



Unit-06

Exception Handling





Outline

- ✓ Exception
- ✓ Using try and catch
- ✓ Nested try statements
- ✓ The Exception Class Hierarchy
- ✓ throw statement
- ✓ throws statement

Exceptions

- ❑ An **exception** is an object that describes an **unusual** or **erroneous situation**.
- ❑ Exceptions are thrown by a program, and may be caught and handled by another part of the program.
- ❑ Java has a predefined set of exceptions and errors that can occur during execution.
- ❑ A program can deal with an exception in one of three ways:
 - ❑ **ignore** it
 - ❑ **handle** it where it occurs
 - ❑ handle it at **another place** in the program

Using try and catch

□ Example:

```
try{  
    // code that may cause exception  
}  
catch(Exception e){  
    // code when exception occurred  
}
```

Exception is the superclass of all the exception that may occur in Java

□ Multiple catch:

```
try{  
    // code that may cause exception  
}  
catch(ArithmeticException ae){  
    // code when arithmetic exception occurred  
}  
catch(ArrayIndexOutOfBoundsException aiobe){  
    // when array index out of bound exception occurred  
}
```

Nested try statements

```
try
{
    try
    {
        // code that may cause array index out of bound exception
    }
    catch(ArrayIndexOutOfBoundsException aiobe)
    {
        // code when array index out of bound exception occurred
    }
    // other code that may cause arithmetic exception
}
catch(ArithmeticException ae)
{
    // code when arithmetic exception occurred
}
```

Types of Exceptions

- An exception is either checked or unchecked.

- **Checked** Exceptions

- A checked exception either must be caught by a method, or must be listed in the throws clause of any method that may throw or propagate it.
- The compiler will issue an error if a checked exception is not caught or asserted in a throws clause
- Example: IOException, SQLException etc...

- **Unchecked** Exceptions

- An unchecked exception does not require explicit handling, though it could be processed using try catch.
- The only unchecked exceptions in Java are objects of type RuntimeException or any of its descendants.
- Example: ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException etc..

Java's Inbuilt Unchecked Exceptions

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.

Java's Inbuilt Checked Exceptions

Exception	Meaning
ClassNotFoundException	Class not found.
IOException	Input Output Exceptions
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

throw statement

- it is possible for your program to throw an exception **explicitly**, using the **throw** statement.
- The general form of throw is shown here:

```
throw ThrowableInstance;
```
- Here, ***ThrowableInstance*** must be an object of type ***Throwable*** or a ***subclass*** of ***Throwable***.

Throw (Example)

```
public class DemoException {  
    public static void main(String[] args) {  
        int balance = 5000;  
  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter Amount to withdraw");  
        int withdraw = sc.nextInt();  
        try {  
            if(balance - withdraw < 1000) {  
                throw new Exception("Balance must be grater than 1000");  
            }  
            else {  
                balance = balance - withdraw;  
            }  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

The finally statement

- The purpose of the **finally** statement will allow the execution of a segment of code regardless if the try statement throws an exception or executes successfully
- The advantage of the **finally** statement is the ability to clean up and release resources that are utilized in the **try** segment of code that might not be released in cases where an exception has occurred.

```
public class MainCall {  
    public static void main(String args[]) {  
        int balance = 5000;  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter Amount to withdraw");  
        int withdraw = sc.nextInt();  
        try {  
            if(balance - withdraw < 1000) {  
                throw new Exception("Balance < 1000 error");  
            }  
            else {  
                balance = balance - withdraw;  
            }  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
        finally {  
            sc.close();  
        }  
    }  
}
```

throws statement

- A throws statement lists the types of exceptions that a **method** might throw.
- This is the general form of a method declaration that includes a **throws clause**:

```
type method-name(parameter-list) throws exception-list {  
    // body of method  
}
```

- Here, *exception-list* is a comma-separated list of the exceptions that a method can throw.
- Example :

```
void myMethod() throws ArithmeticException, NullPointerException  
{  
    // code that may cause exception  
}
```

Create Your Own Exception

- Although Java's built-in exceptions handle most common errors, you will probably want to create your own exception types to handle situations specific to your applications.
- This is quite easy to do: just define a subclass of Exception (which is, of course, a subclass of Throwable).
- The Exception class does not define any methods of its own. It does inherit those methods provided by Throwable.
- Thus, all exceptions have methods that you create and defined by Throwable.

Methods of Exception class

Method	Description
Throwable fillInStackTrace()	Returns a Throwable object that contains a completed stack trace. This object can be rethrown.
Throwable getCause()	Returns the exception that underlies the current exception. If there is no underlying exception, null is returned.
String getMessage()	Returns a description of the exception.
StackTraceElement[] getStackTrace()	Returns an array that contains the stack trace, one element at a time, as an array of StackTraceElement.
Throwable initCause(Throwable causeExc)	Associates causeExc with the invoking exception as a cause of the invoking exception. Returns a reference to the exception.
void printStackTrace()	Displays the stack trace.
void printStackTrace(PrintStream stream)	Sends the stack trace to the specified stream.
void setStackTrace(StackTraceElement elements[])	Sets the stack trace to the elements passed in elements.
String toString()	Returns a String object containing a description of the exception.