CS20004: Object Oriented Programming using Java

Lec-20

In this Discussion . . .

- Own Exception Handling
  - Nested try statements
  - Throwing Exceptions
  - Creating Exceptions
  - throws Statement
  - finally block
- User-Defined Exceptions
- References

Nested try statements

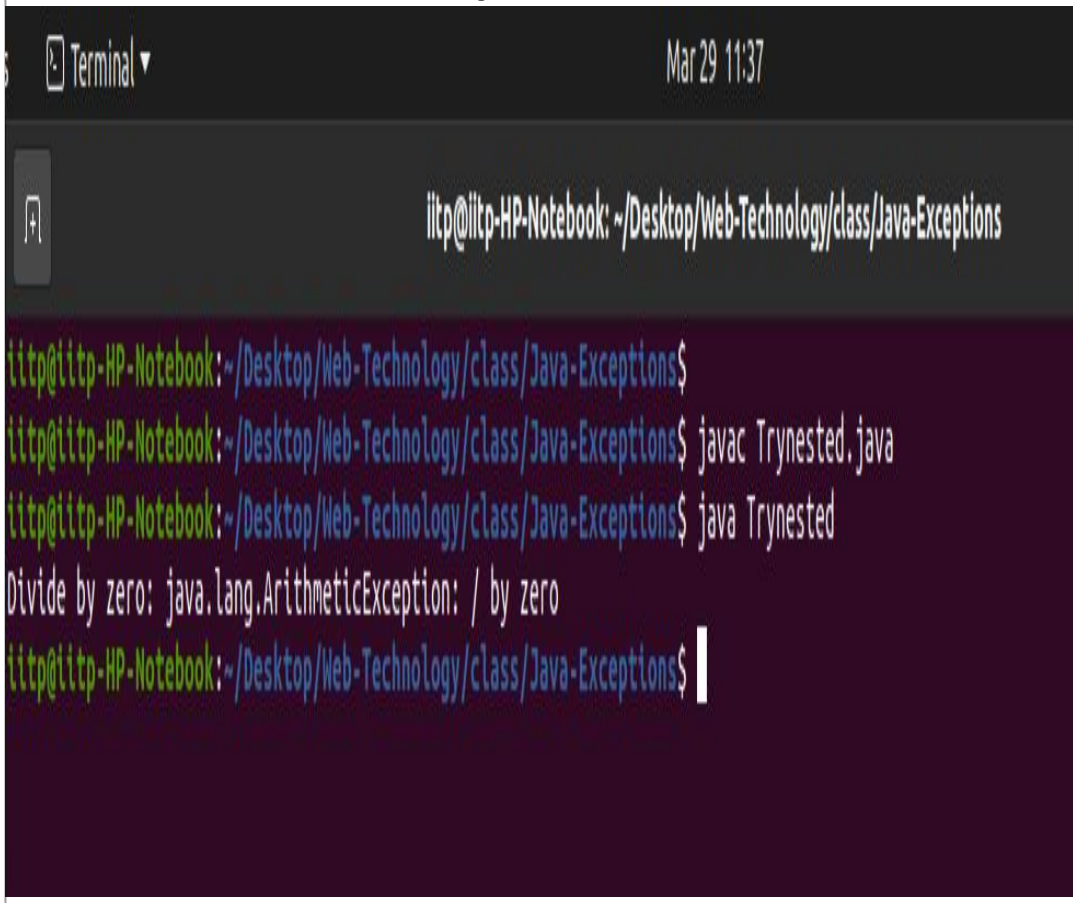
- The try statements can be nested:
  - If an inner try does not catch a particular exception the exception is inspected by the outer try block
  - This continues until:
    - one of the catch statements succeeds or
    - all the nested try statements are exhausted
  - In the latter case, the Java run-time system will handle the exception

```
class Trynested
{
	public static void main(String args[])
	{
		try
		{
			int a = args.length;
			int b = 82/a;
			System.out.println("a= "+a);
			try
			{
				if(a==1)
				{
					a = a/ (a-a);
				}
				if(a==2)
				{
					int c[] = {1};
					c[82]=-999;
				}
			}
			catch(ArrayIndexOutOfBoundsException e)
			{
				System.out.println(e);
			}
		}
		catch(ArithmeticException e)
		{
			System.out.println("Divide by zero: "+e);
		}
	}}
```

