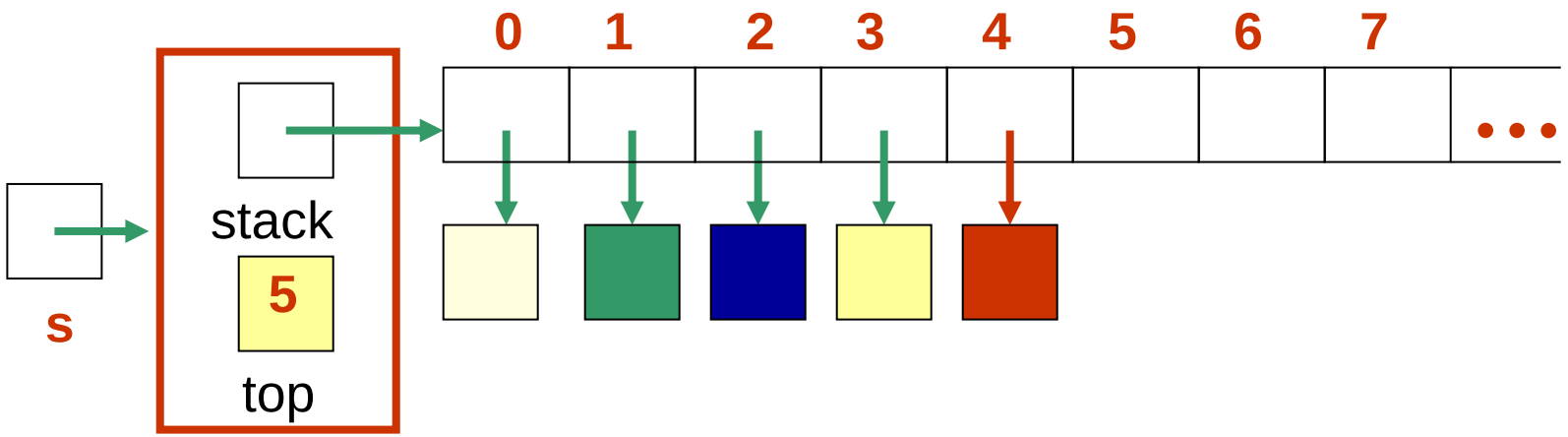
Introduction to Exceptions in Java

Recall the Array Implementation of a Stack



Incorrect implementation of the pop operation in class ArrayStack:

```
public T pop( ) {
    top--;
   T result = stack[top];
   stack[top] = null;
   return result;
}
```

Let the calling method be this:

```
public class TestStack {
   public static void main (String[] args) {
      ArrayStack<String> s = new ArrayStack<String>(5);
      String line;
      line = s.pop();
            . . .
      }
   }
}
```

```
public class TestStack {

   public static void main (String[] args) {
      ArrayStack<String> s = new ArrayStack<String>();
      String line;
      line = s.pop();
          . . .
      }
   }
```

When we run this program we get this error message:

```
Exception in thread “main” java.lang.ArrayIndexOutOfBoundsException: -1
        at ArrayStack.pop(ArrayStack.java:87)
        at TestStack.main(TestStack.java:6)
```

We can fix the problem by using Java Exceptions:

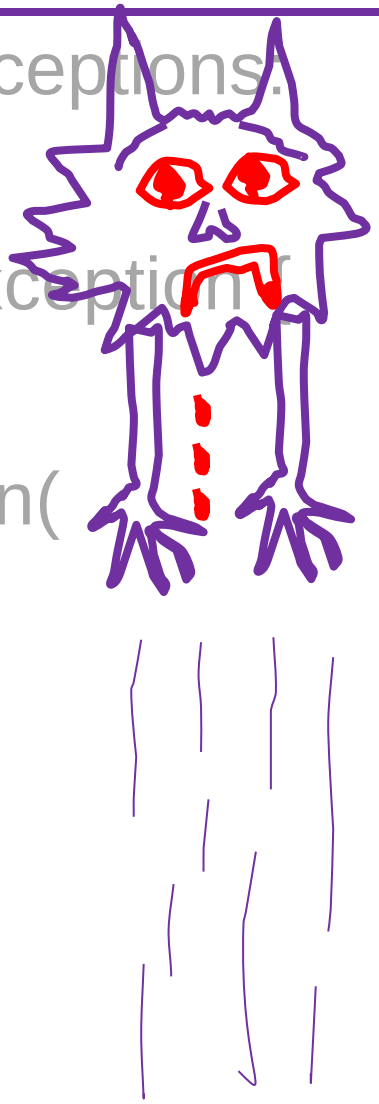
```
public T pop( ) throws EmptyCollectionException {
   if (isEmpty( ))
        throw new EmptyCollectionException(“Stack” );
   top--;
   T result = stack[top];
   stack[top] = null;
   return result;
}
```

