Exception Handling in Java

SAM

Objectives

- Introduction
- What exceptions are for
- Catching & Throwing exceptions
- Exception Specifications
- Standard Java Exceptions
- Exceptions and Polymorphism
- The **`finally`** clause
- Resource Management
- Uncaught Exceptions

Introduction

- Due to design errors or coding errors, our programs may fail in unexpected ways during execution. An exception is a condition that is caused by run time error in the program. The purpose of the exception handling mechanism is to provide a means to detect and report an “ecxceptional circumstances” .

Error

- An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash. So it is our responsibility to detect and manage the error properly.

Types of error

- Runtime Errors: occur while the program is running if the environment detects an operation that is impossible to carry out.
- Logic Errors: occur when a program doesn't perform the way it was intended
- Syntax Errors: Arise because the rules of the language have not been followed. They are detected by the compiler.

Example of Run Time error

```
Class Error
{
public static void main(String args[])
 {
int a=10;
int b=5;
int c=5;

int x=a/(b+c);
System.out.println("x=" +x);
int y=a/(b-c);   // Errorr division by zero
System.out.println("y=" +y);
}
}
```

Errors and Error Handling

- Some typical causes of errors:
  - Memory errors (i.e. memory incorrectly allocated, memory leaks, “null pointer”)
  - File system errors (i.e. disk is full, disk has been removed)
  - Network errors (i.e. network is down, URL does not exist)
  - Calculation errors (i.e. divide by 0)

Errors and Error Handling

- More typical causes of errors:
  - Array errors (i.e. accessing element –1)
  - Conversion errors (i.e. convert ‘q’ to a number)
  - Can you think of some others?

Errors and Error Handling

- Exceptions – a better error handling
  - Exceptions are a mechanism that provides the best of both worlds.
  - Exceptions act similar to method return flags in that any method may raise and exception should it encounter an error.
  - Exceptions act like global error methods in that the exception mechanism is built into Java; exceptions are handled at many levels in a program, locally and/or globally.

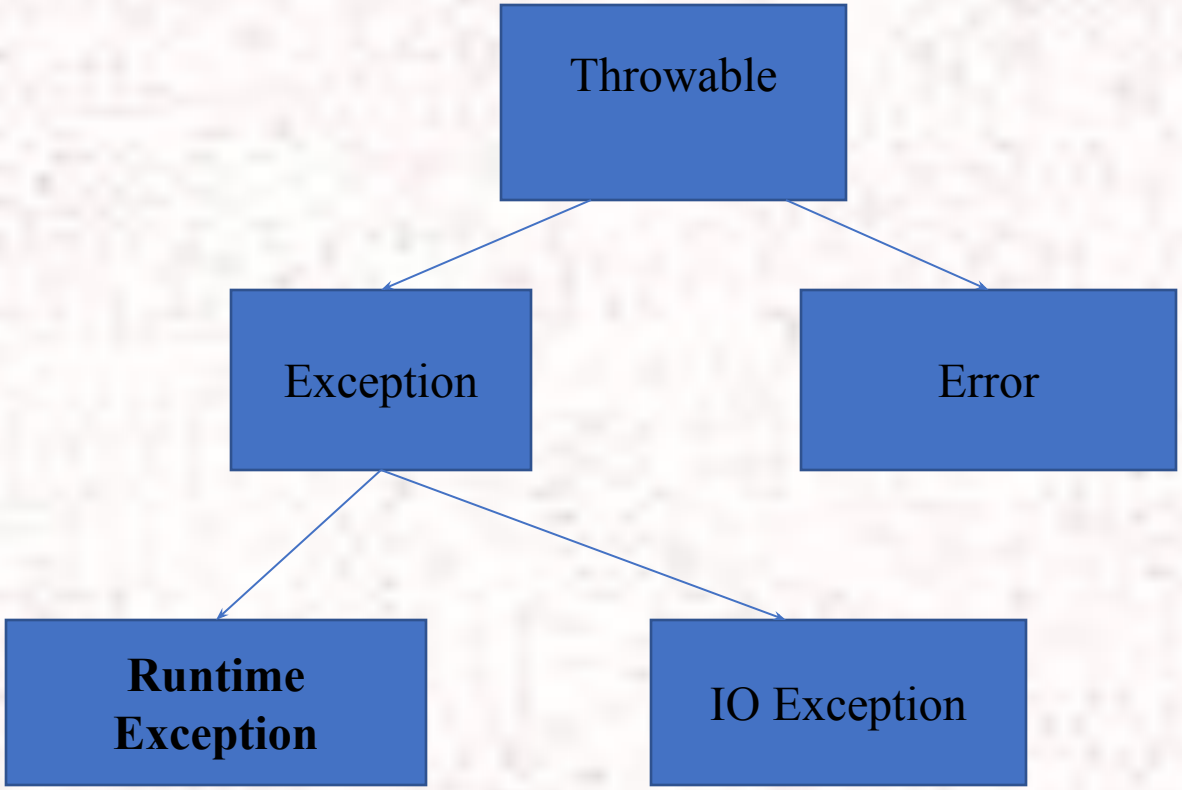
Exceptions

- How do you handle exceptions?
  - To handle the exception, you write a "try-catch" block. To pass the exception "up the chain", you declare a throws clause in your method or class declaration.
  - If the method contains code that may cause a checked exception, you MUST handle the exception OR pass the exception to the parent class (remember, every class has Object as the ultimate parent)

Coding Exceptions

- Coding Exceptions
- Try-Catch Mechanism
  - Wherever your code may trigger an exception, the normal code logic is placed inside a block of code starting with the “try” keyword:
  - After the try block, the code to handle the exception should it arise is placed in a block of code starting with the “catch” keyword.

Standard Java Exceptions



Catching Exceptions

- Wrap code to be checked in a try-block
  - checking occurs all the way down the execution stack
- try-blocks can be nested
  - control resumes at most enclosed matching handler

Coding Exceptions

- Example
  - try {
    … normal program code
    }
    catch(Exception e) {
    … exception handling code
    }

Coding Exceptions

- Types of Exceptions
  - Examples:
    - public void myMethod throws Exception {
    - public void myMethod throws IOException {
    - try { … }
      catch (Exception e) { … }
    - try { … }
      catch (IOException ioe) { … }

Code Examples

- 1. Demonstration of an unchecked exception (NullPointerException)
- 2. Demonstration of checked exceptions:
  - Passing a DivideByZeroException
  - Handling a DivideByZeroException

Example

```
class error2
{
public static void main(String arg[])
{
int a=10;
 int b=5;
 int c=5;
 int x,y;
 try
{
 x=a/(b-c);
}
 catch(ArithmeticException e)
{
System.out.println("Division by Zero");
}
Y=a/(b-c);
System.out.println("y="+y);
}
}
```

In the previous program we cannot see the value of x just because of the error in the value of y, that is division by zero but when we use the try and catch blocks in exception handling then we can see the value of y which is correct and our program will display an error message shown in the try block.

conclusion

- Exceptions are a powerful error handling mechanism.
- Exceptions in Java are built into the language.
- Exceptions can be handled by the programmer (try-catch), or handled by the Java environment (throws).
- Exception handling can only hide the errors.
- It cannot correct the errors.