Nested try statements

```
class Trynested
{
	public static void main(String args[])
	{
		try
		{
			int a = args.length;
			int b = 82/a;
			System.out.println("a= "+a);
			try
			{
				if(a==1)
				{
					a = a/ (a-a);
				}
				if(a==2)
				{
					int c[] = {1};
					c[82]=-999;
				}
			}
			catch(ArrayIndexOutOfBoundsException e)
			{
				System.out.println(e);
			}
		}
		catch(ArithmeticException e)
		{
			System.out.println("Divide by zero: "+e);
		}
	}}
```

Program Running Output



Throwing Exceptions

- The throw statement can be used either to throw our own exceptions or to throw an instance of Java's exception *throw ThrowableInstance;*
- *ThrowableInstance* must be an object of type *Throwable* or its subclass
- Once an exception is thrown by:

  throw ThrowableInstance;

  - the flow of control stops immediately

Throwing Exceptions (Contd.)

- Once an exception is thrown by:

  throw ThrowableInstance;

  - the flow of control stops immediately
  - the nearest enclosing try statement is inspected if it has a catch statement that matches the type of exception:
    - if one exists, control is transferred to that statement
    - otherwise, the next enclosing try statement is examined
  - if no enclosing try statement has a corresponding catch clause, the default exception handler halts the program and prints the stack

Creating Exceptions

- Two ways to obtain a Throwable instance:
  - Creating one with the new operator
    - All Java built-in exceptions have at least two constructors: one without parameters and another with one String parameter:

```
throw new NullPointerException("demo");
```

    - using a parameter of the catch clause:

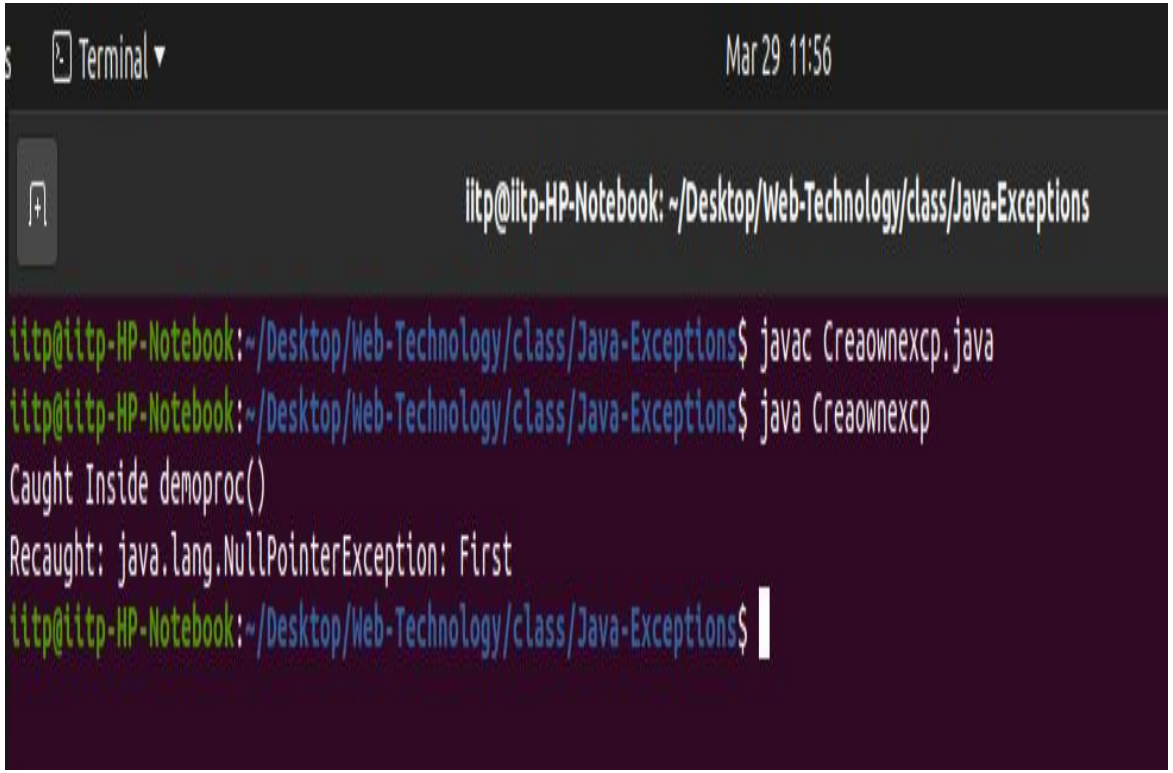
```
try {...}

catch(Throwable e) {... e ...}
```