We can fix the problem by using Java Exceptions:

```
public T pop( ) throws EmptyCollectionException {
   if (isEmpty( ))
        throw new EmptyCollectionException(
                                        "The Stack is empty");
   top--;
   T result = stack[top];
   stack[top] = null;
   return result;
}
```

We can fix the problem by using Java Exceptions:

```
public T pop( ) throws EmptyCollectionException {
   if (isEmpty( ))
        throw new EmptyCollectionException(
                        “The Stack is empty” );
   top--;
   T result = stack[top];
   stack[top] = null;
   return result;
}
```



Ignorig Exceptions

```
public class TestStack {

   public static void main (String[] args) {
      ArrayStack<String> s = new ArrayStack<String>();
      String line;
      line = s.pop();
            . . .
      }
   }
}
```

When running this program we get this error:

```
Exception in thread “main” EmptyStackException: The Stack is empty.
            at ArrayStack.pop(ArrayStack.java.76)
            at TestStack.main(TestStack.java:6)
```

Catching and re-Throwing Exceptions

The calling method can either catch an exception or it can re-throw it.

- The method catches the exception if it know how to deal with the error.
- Otherwise the exception is re-thrown.

Catching Exceptions

```
public class TestStack {

   public static void main (String[] args) {
      ArrayStack<String> s = new ArrayStack<String>();
      String line;
      try {
         line = s.pop();
            . . .
      }
      catch (EmptyStackException e) {
         // Error handling code
         System.out.println (e.getMessage());
      }
      . . .
   }
}
```

```
public T pop( ) throws EmptyCollectionException {
   if (isEmpty( )) throw
        new EmptyCollectionException("Empty Stack" );
   top--;
   T result = stack[top];
   stack[top] = null;
   return result;
}
```

A Try-Catch Example with Multiple Catch Statements

The try-catch syntax:

```
try {
    code
}
catch(exception1 e) {statements}
catch(exception2 e) {statements}
catch(exception3|exception4 e){statements}
```

Re-Throwing Exceptions

```
public class TestStack {
   private static void helper (ArrayStack<String> s) throws
                                                EmptyStackException {
        String line = s.pop();
        . . .
   }
   public static void main (String[] args) {
      ArrayStack<String> s = new ArrayStack<String>();
      String line;
      try {
         helper(s);
            . . .
      }
      catch (EmptyStackException e) {
         System.out.println (“Stack empty”+e.getMessage());
      }
      . . .
   }
```

```
ArrayStack s;
FileReader file;
  private void helper (ArrayStack<String> s) throws EmptyStackException {
      int c = file.read();
      . . .
      String line = s.pop();
      . . .
  }
  private void method1 () {
    file = new FileReader("file1.txt");
    try {
      helper(s);
    }
    catch (EmptyStackException e) {
      System.out.println ("file1.txt has wrong format");
    }
  }
 private void method2 () {
    file = new FileReader("file2.txt");
    try {
      helper(s);
    }
    catch (EmptyStackException e) {
      System.out.println ("file2.txt has wrong format");
    }
```

Declaring Exception Classes

```
public class EmptyStackException extends
                                        RuntimeException {


  public EmptyStackException (String mssg) {
      super (mssg);
  }

}
```

Runtime Errors

- Java differentiates between ***runtime errors*** and ***exceptions***
  - Errors are unrecoverable situations, so the program must be terminated
    - Example: running out of memory

Exceptions

Exception: an abnormal or erroneous situation at runtime

Examples:

- Division by zero
- Array index out of bounds
- Null pointer exception

Exceptions can be thrown by the program or by the java virtual machine, for example

```
i = size / 0;
```

Checked and Unchecked Exceptions

- Checked exceptions are checked by the compiler
- Unchecked exceptions are not

Example: Checked Exception

```
import java.io.*;

class Main {
    public static void main(String[] args) {
        try {
            FileReader file = new FileReader(“test.txt");
            BufferedReader fileInput = new BufferedReader(file);
            System.out.println(fileInput.readLine());
            fileInput.close();
        }
        catch (FileNotFoundException e) { ... }
        catch (IOException e) { ... }
    }
}
```

Example: Checked Exception

```
import java.io.*;

class Main {
    public static void main(String[] args) {
        FileReader file = new FileReader(“test.txt");
        BufferedReader fileInput = new BufferedReader(file);
        System.out.println(fileInput.readLine());
        fileInput.close();
    }
}
```

The compiler gives the error:

```
Main.java:5: error: unreported exception FileNotFoundException; must be
caught or declared to be thrown
                    FileReader file = new FileReader(“test.txt”);
```

Example: UnChecked Exception

```
class Main {

    public static void main(String[] args) {
        int x = 10;
        int y = 0;
        int z = x /y;
    }
}
```

The compiler does not give an error.

