: foo
                (No such file or directory)
  at java.io.FileInputStream.open(Native Method)
  at java.io.FileInputStream.<init>(FileInputStream.java:10
  at java.io.FileInputStream.<init>(FileInputStream.java:66
  at java.io.FileReader.<init>(FileReader.java:41)
  at CheckedException3.open(CheckedException3.java:7)
  at CheckedException3.main(CheckedException3.java:16)

Process CheckedException3 exited abnormally with
code 1
```

Declared to be thrown

```
public class CheckedException4{
  public static void open(String filename)
                       throws IOException{
      BufferedReader infile=  new BufferedReader(
                       new FileReader(filename));
  }
  public static void main(String[] args){  open("foo"); }
```

- Does this work?
- Why or why not?

Declared to be thrown

```
public class CheckedException4{
  public static void open(String filename)
                        throws IOException{
      BufferedReader infile=  new BufferedReader(
                        new FileReader(filename));
  }
  public static void main(String[] args){  open("foo"); }
```

- Does this work?
- Why or why not?

Declared to be thrown

```
public class CheckedException4{
  public static void open(String filename)
                         throws IOException{
      BufferedReader infile=  new BufferedReader(
                         new FileReader(filename));
  }
  public static void main(String[] args){   open("foo"); }
```

- Does this work?
- Why or why not?

Declared to be thrown

```
public class CheckedException4{
  public static void open(String filename)
                          throws IOException{
      BufferedReader infile=  new BufferedReader(
                          new FileReader(filename));
  }
  public static void main(String[] args){  open("foo"); }
```

- Does this work?
- Why or why not?

```
CheckedException4.java:10: unreported exception
java.io.IOException; must be caught
      or declared to be thrown
      open("foo");
```

```
public static void main(String[] args)
                    throws IOException{
  open("foo");
}
```

```
Exception in thread "main" java.io.FileNotFoundException:
        foo (No such file or directory)
  at java.io.FileInputStream.open(Native Method)
  at java.io.FileInputStream.<init>(FileInputStream.java:10
  at java.io.FileInputStream.<init>(FileInputStream.java:66
  at java.io.FileReader.<init>(FileReader.java:41)
  at CheckedException4.open(CheckedException4.java:6)
        at CheckedException4.main(CheckedException4.java:10
```

- Throwing exceptions from main is generally bad style
- Checked exceptions are meant to be checked

```
public static void main(String[] args)
                      throws IOException{
  open("foo");
}

Exception in thread "main" java.io.FileNotFoundException:
        foo (No such file or directory)
  at java.io.FileInputStream.open(Native Method)
  at java.io.FileInputStream.<init>(FileInputStream.java:10
  at java.io.FileInputStream.<init>(FileInputStream.java:66
  at java.io.FileReader.<init>(FileReader.java:41)
  at CheckedException4.open(CheckedException4.java:6)
        at CheckedException4.main(CheckedException4.java:10
```

- Throwing exceptions from main is generally bad style
- Checked exceptions are meant to be checked

```
public static void main(String[] args)
                    throws IOException{
  open("foo");
}

Exception in thread "main" java.io.FileNotFoundException:
        foo (No such file or directory)
  at java.io.FileInputStream.open(Native Method)
  at java.io.FileInputStream.<init>(FileInputStream.java:10
  at java.io.FileInputStream.<init>(FileInputStream.java:66
  at java.io.FileReader.<init>(FileReader.java:41)
  at CheckedException4.open(CheckedException4.java:6)
        at CheckedException4.main(CheckedException4.java:10
```

- Throwing exceptions from main is generally bad style
- Checked exceptions are meant to be checked

```
public static void main(String[] args)
                    throws IOException{
  open("foo");
}

Exception in thread "main" java.io.FileNotFoundException:
        foo (No such file or directory)
  at java.io.FileInputStream.open(Native Method)
  at java.io.FileInputStream.<init>(FileInputStream.java:10
  at java.io.FileInputStream.<init>(FileInputStream.java:66
  at java.io.FileReader.<init>(FileReader.java:41)
  at CheckedException4.open(CheckedException4.java:6)
        at CheckedException4.main(CheckedException4.java:10
```

- Throwing exceptions from main is generally bad style
- Checked exceptions are meant to be checked

When to catch and when to throw?

You should definitely catch if

- You expect an error fairly often (user input), and
- You (or the user) can correct the error.

You should definitely not catch if

- The error is extremely rare.
- You have no way of correcting the error

Checked exceptions in Java do not always follow these rules; hard luck for us.

When to catch and when to throw?

You should definitely catch if

- You expect an error fairly often (user input), and
- You (or the user) can correct the error.

You should definitely not catch if

- The error is extremely rare.
- You have no way of correcting the error

Checked exceptions in Java do not always follow these rules; hard luck for us.

When to catch and when to throw?

You should definitely catch if

- You expect an error fairly often (user input), and
- You (or the user) can correct the error.

You should definitely not catch if

- The error is extremely rare.
- You have no way of correcting the error

Checked exceptions in Java do not always follow these rules; hard luck for us.

Catching and throwing example

```
while(!ok){
    System.out.println("enter a number");
    String s=sc.next();
    ok=true;







}
```

- What is ignored and why?

Catching and throwing example

```
while(!ok){
    System.out.println("enter a number");
    String s=sc.next();
    ok=true;
    try{ double d=Double.parseDouble(s); }




}
```

- What is ignored and why?

Catching and throwing example

```
while(!ok){
    System.out.println("enter a number");
    String s=sc.next();
    ok=true;
    try{ double d=Double.parseDouble(s); }
    catch (NumberFormatException e){
        System.out.println("Not a number!");
        ok=false;
    }
}
```

- What is ignored and why?

Catching and throwing example

```
while(!ok){
    System.out.println("enter a number");
    String s=sc.next();
    ok=true;
    try{ double d=Double.parseDouble(s); }
    catch (NumberFormatException e){
        System.out.println("Not a number!");
        ok=false;
    }
}
```

- What is ignored and why?