Creating Exceptions (Contd.)

```
class Creaownexcp
{
	static void demoproc()
	{
		try
		{
			throw new NullPointerException("First");
		}
		catch(NullPointerException e)
		{
			System.out.println("Caught Inside demoproc()");
			throw e;
		}
	}
	public static void main(String args[])
	{
		try
		{
			demoproc();
		}
		catch(NullPointerException e)
		{
			System.out.println("Recaught: "+e);
		}
	}
}
```

Creating Exceptions (Contd.)

```
class Creaownexcp
{
	static void demoproc()
	{
		try
		{
			throw new NullPointerException("First");
		}
		catch(NullPointerException e)
		{
			System.out.println("Caught Inside demoproc()");
			throw e;
		}
	}
	public static void main(String args[])
	{
		try
		{
			demoproc();
		}
		catch(NullPointerException e)
		{
			System.out.println("Recaught: "+e);
		}
	}
}
```



throws statement

- If a method is capable of causing an exception that it does not handle, it must specify this behavior by the *throws* clause in its declaration:

Syntax:

```
return_type method_name(parameter-list) throws exception-list
{
                    //statements
}
```

where exception-list is a comma-separated list of all types of exceptions that a method might throw

throws statement (Contd.)

```
class Demothrows
{
    static void throwOne()
    {
        System.out.println("Inside the method throwOne()");
        throw new IllegalAccessException("second");
    }
    public static void main(String args[])
    {
        throwOne();
    }
}
```

throws statement (Contd.)

```
class Demothrows
{
    static void throwOne()
    {
        System.out.println("Inside the method throwOne()");
        throw new IllegalAccessException("second");
    }
    public static void main(String args[])
    {
        throwOne();
    }
}
```

throws statement (Contd.)

```
class Demothrows
{
    static void throwOne()
    {
        System.out.println("Inside the method throwOne()");
        throw new IllegalAccessException("second");
    }
    public static void main(String args[])
    {
        throwOne();
    }
}
```

The throwOne method throws an exception that it does not catch, nor declares it within the throws clause. Therefore the program does not compile

throws statement (Contd.)

```
class Demothrows
{
    static void throwOne() throws
IllegalAccessException
    {
        System.out.println("Inside the
method throwOne()");
        throw new
IllegalAccessException("second");
    }
```

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

throws statement (Contd.)

