



Exception Handling

Chapter 13

Objectives

- Describe the notion of exception handling
- React correctly when certain exceptions occur
- Use Java's exception-handling facilities effectively in classes and programs

Basic Exception Handling: Outline

- Exceptions in Java
- Predefined Exception Classes

Exceptions in Java

- An exception is an object
 - Signals the occurrence of unusual event during program execution
- Throwing an exception
 - Creating the exception object
- Handling the exception
 - Code that detects and deals with the exception

Exceptions in Java

- Consider a program to assure us of a sufficient supply of milk
- View [possible solution](#), listing 13.1
class GotMilk

Enter number of donuts:

2

Enter number of glasses of milk:

0

No milk!

Go buy some milk.

End of program.

Sample
screen
output

Exceptions in Java

- Now we revise the program to use exception-handling
- View [new version](#), listing 13.2
class ExceptionDemo

Enter number of donuts:

3

Enter number of glasses of milk:

2

3 donuts.

2 glasses of milk.

You have 1.5 donuts.

End of program.

Sample
screen

Enter number of donuts:

2

Enter number of glasses of milk:

0

Exception: No milk!

Go buy some milk.

End of program.

Sample
screen
output 2

Exceptions in Java

- Note try **block**
 - Contains code where something could possibly go wrong
 - If it does go wrong, we *throw an exception*
- Note **catch** block
 - When exception thrown, **catch** block begins execution
 - Similar to method with parameter
 - Parameter is the thrown object

Exceptions in Java

- Note flow of control when no exception is thrown
- View [demo with no exception](#), listing 13.3
class ExceptionDemo

```
Enter number of donuts:  
3  
Enter number of glasses of milk:  
2  
3 donuts.  
2 glasses of milk.  
You have 1.5 donuts for each glass of milk.  
End of program.
```

Sample
screen output with
no exception

Exceptions in Java

- Note flow of control when exception IS thrown
- View [demo with exception](#), listing 13.4
class ExceptionDemo

```
Enter number of donuts:  
2  
Enter number of glasses of milk:  
0  
Exception: No milk!  
Go buy some milk.  
End of program.
```

Sample
screen output when
exception is thrown

Predefined Exception Classes

- Java has predefined exception classes within Java Class Library
 - Can place method invocation in **try** block
 - Follow with **catch** block for this type of exception
- Example classes
 - **BadStringOperationException**
 - **ClassNotFoundException**
 - **IOException**
 - **NoSuchMethodException**

Predefined Exception Classes

- Example code

```
SampleClass object = new SampleClass();
try
{
    <Possibly some code>
    object.doStuff(); //may throw IOException
    <Possibly some more code>
}
catch(IOException e)
{
    <Code to deal with the exception, probably including the following:>
    System.out.println(e.getMessage());
}
```

Defining Your Own Exception Classes

- Must be derived class of some predefined exception class
 - Text uses classes derived from class **Exception**

- View [sample class](#), listing 13.5

```
class DivideByZeroException  
    extends Exception
```

- View [demo program](#), listing 13.6

```
class DivideByZeroDemo
```

Defining Your Own Exception Classes

- Different runs of the program

```
Enter numerator:  
5  
Enter denominator:  
10  
5/10 = 0.5  
End of program
```

```
Enter numerator:  
5  
Enter denominator:  
0  
Dividing by Zero!  
Try again.  
Enter numerator:  
5  
Enter denominator:  
Be sure the denominator is not zero.  
10  
5/10 = 0.5  
End of program
```

```
Enter numerator:  
5  
Enter denominator:  
0  
Dividing by Zero!  
Try again.  
Enter numerator:  
5  
Enter denominator:  
Be sure the denominator is not zero.  
0  
I cannot do division by zero.  
Since I cannot do what you want,  
the program will now end.
```

Sample
screen
output 3

Defining Your Own Exception Classes

- Note method `getMessage` defined in exception classes
 - Returns string passed as argument to constructor
 - If no actual parameter used, default message returned
- The type of an object is the name of the exception class

Defining Your Own Exception Classes

Guidelines

- Use the **Exception** as the base class
- Define at least two constructors
 - Default, no parameter
 - With **String** parameter
- Start constructor definition with call to constructor of base class, using **super**
- Do not override inherited **getMessage**

