

## Exceptions: An OO Way for Handling Errors

*Rajkumar Buyya*

Grid Computing and Distributed Systems (GRIDS) Laboratory  
Dept. of Computer Science and Software Engineering  
University of Melbourne, Australia  
<http://www.buyya.com>

1

## Introduction

- Rarely does a program runs successfully at its very first attempt.
- It is common to make mistakes while developing as well as typing a program.
- Such mistakes are categorised as:
  - syntax errors - compilation errors.
  - semantic errors- leads to programs producing unexpected outputs.
  - runtime errors – most often lead to abnormal termination of programs or even cause the system to crash.

2

## Common Runtime Errors

- Dividing a number by zero.
- Accessing an element that is out of bounds of an array.
- Trying to store incompatible data elements.
- Using negative value as array size.
- Trying to convert from string data to a specific data value (e.g., converting string "abc" to integer value).
- File errors:
  - opening a file in "read mode" that does not exist or no read permission
  - Opening a file in "write/update mode" which has "read only" permission.
- Corrupting memory: - common with pointers
- Any more ...

3

## Without Error Handling – Example 1

```
class NoErrorHandling{
    public static void main(String[] args){
        int a,b;
        a = 7;
        b = 0;
        System.out.println("Result is " + a/b);
        System.out.println("Program reached this line");
    }
}
```

Program does not reach here

No compilation errors. While running it reports an error and stops without executing further statements:  
java.lang.ArithmeticException: / by zero at Error2.main(Error2.java:10)

4

## Traditional way of Error Handling - Example 2

```
class WithErrorHandling{
    public static void main(String[] args){
        int a,b;
        a = 7; b = 0;
        if (b != 0){
            System.out.println("Result is " + a/b);
        }
        else{
            System.out.println("B is zero");
        }
        System.out.println("Program is complete");
    }
}
```

Program reaches here

5

## Error Handling

- Any program can find itself in unusual circumstances – **Error Conditions**.
- A "good" program should be able to handle these conditions gracefully.
- Java provides a mechanism to handle these error condition - **exceptions**

6

## Exceptions

- An exception is a condition that is caused by a runtime error in the program.
- Provide a mechanism to signal errors directly without using flags.
- Allow errors to be handled in one central part of the code without cluttering code.

7

## Exceptions and their Handling

- When the JVM encounters an error such as divide by zero, it creates an exception object and throws it – as a notification that an error has occurred.
- If the exception object is not caught and handled properly, the interpreter will display an error and terminate the program.
- If we want the program to continue with execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then take appropriate corrective actions. This task is known as *exception handling*.

8

## Common Java Exceptions

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- FileNotFoundException
- IOException – general I/O failure
- NullPointerException – referencing a null object
- OutOfMemoryException
- SecurityException – when applet tries to perform an action not allowed by the browser's security setting.
- StackOverflowException
- StringIndexOutOfBoundsException

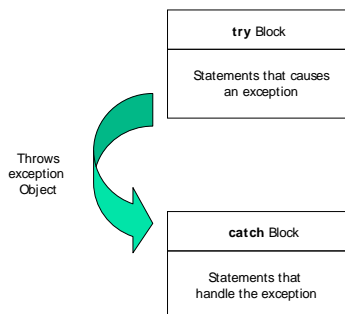
9

## Exceptions in Java

- A method can signal an error condition by throwing an exception – *throws*
- The calling method can transfer control to a exception handler by catching an exception - *try, catch*
- Clean up can be done by - *finally*

10

## Exception Handling Mechanism



11

## Syntax of Exception Handling Code

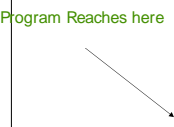
```
...
...
try {
    // statements
}
catch( Exception-Type e)
{
    // statements to process exception
}
..
..
```

12

## With Exception Handling - Example 3

```
class WithExceptionHandling{
    public static void main(String[] args){
        int a,b; float r;
        a = 7; b = 0;
        try{
            r = a/b;
            System.out.println("Result is " + r);
        }
        catch(ArithmeticException e){
            System.out.println(" B is zero");
        }
        System.out.println("Program reached this line");
    }
}
```

Program Reaches here



13

## Finding a Sum of Integer Values Passed as Command Line Parameters

```
// ComLineSum.java: adding command line parameters
class ComLineSum
{
    public static void main(String args[])
    {
        int InvalidCount = 0;
        int number, sum = 0;
        for( int i = 0; i < args.length; i++)
        {
            try {
                number = Integer.parseInt(args[i]);
            }
            catch(NumberFormatException e)
            {
                InvalidCount++;
                System.out.println("Invalid Number: "+ args[i]);
                continue; // skip the remaining part of loop
            }
            sum += number;
        }
        System.out.println("Number of Invalid Arguments = "+ InvalidCount);
        System.out.println("Number of Valid Arguments = "+ (args.length-InvalidCount));
        System.out.println("Sum of Valid Arguments = "+ sum);
    }
}
```