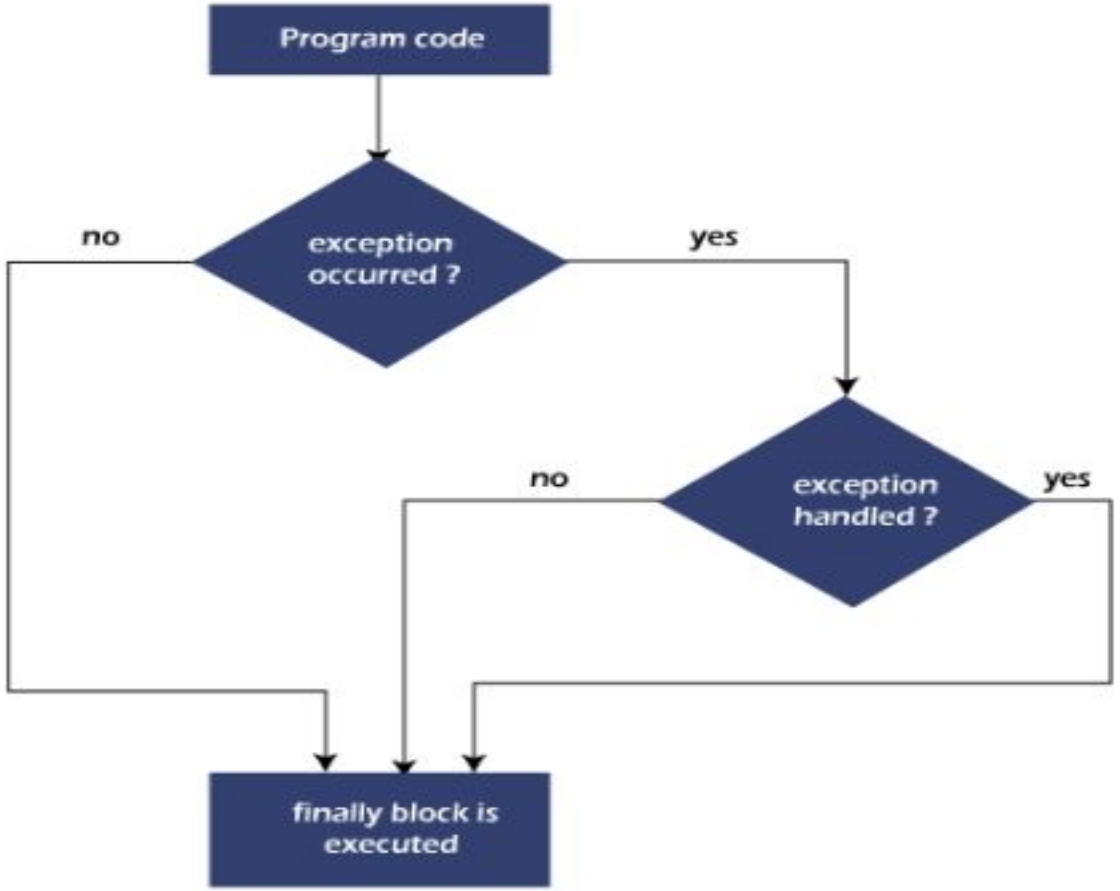
On running the program in the previous slide, we are able to successfully obtain the following output:

```
Terminal ▾                                                    Mar 29 12:13
iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Exceptions
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Exceptions$ javac Demothrows.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Exceptions$ java Demothrows
Inside the method throwOne()
Caught: java.lang.IllegalAccessException: second
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Exceptions$
```

finally block

- When an exception is thrown:
  - the execution of a method is changed
  - the method may even return prematurely
- This may be a problem is many situations
  - For instance, if a method opens a file on entry and closes on exit; exception handling should not bypass the proper closure of the file
- The finally block will execute whether or not an exception is thrown

finally block flowchart

- 

finally block

```
try { ... }
catch(Exception1 ex1) { ... } ...
finally { ... }
```

Executed after try/catch whether or not the exception is thrown

finally block

- Any time a method is to return to a caller from inside the try/catch block via:
  - uncaught exception or
  - explicit return

  The finally clause is executed just before the method returns
- The try statement requires at least one catch or finally clause, although both are optional

finally block

```
class Demofinally
{
	static void procA()
	{
		try
		{
			System.out.println("Inside the procA() method");
			throw new RuntimeException("Third");
		}
		finally
		{
			System.out.println("ProcA's method's finally");
		}
	}
	public static void main(String args[])
	{
		try
		{
			procA();
		}
		catch(Exception e)
		{
			System.out.println("Exception caught");
		}
	}

}
```

finally block

```
class Demofinally
{
	static void procA()
	{
		try
		{
			System.out.println("Inside the procA() method");
			throw new RuntimeException("Third");
		}
		finally
		{
			System.out.println("ProcA's method's finally");
		}
	}
	public static void main(String args[])
	{
		try
		{
			procA();
		}
		catch(Exception e)
		{
			System.out.println("Exception caught");
		}
	}

}
```

finally block

```
class Demofinally
{
        static void procA()
        {
                try
                {
                        System.out.println("Inside the procA() method");
                        throw new RuntimeException("Third");
                }
                finally
                {
                        System.out.println("ProcA's method's finally");
                }
        }
        public static void main(String args[])
        {
                try
                {
                        procA();
                }
                catch(Exception e)
                {
                        System.out.println("Exception caught");
                }
        }

}
```

Here, procA() prematurely breaks out of the try by throwing an exception, followed by executing the statements in the finally block.