Java Exceptions

- In Java, an exception is an ***object***
- There are Java predefined ***exception classes***
  - ArithmeticException
  - IndexOutOfBoundsException
  - IOException
  - NullPointerException

Some Java Error and Exception Classes

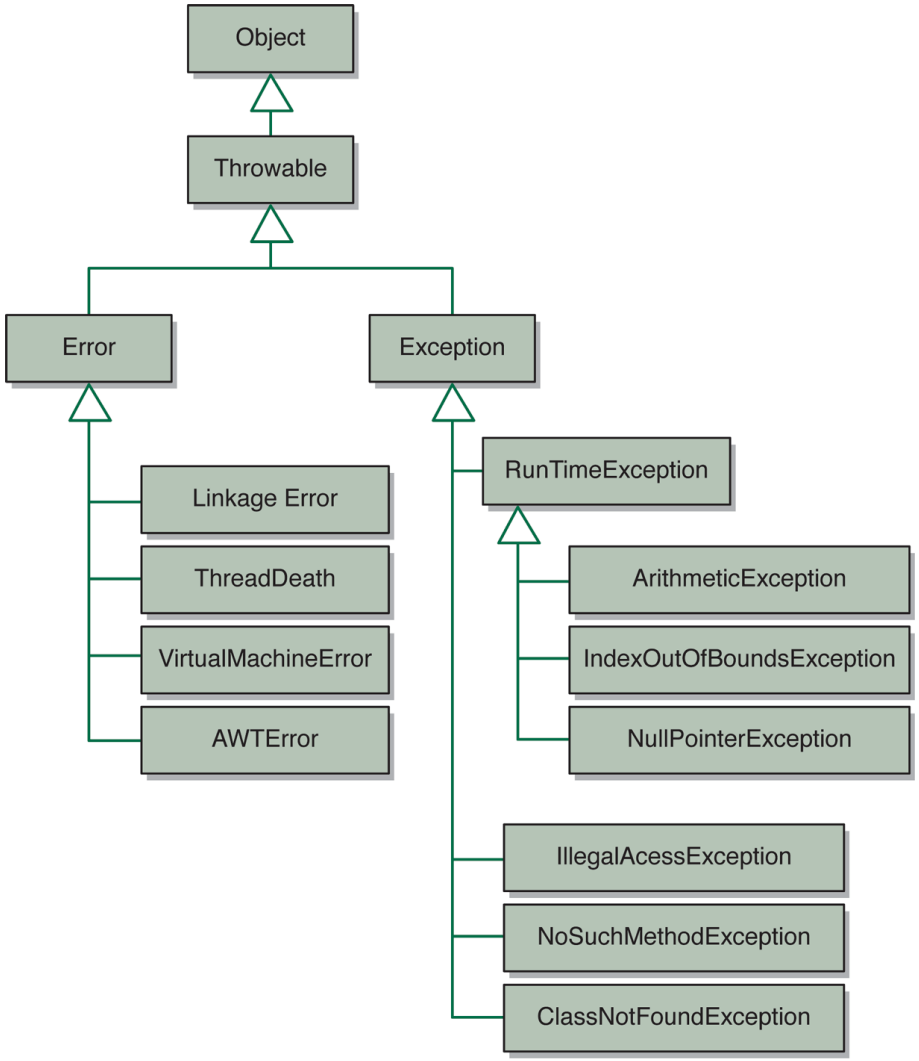


FIGURE 2.7 Part of the `Error` and `Exception` class hierarchy

A Try-Catch Example with Multiple Catch Statements

The try-catch-finally syntax:

```
try {
    code
}
catch(exception1 e) {statements}
catch(exception2 e) {statements}
catch(exception3|exception4 e){statements}
finally {statements}
```

Finally Block

The finally block always executes when the try block exits, whether an exception was thrown or not (even if the exception was not caught by any of the catch statements!)

The finally block is executed even if there is a return statement inside the try or catch blocks or if a new exception is thrown.

```
PrintWriter out;
try {
    out = new PrintWriter(new FileWriter("OutFile.txt"));
    out.println(“Data”);
    int x = 5 / 0;
}
catch(FileNotFoundException e) {…}
catch(IOException e) {…}
if (out != null) out.close();
```

The file will not be closed, so the data will not be stored in it.

```
PrintWriter out = null;
try {
    out = new PrintWriter(new FileWriter("OutFile.txt"));
    out.println(“Data”);
    int x = 5 / 0;
}
catch(FileNotFoundException e) {…}
catch(IOException e) {…}
finally  {
  if (out != null) out.close();
}
```

The file will be closed.