Chapter 5 : Exception handling

Introduction

Error is a abnormal condition in a program which make program go wrong.

Types of Errors :

1. Compile time Errors :

Compile time errors are the syntax errors which are detected by the java compiler and gets displayed on the screen at compile time.

Example of compile time errors:

- Missing semicolon
- Command not found

2. Run Time Errors :

Run time errors are the logical errors which can not be detected at compile time of a program.

Example of run time errors :

- Dividing an integer by zero
- converting invalid string to a number

Exception

An exception is an abnormal condition of a program which caused termination of execution of program.

Exception is topmost class in error handling. It stores all the exceptions which are generated by the application.

Types of Exceptions :

1. System defined(in-built)Exception(Checked exceptions)
2. User defined Exception(Unchecked exceptions)

Common Java Exceptions

Exceptions	Description
ArithmeticException	Caused by math errors such as division by zero
ArrayIndexOutOfBounds Exception	Caused by bad array indexes
ArrayStoreException	Caused when a program tries to store the wrong type of data in an array
FileNotFoundException	Caused by an attempt to access a nonexistent file
IOException	Caused by general I/O failures, such as inability to read from a file
NullPointerException	Caused by referencing a null object
NumberFormatException	Caused when a conversion between strings and numbers fails
OutOfMemoryException	Caused when there's not enough memory to allocate a new object
SecurityException	Caused when an applet tries to perform an action not allowed by the browser's security setting
StackOverflowException	Caused when the system runs out of stack space
StringIndexOutOfBoundsException	Caused when a program attempts to access a nonexistent character position in a string

Using try, catch and multiple catch

You protect the code that contains the method that might throw an exception inside a try block. You test for and deal with an exception inside a catch block.

Syntax :

```
try
{

}
catch(Exception_Type Object)
{
}
```

Example :

```
try
{

}
Thread.sleep (1000)
catch (InterruptedException e)
{

}
```

Multiple catch statements

Multiple catch statements are used to handle different types of exceptions.

Syntax :

```
try
{

}
catch()
{
}
catch()
{

}
```

Nested try blocks

A try block can be inside another try block

Syntax :

```
try
{
    try
{ }
    catch()
{
}
}
catch()
{ }
```

throw , throws and finally

The throw Clause

It is used to throw user defined exception in java. It is used with ***try*** and ***catch***

block.

Syntax :

throw new Exception_Type();

So far, you have only been catching exceptions that are thrown by the Java run-time system. However, it is possible for your program to throw an exception explicitly, using the throw statement. The flow of execution stops immediately after the throw statement; any subsequent statements are not executed. The nearest enclosing try block is inspected to see if it has a catch statement that matches the type of the exception.

If it does find a match, control is transferred to that statement. If not, then the next enclosing try statement is inspected, and so on. If no matching catch is found, then the default exception handler halts the program and prints the stack trace.

An exception object can be created using the new operator or simply it is a parameter to a catch clause.

For throwing an exception :

create an instance of an object which is a sub class of java.lang.Throwable.

Use throw keyword to throw an exception.

Syntax : throw throwable instance;

Example :

```
class ThrowDemo { static void
demoproc() { try
{ throw new NullPointerException("demo");
}
catch(NullPointerException e)
{
  System.out.println("Caught inside demoproc."); throw
  e; // rethrow the exception
}
}
```

```
public static void main(String args[]) { try {
demoproc();
}
catch(NullPointerException e)
{
System.out.println ("Recaught: " + e);
}
}
}
```

The throws Clause

The throws Clause is used to indicate that some code in the body of your method may throw an exception,for this simply add the throws keyword after the signature for the method (before the opening brace) with the name or names of the exception that your method throws:

```
public boolean myMethod (int x, int y) throws AnException {
  ...
}
```

If your method may possibly throw multiple kinds of exceptions, you can put all of them in the throws clause, separated by commas:

```
public boolean myOtherMethod (int x, int y)
 throws AnException, AnotherExeption, AThirdException {
  ...
  }
```

Finally Clause

There may be times when you want to be sure that a specific block of code gets executed whether or not an exception is generated. You can do this by adding a finally program block after the last catch. The code in the finally block gets executed after the try block or catch block finishes its thing.

Syntax :

```
try
{
 // The code that may generate an exception goes here.
}
catch (Exception e)
{
// The code that handles the exception goes here.
}
finally
{
// The code here is executed after the try or
 // catch blocks finish executing.
}
```

Creating User Defined Exceptions

Exceptions are simply classes, just like any other classes in the Java hierarchy. Although there are a fair number of exceptions in the Java class library that you can use in your own methods, there is a strong possibility that you may want to create your own exceptions to handle different kinds of errors your programs might run into.

Your new exception should inherit from some other exception in the Java hierarchy. Look for an exception that's close to the one you're creating; for example, an exception for a bad file format would logically be an IOException.

If you can't find a closely related exception for your new exception, consider inheriting from Exception, which forms the "top" of the exception hierarchy for explicit exceptions (remember that implicit exceptions, which include subclasses of Error and RuntimeException, inherit from Throwable).

Exception classes typically have two constructors: The first takes no arguments and the second takes a single string as an argument.

In the latter case you'll want to call super() in that constructor to make sure the string is applied to the right place in the exception.
Beyond those three rules, exception classes look just like other classes. You can put them in their own source files and compile them just as you would other classes:

```
public class SunSpotException extends Exception
{
public SunSpotException()
   {
   }
  public SunSpotExceotion(String msg)
   {
    super(msg);
   }
}
```

Assertion

Assertion is a statement in java. It can be used to test your assumptions about the program.

While executing assertion, it is believed to be true. If it fails, JVM will throw an error named

AssertionError. It is mainly used for testing purpose.

Advantage of Assertion :

It provides an effective way to detect and correct programming errors.

Syntax of using Assertion :

There are two ways to use assertion.

First way is :

assert expression;

and second way is :

assert expression1 : expression2;

Simple Example of Assertion in java :

```
import java.util.Scanner;
classAssertionExample{
public static void main( String args[] ){
Scanner scanner = new Scanner( System.in );
System.out.print("Enter ur age ");
int value = scanner.nextInt();
assert value>=18:" Not valid";
```

```
System.out.println("value is "+value);
}
}
```

Output :

```
Enter ur age 11
Exception in thread "main" java.lang.AssertionError: Not valid
```