More About Exception Classes: Outline

- Declaring Exceptions (Passing the Buck)
- Kinds of Exceptions
- Errors
- Multiple Throws and Catches
- The **finally** Block
- Rethrowing an Exception
- Case Study: A Line-Oriented Calculator

Declaring Exceptions

- Consider method where code throws exception
 - May want to handle immediately
 - May want to delay until something else is done
- Method that does not catch an exception
 - Notify programmers with **throws** clause
 - Programmer then given responsibility to handle exception

Declaring Exceptions

- Note syntax for throws clause

```
public Type Method_Name(Parameter_List) throws List_Of_Exceptions  
Body_Of_Method
```

- Note distinction
 - Keyword **throw** used to throw exception
 - Keyword **throws** used in method heading to declare an exception

Declaring Exceptions

- If a method throws exception and exception not caught inside the method
 - Method ends immediately after exception thrown
- A throws clause in overriding method
 - Can declare fewer exceptions than declared
 - But not more
- View [program example](#), listing 13.7
class DoDivision

Kinds of Exceptions

- In most cases, exception is caught ...
 - In a **catch** block ... or
 - Be declared in **throws** clause
- But Java has exceptions you do not need to account for
- Categories of exceptions
 - Checked exceptions
 - Unchecked exceptions

Kinds of Exceptions

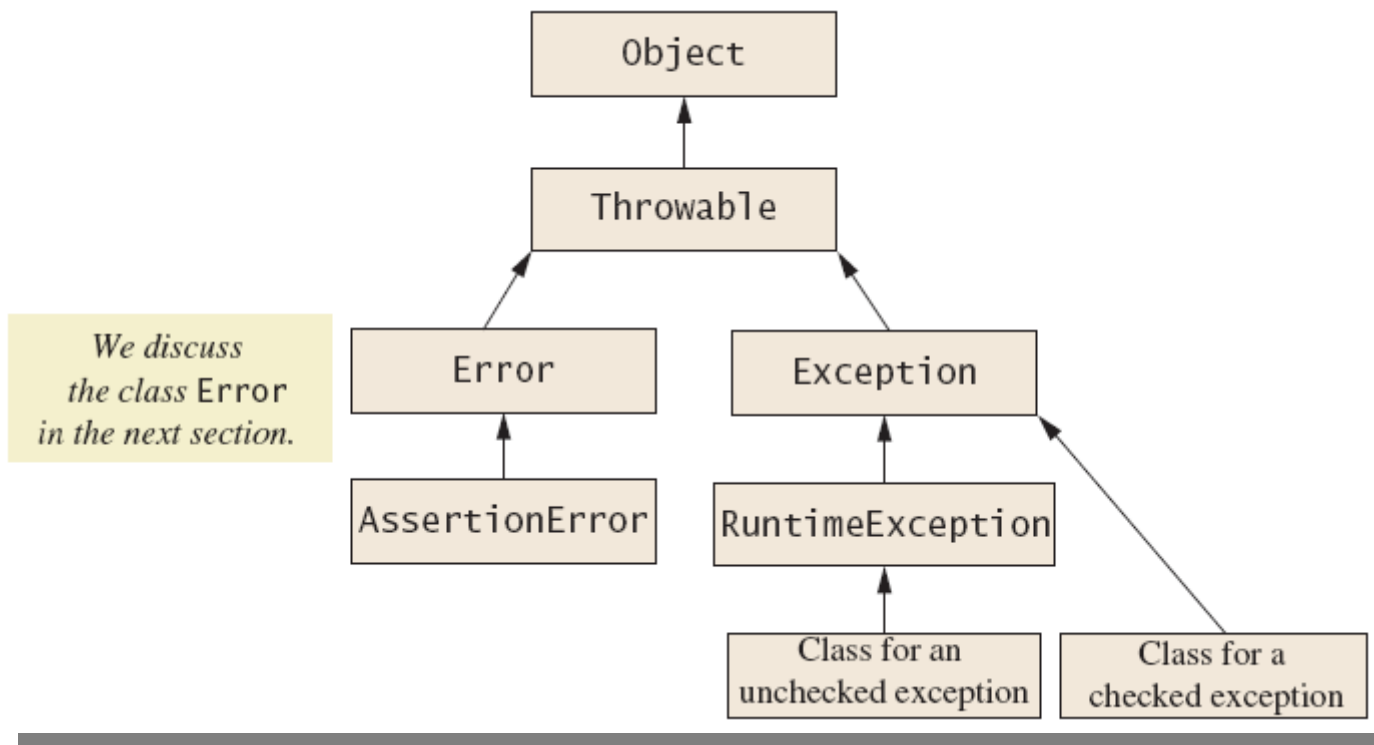
- *Checked* exception
 - Must be caught in **catch** block
 - Or declared in **throws** clause
- *Unchecked* exception
 - Also called *run-time*
 - Need not be caught in **catch** block or declared in **throws**
 - Exceptions that coding problems exist, should be fixed