14

## Sample Runs

```
[raj@mundroo] java ComLineSum 1 2
Number of Invalid Arguments = 0
Number of Valid Arguments = 2
Sum of Valid Arguments = 3
```

```
[raj@mundroo] java ComLineSum 1 2 abc
Invalid Number: abc
Number of Invalid Arguments = 1
Number of Valid Arguments = 2
Sum of Valid Arguments = 3
```

15

## Multiple Catch Statements

- If a try block is likely to raise more than one type of exceptions, then multiple catch blocks can be defined as follows:

```
...
...
try {
    // statements
}
catch( Exception-Type1 e)
{
    // statements to process exception 1
}
...
...
catch( Exception-TypeN e)
{
    // statements to process exception N
}
...
```

16

## finally block

- Java supports definition of another block called finally that be used to handle any exception that is not caught by any of the previous statements. It may be added immediately after the try block or after the last catch block:

```
...
try {
    // statements
}
catch( Exception-Type1 e)
{
    // statements to process exception 1
}
...
...
finally {
    ...
}
}
```

- When a finally is defined, it is executed regardless of whether or not an exception is thrown. Therefore, it is also used to perform certain house keeping operations such as closing files and releasing system resources.

17

## Catching and Propagating Exceptions

- Exceptions raised in try block can be caught and then they can be thrown again/propagated after performing some operations. This can be done by using the keyword "throw" as follows:

- throw exception-object;
- OR
- throw new Throwable\_Subclass;

18

## With Exception Handling - Example 4

```
class WithExceptionCatchThrow{
    public static void main(String[] args){
        int a,b; float r; a = 7; b = 0;
        try{
            r = a/b;
            System.out.println("Result is " + r);
        }
        catch(ArithmeticException e){
            System.out.println(" B is zero");
            throw e;
        }
        System.out.println("Program is complete");
    }
}
```

Program Does Not reach here when exception occurs

19

## With Exception Handling - Example 5

```
class WithExceptionCatchThrowFinally{
    public static void main(String[] args){
        int a,b; float r; a = 7; b = 0;
        try{
            r = a/b;
            System.out.println("Result is " + r);
        }
        catch(ArithmeticException e){
            System.out.println(" B is zero");
            throw e;
        }
        finally{
            System.out.println("Program is complete");
        }
    }
}
```

Program reaches here

20

## User-Defined Exceptions

### ■ Problem Statement :

- Consider the example of the Circle class
- Circle class had the following constructor

```
public Circle(double centreX, double centreY,
              double radius){
    x = centreX; y = centreY; r = radius;
}
```

- How would we ensure that the radius is not zero or negative?

21

## Defining your own exceptions

```
import java.lang.Exception;
class InvalidRadiusException extends Exception {

    private double r;

    public InvalidRadiusException(double radius){
        r = radius;
    }
    public void printError(){
        System.out.println("Radius [" + r + "] is not valid");
    }
}
```

22

## Throwing the exception

```
class Circle {
    double x, y, r;

    public Circle (double centreX, double centreY, double
radius ) throws InvalidRadiusException {
        if (r <= 0) {
            throw new InvalidRadiusException(radius);
        }
        else {
            x = centreX; y = centreY; r = radius;
        }
    }
}
```

23

## Catching the exception

```
class CircleTest {
    public static void main(String[] args){
        try{
            Circle c1 = new Circle(10, 10, -1);
            System.out.println("Circle created");
        }
        catch(InvalidRadiusException e)
        {
            e.printError();
        }
    }
}
```

24

## User-Defined Exceptions in standard format

```
class MyException extends Exception
{
    MyException(String message)
    {
        super(message); // pass to superclass if parameter is not handled by used defined exception
    }
}
class TestMyException {
    ...
    try {
        ...
        throw new MyException("This is error message");
    }
    catch(MyException e)
    {
        System.out.println("Message is: "+e.getMessage());
    }
}
```

Get Message is a method defined in a standard Exception class.

25

## Summary

- A good programs does not produce unexpected results.
- It is always a good practice to check for potential problem spots in programs and guard against program failures.
- Exceptions are mainly used to deal with runtime errors.
- Exceptions also aid in debugging programs.
- Exception handling mechanisms can effectively used to locate the type and place of errors.

26

## Summary

- *Try* block, code that could have exceptions / errors
- *Catch* block(s), specify code to handle various types of exceptions. First block to have appropriate type of exception is invoked.
- If no 'local' catch found, exception propagates up the method call stack, all the way to *main()*
- Any execution of try, normal completion, or catch then transfers control on to *finally* block

27