CSE306T

Advanced Java Programming

Exception Handling

B.Tech (CSE)
January, 2025

Nihar Ranjan Roy
Associate Professor, VIPS Engineering
nihar.roy@vips.edu

VIPS
योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

Problem

```
int i=5,j=0;
i=i/j;
```

Arithmetic Exception

```
int arr[]={5,6};
System.out.println(arr[4]);
```

ArrayIndexOutofBound Exception

```
String str="nihar";
int i=Integer.parseInt(str);
```

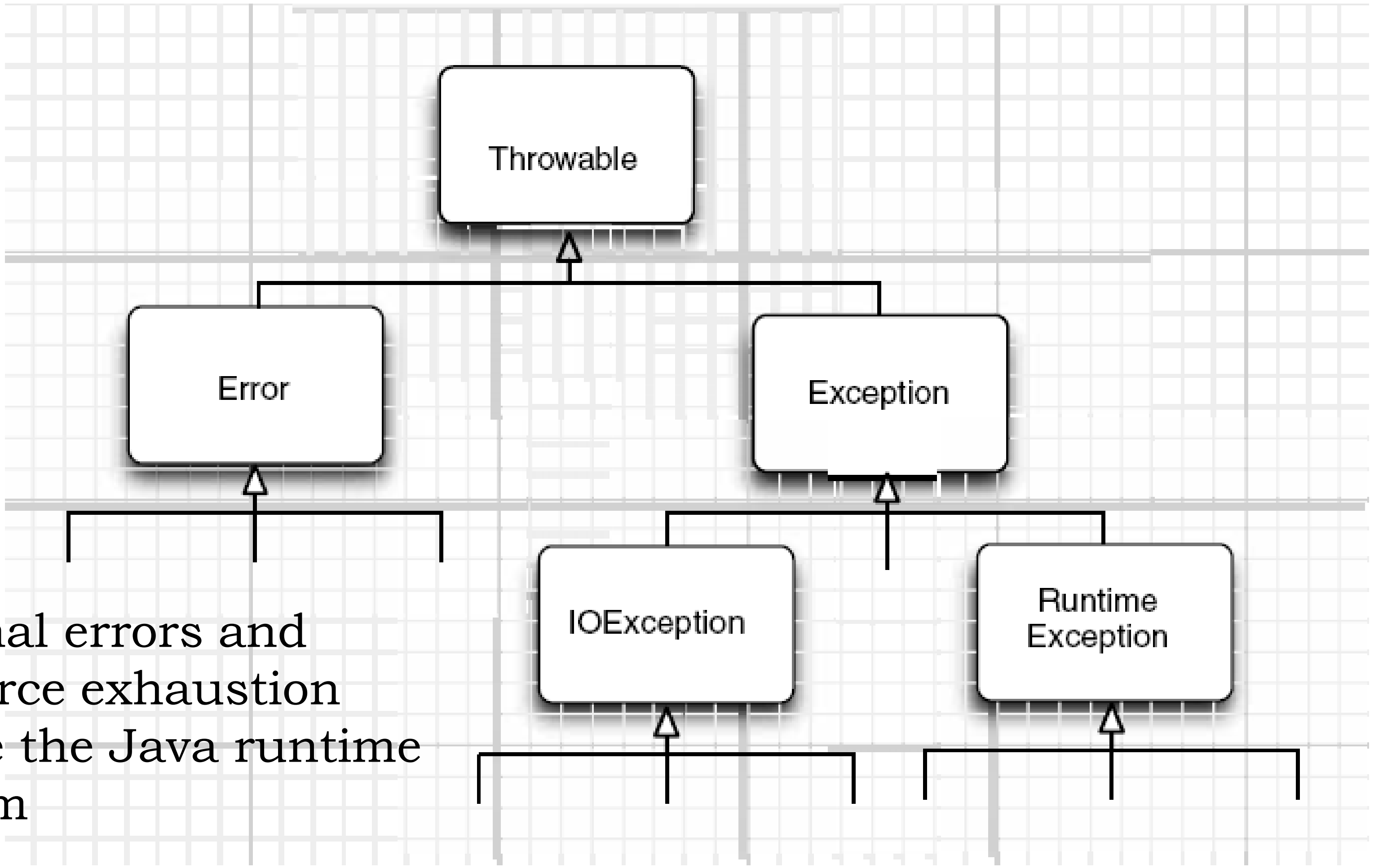
NumberFormat Exception

Result➔ abnormal termination of program

Solution

- Notify the user about the exception
- Save all the work
- Gracefully exit the program