Kinds of Exceptions

- Examples why unchecked exceptions to are thrown
 - Attempt to use array index out of bounds
 - Division by zero
- Uncaught runtime exception terminates program execution

Kinds of Exceptions

- Figure 13.1 Hierarchy of the predefined exception classes



Errors

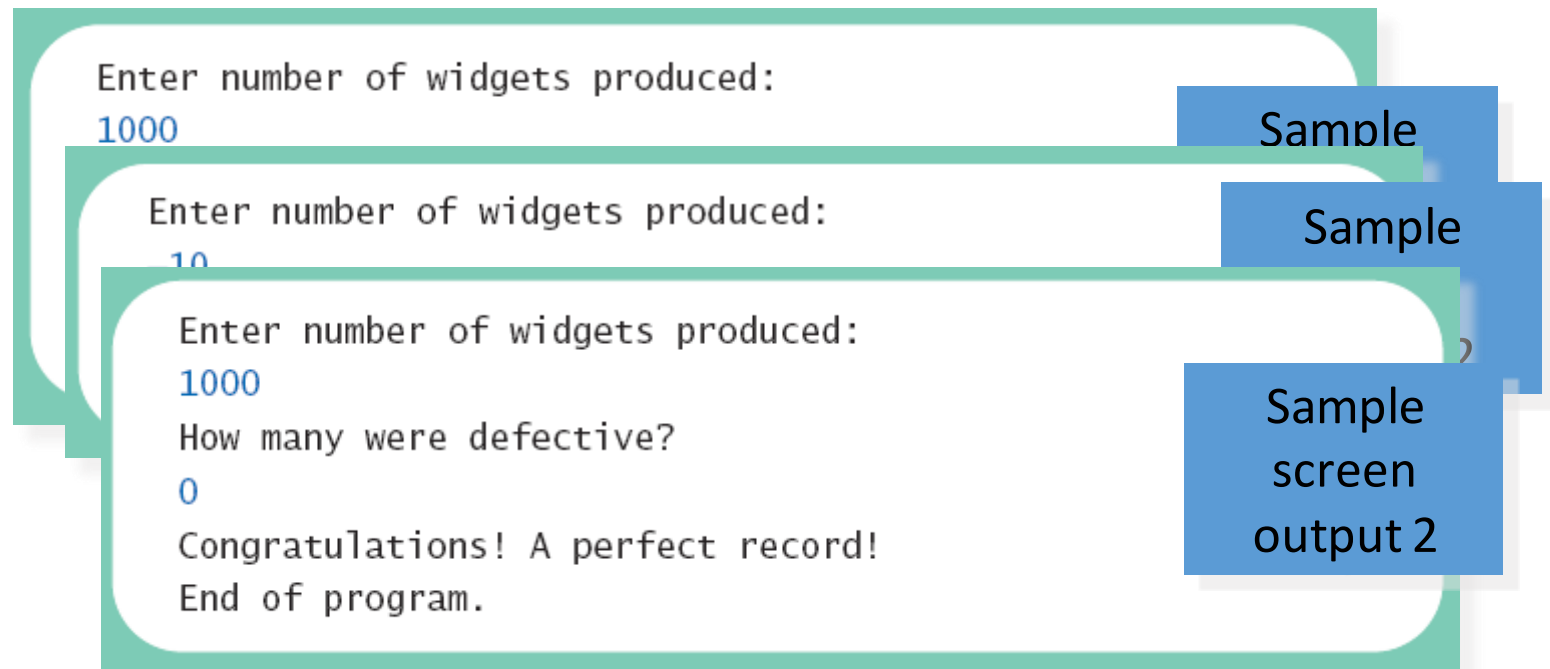
- An *error* is an object of class **Error**
 - Similar to an unchecked exception
 - Need not catch or declare in throws clause
 - Object of class **Error** generated when abnormal conditions occur
- Errors are more or less beyond your control
 - Require change of program to resolve

