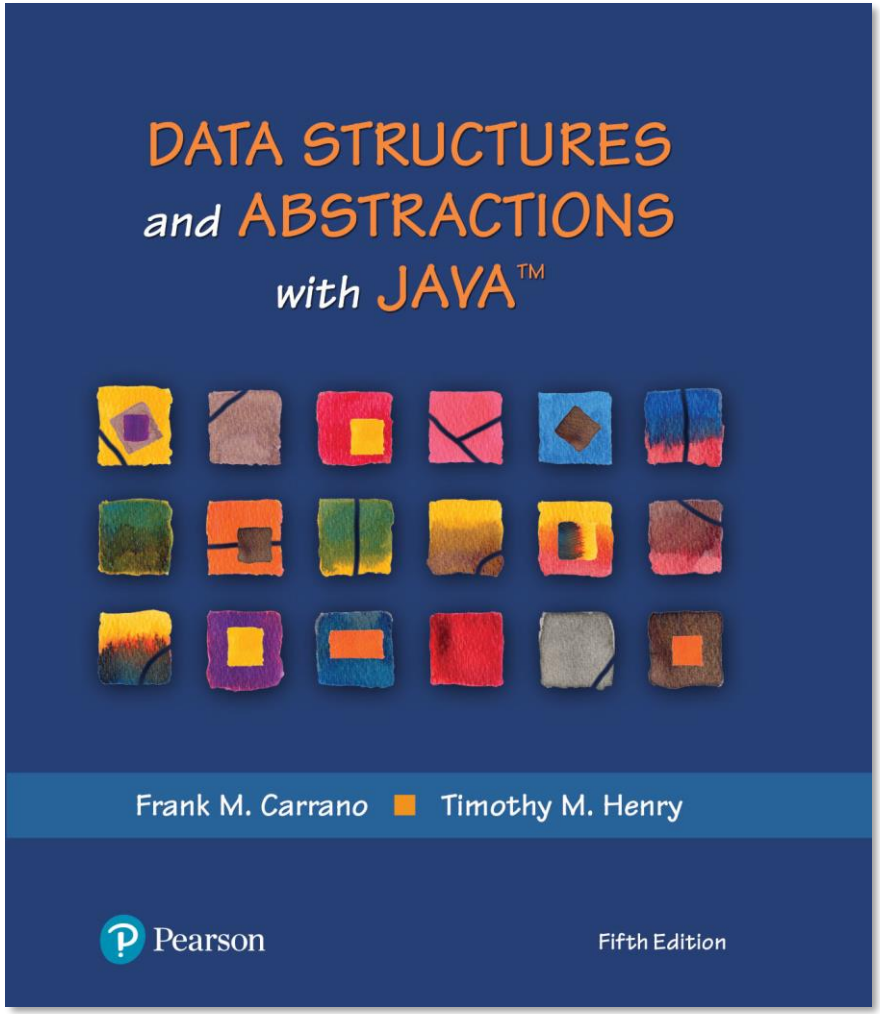
Data Structures and Abstractions with Java™

5th Edition



Java Interlude 3

More About Exceptions

Programmer-Defined Exception Classes

- Define your own exception classes by extending existing exception classes
  - Existing superclass could be one in the Java Class Library or one of your own
- Constructors in an exception subclass are the most important
  - Often the only methods you need to define

Programmer-Defined Exception Classes

```
/** A class of runtime exceptions thrown when an attempt
   is made to find the square root of a negative number. */
public class SquareRootException extends RuntimeException
{
   public SquareRootException()
   {
      super("Attempted square root of a negative number.");
   } // end default constructor

   public SquareRootException(String message)
   {
      super(message);
   } // end constructor
} // end SquareRootException
```

LISTING JI3 -1 The exception class `SquareRootException`

Programmer-Defined Exception Classes

```
public class OurMath
{
  /** Computes the square root of a nonnegative real number.
     @param value  A real value whose square root is desired.
     @return  The square root of the given value.
     @throws  SquareRootException if value < 0. */
  public static double squareRoot(double value) throws SquareRootException
  {
    if (value < 0)
      throw new SquareRootException();
    else
      return Math.sqrt(value);
  } // end squareRoot

  // < Other methods not relevant to this discussion are here. >

} // end OurMath
```

LISTING JI3-2 The class `OurMath` and its static method `squareRoot`

Programmer-Defined Exception Classes

```
/**  A demonstration of a runtime exception using the class OurMath. */
public class OurMathDriver
{
  public static void main(String[] args)
  {
    System.out.print("The square root of 9 is ");
    System.out.println(OurMath.squareRoot(9.0));

    System.out.print("The square root of -9 is ");
    System.out.println(OurMath.squareRoot(-9.0));

    System.out.print("The square root of 16 is ");
    System.out.println(OurMath.squareRoot(16.0));
  } // end main
} // end OurMathDriver
```