Class hierarchy



internal errors and resource exhaustion inside the Java runtime system

Runtime Exceptions

The term is misleading.

Exceptions that inherit from RuntimeException include such problems as

- A bad cast→ `Object obj="Nihar"; Integer num = (Integer) obj;`
- An out-of-bounds array access→

```
int[] numbers = {1, 2, 3};
System.out.println(numbers[5]);
```

- A null pointer access

```
String str = null;
System.out.println(str.length());
```

"If it is a RuntimeException, it was your fault"

Checked & unchecked Exceptions

Unchecked→ Exception that derives from the **class Error** or the class **RuntimeException** are unchecked exception.

checked→ All others are checked exceptions

The compiler checks that you provide exception handlers for all checked exceptions.

Example

Unchecked

```
int
  arr[]={5,6};
arr[5];
```

```
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
```

Checked

```
br.readLine();
```

If an Exception occurs then

When an exception occurs. JVM creates an object of the corresponding exception type and throws it.

It no one is there to handle it then the program terminates abnormally

try catch blocks

```
 try
{
//code that might throw exception
}catch(ExceptionType1 e)
{
//handler code
}
```

Key words related to exception handling

`try`➔ refers to a block where exception might occur

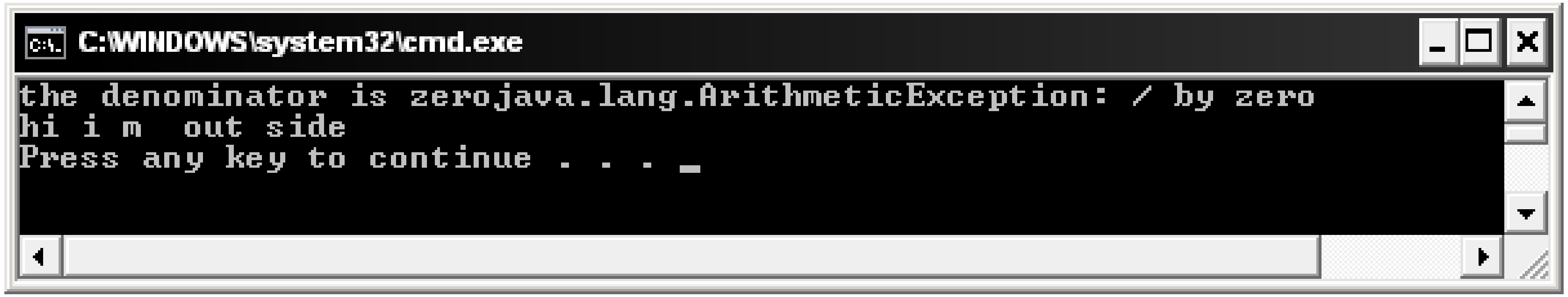
`catch`➔ immediately after try block to catch the exception object

`throw`➔ throws exceptions

`throws`➔ tells the that following exceptions can be thrown by this method and u need to have a handler for this.

