Managing Errors and Exceptions

Errors

Errors are the wrongs that can make a program go wrong. An error produces an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash. It is therefore important to detect and manage properly all the possible error conditions in the program so that the program will not terminate or crash during execution.

Types of Errors

Errors may be classified into two categories:

i) Compile-time errors
ii) Run-time errors

i) Compile-time Errors

All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the .class file. It is therefore necessary that we fix all the errors before we can successfully compile and run the program.

Most of the compile-time errors are due to typing mistakes. The most common problems are:

- Missing semicolons
- Missing brackets in classes and methods
- Misspelling of identifiers and keywords
- Use of undeclared variables
- Incompatible types in assignments/ initialization
- Use of = in place of == operator.

ii) Run-time errors

Errors which occur during program execution (run-time) after successful compilation are called run-time errors. One of the most common run-time error is division by zero also known as Division error. These types of error are hard to find as the compiler doesn't point to the line at which the error occurs.

Most common run-time errors are:

- Dividing an integer by zero
- Accessing an element that is out of the bounds of an array
- Trying to store a value into an array of an incompatible class or type
- Attempting to use a negative size of an array

When such errors encountered, java typically generates an error message and aborts the program.

Exceptions

An exception is a condition that is caused by a run-time error in the program. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exceptions object and throws it i.e. it informs us that an error has occurred.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

If we want the program with errors to continue the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and the display an appropriate message for taking corrective actions. This task is known as exception handling. Some common exceptions in java are:

Exception Type	*Cause of Exception*
ArithmeticException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundsException	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 number 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

Exception handling mechanism

The purpose of exception handling is to provide a means to detect and report an "exceptional circumstance" so that appropriate action can be taken. The mechanism perform the following tasks:

i) Find the problem (Hit the exception)
ii) Inform that an error has occur (Throw the exception)

iii) Receive the error information (Catch the exception)

iv) Take corrective action (Handle the exception)

The basic concepts of exception handling are throwing an exception and catching it as illustrated in below figure:

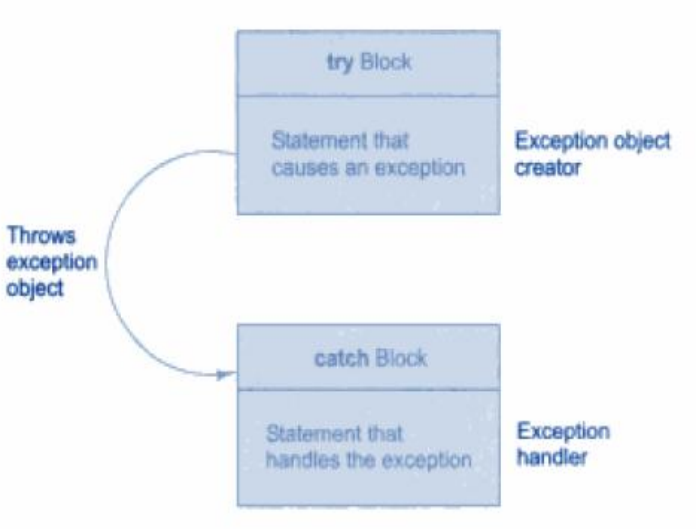

Figure: Exception handling mechanism

Java uses a keyword 'try' to preface a block of code that is likely to cause an error condition and 'throw' an exception. A catch block defined by the keyword 'catch' catches the exception thrown by the try block and handles it appropriately.

The syntax for try and catch block:

```
.....
.....
try
{
    statement;      // generates an exception
}
catch (Exception-type e)
{
    statement;      // processes the exception
}
.....
.....
```

The try block can have one or more statements that could generate an exception. If any one statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block that is placed next to the try block.

Example:

```
public class JavaExceptionExample
{
    public static void main(String args[])
    {
            try
            {
                    //code that may raise exception
                int data=100/0;
            }
            catch(ArithmeticException e)
            {
                    System.out.println(e);
            }
            //rest code of the program
            System.out.println("rest of the code...");
    }
}
```

OUTPUT:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

- **try:** The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
- **catch:** The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
- **finally**: The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
- **throw**: The "throw" keyword is used to throw an exception.
- **throws**: The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method.

Java throw keyword

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or uncheked exception in java by **throw** keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is:

```
throw exception;
```

Example: In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
public class TestThrow1
{
    static void validate(int age)
    {
        if(age<18)
        throw new ArithmeticException("not valid");
        else
        System.out.println("welcome to vote");
    }
    public static void main(String args[])
    {
        validate(13);
        System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
Exception in thread main java.lang.ArithmeticException: not valid
```

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Syntax of java throws

```
return_type method_name() throws exception_class_name
{
    //method code
}
```

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1
{
    void m()throws IOException
    {
        throw new IOException("device error");//checked exception
    }
    void n()throws IOException
    {
        m();
    }
    void p()
    {
        try
        {
            n();
        }catch(Exception e){System.out.println("exception handled");}
    }
    public static void main(String args[])
    {
        Testthrows1 obj=new Testthrows1();
        obj.p();
        System.out.println("normal flow...");
    }
}
```

Output:

```
exception handled
normal flow...
```

Difference between throw and throws in Java

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

	Throw	Throws
i.	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
ii.	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.

iii.	Throw is followed by an instance.	Throws is followed by class.
iv.	Throw is used within the method.	Throws is used with the method signature.
v.	we cannot throw multiple exceptions.	We can declare multiple exceptions e.g. public void method()throws IOException,SQLException.