Multiple Throws and Catches

- A try block can throw any number of exceptions of different types
- Each catch block can catch exceptions of only one type
 - Order of catch blocks matter
- View [example program](#), listing 13.8
class TwoCatchesDemo
- View [exception class](#) used, listing 13.9
class NegativeNumberException

Multiple Throws and Catches

- Note multiple sample runs



The image displays three overlapping sample screen outputs for a Java program, each with a blue label on the right. The top output is labeled 'Sample', the middle one 'Sample', and the bottom one 'Sample screen output 2'. Each output shows a prompt 'Enter number of widgets produced:' followed by a user input in blue. The bottom output also includes a second prompt 'How many were defective?' with input '0', and a final message 'Congratulations! A perfect record! End of program.'

Sample

Sample

Sample screen output 2

Enter number of widgets produced:
1000

Enter number of widgets produced:
10

Enter number of widgets produced:
1000
How many were defective?
0
Congratulations! A perfect record!
End of program.

Multiple Throws and Catches

- Exceptions can deal with invalid user input
- To handle an exception thrown by a method
 - It does not matter where in the method the `throw` occurs
- Use of `throw` statement should be reserved for cases where it is unavoidable
- Text suggests separate methods for throwing and catching of exceptions
- Nested try-catch blocks rarely useful

The **finally** Block

- Possible to add a **finally** block after sequence of **catch** blocks
- Code in **finally** block executed
 - Whether or not execution thrown
 - Whether or not required **catch** exists

Rethrowing an Exception

- Legal to throw an exception within a **catch** block
- Possible to use contents of **String** parameter to **throw** same or different type exception

Case Study

- A Line-Oriented Calculator
 - Should do addition, subtraction, division, multiplication
 - Will use line input/output
- User will enter
 - Operation, space, number
 - Calculator displays result

Case Study

- Proposed initial methods
 - Method to **reset** value of **result** to zero
 - Method to **evaluate** result of one operation
 - Method **doCalculation** to perform series of operations
 - Accessor method **getResult**: returns value of instance variable **result**
 - Mutator method **setResults**: sets value of instance variable **result**

Case Study

- View [exception class](#), listing 13.10
class UnknownOpException
- View first [version of calculator](#), listing 13.11
class PrelimCalculator

```
Calculator is on.  
Format of each line: operator space number  
For example: + 3  
To end, enter the letter e.  
result = 0.0  
+ 4  
result + 4.0 = 4.0  
updated result = 4.0  
* 2  
result * 2.0 = 8.0  
updated result = 8.0  
e  
The final result is 8.0  
Calculator program ending.
```

Sample
screen
output

Case Study

- Final version adds exception handling
- Ways to handle unknown operator
 - Catch exception in method `evaluate`
 - Let `evaluate` throw exception, catch exception in `doCalculation`
 - Let `evaluate`, `doCalculation` both throw exception, catch in `main`
- Latter option chosen

Case Study

- View [final version](#), listing 13.12

class Calculator

Calculator is on.

% 4

-2

result - 2.0 = 78.0

updated result = 78.0

* 0.04

result * 0.04 = 3.12

updated result = 3.12

e

The final result is 3.12

Calculator program ending.

Sample
screen
output

Summary

- An exception is an object derived from class **Exception**
 - Descendants of class **Error** behave like exceptions
- Exception handling allows design of normal cases separate from exceptional situations
- Two kinds of exceptions
 - Checked and unchecked

Summary

- Exceptions can be thrown by
 - Java statements
 - Methods from class libraries
 - Programmer use of **throw** statement
- Method that might **throw** but not **catch** an exception should use **throws** clause
- Exception is caught in **catch** block

Summary

- A **try** block followed by one or more **catch** blocks
 - More specific exception **catch** types should come first
- Every exception type has **getMessage** method usable to recover description of caught exception
- Do not overuse exceptions