`finally`➔ even if an exception occurs execute the code in finally block then exit

```
class ArithmeticEx
{public static void main(String args[])
{int i=5,j=0;
  try{
  i=i/j;
  }
  catch(ArithmeticException e)
  {System.out.println("the denominator is
    zero"+e);}
System.out.println("hi i m  out side");
}}
```



C:\WINDOWS\system32\cmd.exe

```
the denominator is zerojava.lang.ArithmeticException: / by zero
hi i m  out side
Press any key to continue . . . _
```

Multiple catch blocks

```
 try
{

Statement1
Statement 2

}catch(ExceptionType1 e){……}
catch(ExceptionTyp2 e2){……}
```

Finally block

Finally block is executed irrespective of whether an exception has occurred or not.

```
try{………}
catch(ExceptionType e){…….}
finally
{……
……
}
```

throws

```
type method() throws exceptionlist
{
//body
}

    Example
    void calulate() throws ArithmeticException
           {int i=5,j=0;
           i=i/j;
           }
```

throw

- Used to throw exception.
- The exception must be evaluated to an instance of class throwable or it may be a subclass

```
try
{
throw new ArithmeticException(“This is a test”);

}catch(ArithmeticException e){……….}
```

Problem

USER DEFINED EXCEPTION

Create a user defined exception. Which is thrown when the marks of a student is greater then 100 in any subject.

```
class MyException extends Exception
{  MyException(String str)
   {   super(str); }
}

      class TestException
      {public static void main(String args[])
      {

      int marks=101;
      try
      {
              if (marks>100)
              {
              throw new MyException("Marks Cannot be
              greater than 100");
              }
      }catch(MyException e){  System.out.println(e); }
      }}
```

Bank Account Withdrawal with User-Defined Exception

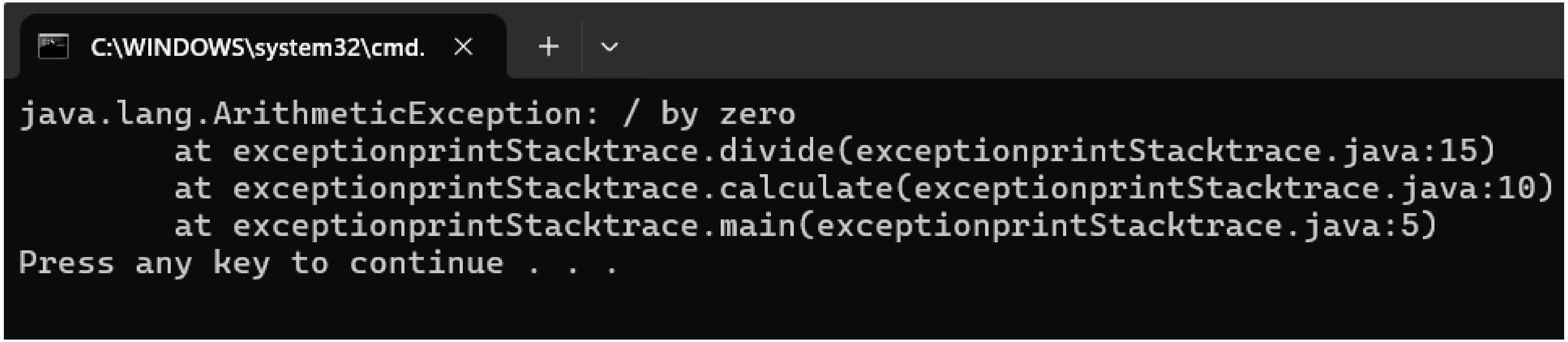
Design and implement a Java program to simulate a bank account withdrawal process. The program should allow users to withdraw money from their account while ensuring that the withdrawal amount does not exceed the available balance.
If a user attempts to withdraw an amount greater than their current balance, the program should throw a **user-defined exception** named InsufficientFundsException. The exception should display an appropriate error message.

Requirements:

1. Define a class `InsufficientFundsException` that extends Exception.
2. Implement a `BankAccount` class with:
   - A `private` field `balance` to store the account balance.
   - A `constructor` to initialize the balance.
   - A method `withdraw(double amount)` that:
     - Deducts the amount if sufficient funds are available.
     - Throws `InsufficientFundsException` if the amount exceeds the balance.
3. Create a main method to:
   - Instantiate a `BankAccount` object with an initial balance.
   - Take a withdrawal amount as input.
   - Handle the exception gracefully using `try-catch`.

e.printStackTrace() method

- In Java, e.printStackTrace() is a method used to print the stack trace of an exception to the standard error stream. It helps in debugging by displaying the sequence of method calls that led to the exception.