finally block

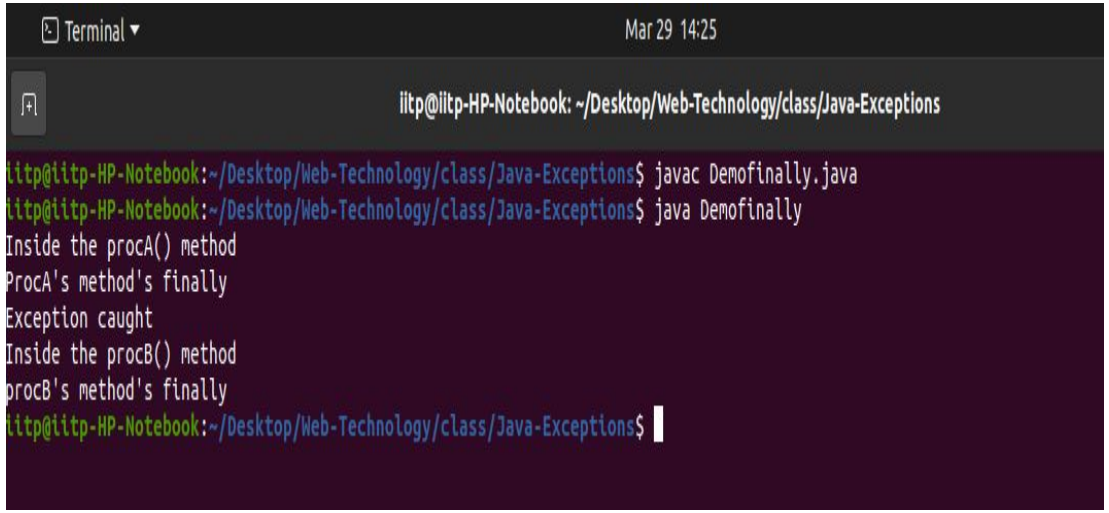
```
class Demofinally
{
	static void procA()
	{
		try
		{
			System.out.println("Inside the procA()
method");
			throw new RuntimeException("Third");
		}
		finally
		{
			System.out.println("ProcA's method's
finally");
		}
	}
	public static void main(String args[])
	{
		try
		{
			procA();
		}
		catch(Exception e)
		{
			System.out.println("Exception caught");
		}
		procB();
	}
```

```
static void procB()
	{
		try
		{
			System.out.println("Inside the procB()
method");
			return;
		}
		finally
		{
			System.out.println("procB's method's
finally");
		}

	}
```

finally block

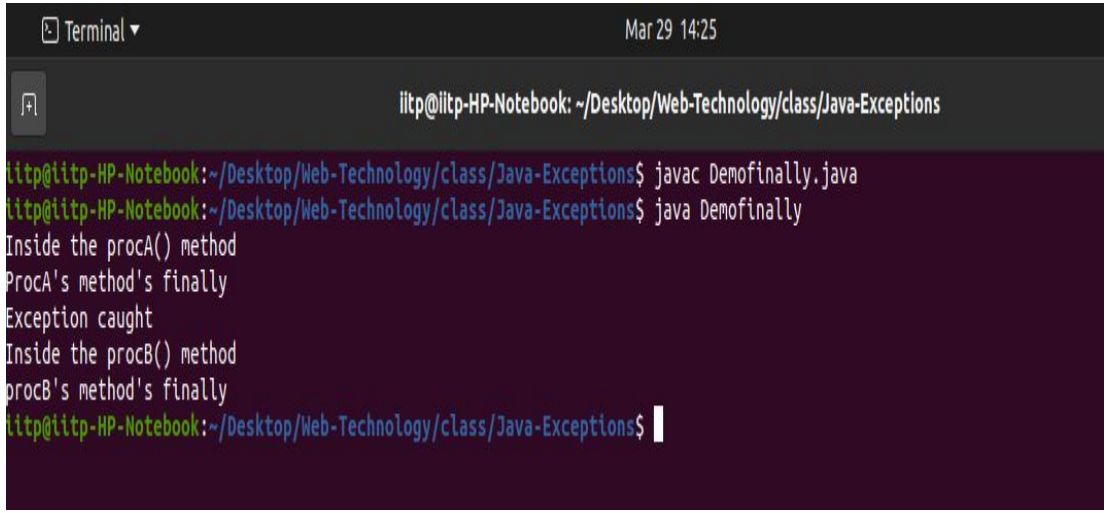
```java
class Demofinally
{
        static void procA()
        {
                try
                {
                        System.out.println("Inside the procA() method");
                        throw new RuntimeException("Third");
                }
                finally
                {
                        System.out.println("ProcA's method's finally");
                }
        }
        public static void main(String args[])
        {
                try
                {
                        procA();
                }
                catch(Exception e)
                {
                        System.out.println("Exception caught");
                }
                procB();
        }

}
```

```java
static void procB()
        {
                try
                {
                        System.out.println("Inside the procB() method");
                        return;
                }
                finally
                {
                        System.out.println("procB's method's finally");
                }

        }
```



```java
class Demofinally
{
	static void procA()
	{
		try
		{
			System.out.println("Inside the procA() method");
			throw new RuntimeException("Third");
		}
		finally
		{
			System.out.println("ProcA's method's finally");
		}
	}
	public static void main(String args[])
	{
		try
		{
			procA();
		}
		catch(Exception e)
		{
			System.out.println("Exception caught");
		}
		procB();
	}

}
```

```java
static void procB()
	{
		try
		{
			System.out.println("Inside the procB() method");
			return;
		}
		finally
		{
			System.out.println("procB's method's finally");
		}

	}
```