Program Output

```
The square root of 9 is 3.0
The square root of -9 is Exception in thread "main" SquareRootException:
Attempted square root of a negative number.
at OurMath.squareRoot(OurMath.java:14)
at OurMathDriver.main(OurMathDriver.java:12)
```

LISTING JI3 -3 A driver for the class `OurMath`

Programmer-Defined Exception Classes

```
/**  A class of static methods to perform various mathematical
   computations, including the square root. */
public class JoeMath
{
  /** Computes the square root of a real number.
      @param value  A real value whose square root is desired.
      @return  A string containing the square root. */
  public static String squareRoot(double value)
  {
    String result = "";
    try
    {
      Double temp = OurMath.squareRoot(value);
      result = temp.toString();
    }
    catch (SquareRootException e)
    {
      Double temp = OurMath.squareRoot(-value);
      result = temp.toString() + "i";
    }

    return result;
  } // end squareRoot

  // < Other methods not relevant to this discussion could be here. >
} // end JoeMath
```

LISTING JI3 -4 The class JoeMath

Programmer-Defined Exception Classes

```
/** A demonstration of a runtime exception using the class JoeMath. */
public class JoeMathDriver
{
  public static void main(String[] args)
  {
    System.out.print("The square root of 9 is ");
    System.out.println(JoeMath.squareRoot(9.0));

    System.out.print("The square root of -9 is ");
    System.out.println(JoeMath.squareRoot(-9.0));

    System.out.print("The square root of 16 is ");
    System.out.println(JoeMath.squareRoot(16.0));

    System.out.print("The square root of -16 is ");
    System.out.println(JoeMath.squareRoot(-16.0));
  } // end main
} // end JoeMathDriver
```

Program Output

```
The square root of 9 is 3.0
The square root of -9 is 3.0i
The square root of 16 is 4.0
The square root of -16 is 4.0i
```

LISTING JI3 -5 A driver for the class JoeMath

Inheritance and Exceptions

```
public class SuperClass
{
   public void someMethod() throws Exception1
   {
   } // end someMethod
} // end SuperClass


public class SubClass extends SuperClass
{
   public void someMethod() throws Exception1, Exception2 // ERROR!
   {
   } // end someMethod
} // end SubClass
```

Consider this superclass and subclass — cannot override `someMethod` in a subclass and list additional checked exceptions in its throws clause

Inheritance and Exceptions

```
public class Driver
{
  public static void main(String[] args)
  {
    SuperClass superObject = new SubClass();
    try
    {
      superObject.someMethod();
    }
    catch (Exception1 e)
    {
      System.out.println(e.getMessage());
    }
  } // end main
} // end Driver
```

Only `Exception1` is caught. If the throws clause in `SubClass` was legal, we could call `SubClass`'s `someMethod` without catching `Exception2`.

Inheritance and Exceptions

```
public class SuperClass
{
   public void someMethod() throws Exception1
   {
   } // end someMethod
} // end SuperClass
```

```
public class SubClass extends SuperClass
{
   public void someMethod() throws Exception2 // OK, assuming Exception2
   {                                  // extends Exception1

   } // end someMethod
} // end SubClass
```

If `Exception2` extends `Exception1`, the above is legal

The finally Block

```
try
{
< Code that might throw an exception, either by executing a throw statement or by calling a
method that throws an exception >
}
catch (AnException e)
{
< Code that handles exceptions of type AnException or a subclass of AnException >
}
< Possibly other catch blocks to handle other types of exceptions >
finally
{
< Code that executes after either the try block or an executing catch block ends >
}
```

This code shows the placement of the `finally` block

The `finally` Block

```
public static void main(String[] args)
{
      try
  {
    openRefrigerator();
    takeOutMilk();
    pourMilk();
    putBackMilk();
  }
  catch (NoMilkException e)
  {
    System.out.println(e.getMessage());
  }
  finally
  {
    closeRefrigerator();
  }
} // end main
```

Whether an exception occurs or not, `closeRefrigerator` is called within the `finally` block.

The `finally` Block

```
  public static void openRefrigerator()
  {
    System.out.println("Open the refrigerator door.");
  } // end openRefrigerator

  public static void takeOutMilk() throws NoMilkException
  {
    if (Math.random() < 0.5)
      System.out.println("Take out the milk.");
    else
      throw new NoMilkException("Out of Milk!");
  } // end openRefrigerator

  // < The methods pourMilk, putBackMilk,
          and closeRefrigerator are analogous to
  //   openRefrigerator and are here. >
  // . . .
} // end GetMilk
```

LISTING JI3-6 A demonstration of a `finally` block

The `finally` Block

Sample Output 1 (no exception is thrown)

```
Open the refrigerator door. Take out the milk.
Pour the milk.
Put the milk back.
```

Sample Output 2 (exception is thrown)

```
Open the refrigerator door. Out of milk!
Close the refrigerator door.
```

LISTING JI3-6 A demonstration of a finally block output

End

Java Interlude 3