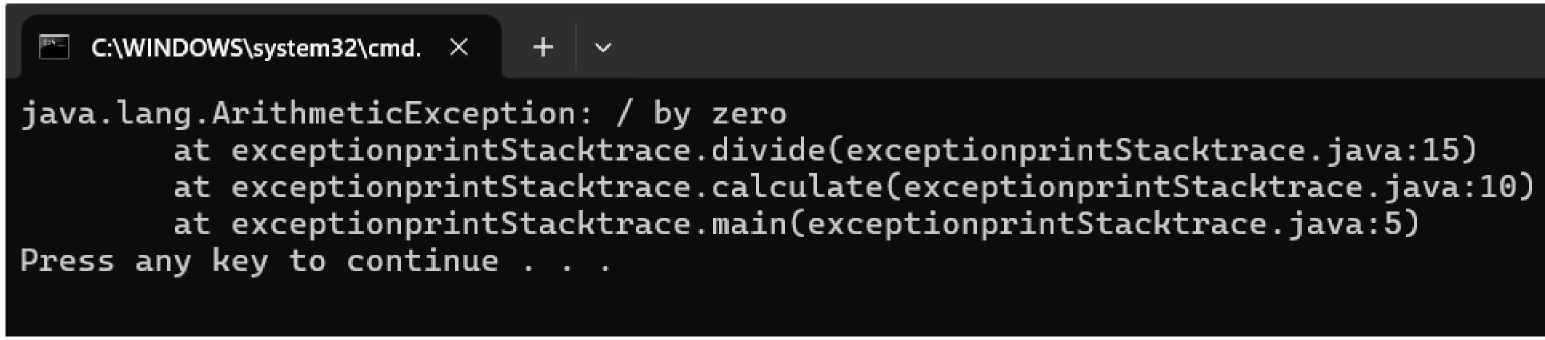
```
java.lang.ArithmeticException: / by zero
        at exceptionprintStacktrace.divide(exceptionprintStacktrace.java:15)
        at exceptionprintStacktrace.calculate(exceptionprintStacktrace.java:10)
        at exceptionprintStacktrace.main(exceptionprintStacktrace.java:5)
Press any key to continue . . .
```

Key Points:

- It prints the class name, exception message, and method call hierarchy.
- The default output is to System.err, but you can also redirect it to logs.

Example

```
class exceptionprintStacktrace
{
public static void main(String args[])
{
calculate();
}
static public void calculate()
{
        divide();
        }
static public void divide()
{       try {
            int result = 10 / 0; // This will cause ArithmeticException
        } catch (Exception e) {
            e.printStackTrace(); // Prints the stack trace
}}}
```



```
java.lang.ArithmeticException: / by zero
        at exceptionprintStacktrace.divide(exceptionprintStacktrace.java:15)
        at exceptionprintStacktrace.calculate(exceptionprintStacktrace.java:10)
        at exceptionprintStacktrace.main(exceptionprintStacktrace.java:5)
Press any key to continue . . .
```

Redirect Exception Error to a file

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class ExceptionToFile {
    public static void main(String[] args) {
        try {
            processFile();
        } catch (Exception e) {
      System.out.println("An error occurred! Check
error_log.txt for details.");
            logExceptionToFile(e);
        }
    }

    static void processFile() throws Exception {
        readFile();
    }
```

```
    static void readFile() throws Exception {
        throw new Exception("File not found!");
    }

    static void logExceptionToFile(Exception e) {
        try (PrintWriter pw = new PrintWriter(new
FileWriter("error_log.txt", true))) {
// Redirect stack trace to file
e.printStackTrace(pw);
        } catch (IOException ioException) {
            System.err.println("Failed to write to log
file: " + ioException.getMessage());
        }
    }
}
```

Thank You