procB's try statement is exited via a return statement, the finally clause is executed before the procB returns

finally block

```
class Demofinally
{
        static void procA()
        {
                try
                {
                        System.out.println("Inside the procA() method");
                        throw new RuntimeException("Third");
                }
                finally
                {
                        System.out.println("ProcA's method's finally");
                }
        }
        public static void main(String args[])
        {
                try
                {
                        procA();
                }
                catch(Exception e)
                {
                        System.out.println("Exception caught");
                }
                procB();
                procC();
        }

}
```

```
static void procB()
        {
                try
                {
                        System.out.println("Inside the procB() method");
                        return;
                }
                finally
                {
                        System.out.println("procB's method's finally");
                }

        }

static void procC()
        {
                try
                {
                        System.out.println("Inside the procC() method");
                }
                finally
                {
                        System.out.println("procC's method's finally");
                }

        }
```

```
Terminal                                                    Mar 29  14:31
iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Exceptions
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Exceptions$ javac Demofinally.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Exceptions$ java Demofinally
Inside the procA() method
ProcA's method's finally
Exception caught
Inside the procB() method
procB's method's finally
Inside the procC() method
procC's method's finally
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Exceptions$
```

In procC the try statement executes normally without error, however the finally clause is still executed.

User Defined Exceptions

- Build-in exception classes handle some generic errors. For application-specific errors define your own exception classes
- Define a subclass of Exception:

```
class MyException extends Exception { ... }
```

- Exception itself is a sub-class of Throwable

User Defined Exceptions

- All user exceptions have the methods defined by the Throwable class:
  - *Throwable fillInStackTrace():* returns a Throwable object that contains a completed stack trace; the object can be rethrown
  - *Throwable getCause():* returns the exception that underlies the current exception. If no underlying exception exists, null is returned
  - *String getLocalizedMessage():* returns a localized description of the exception

User Defined Exceptions

- All user exceptions have the methods defined by the Throwable class:
  - *String getMessage():* returns a description of the exception
  - *StackTraceElement[] getStackTrace():* returns an array that contains the stack trace; the method at the top is the last one called before exception
  - *Throwable initCause(Throwable causeExc):* associates causeExc with the invoking exception as its cause, returns the exception reference
  -

User Defined Exceptions

- All user exceptions have the methods defined by the Throwable class:
  - *void printStackTrace():* displays the stack trace
  - *void printStackTrace(PrintStream stream):* sends the stack trace to the specified stream
  - *void setStackTrace(StackTraceElement elements[]):* sets the stack trace to the elements passed in elements; for specialized applications only
  - *String toString():* returns a String object containing a description of the exception; called by print() when displaying a Throwable object

User Defined Exceptions

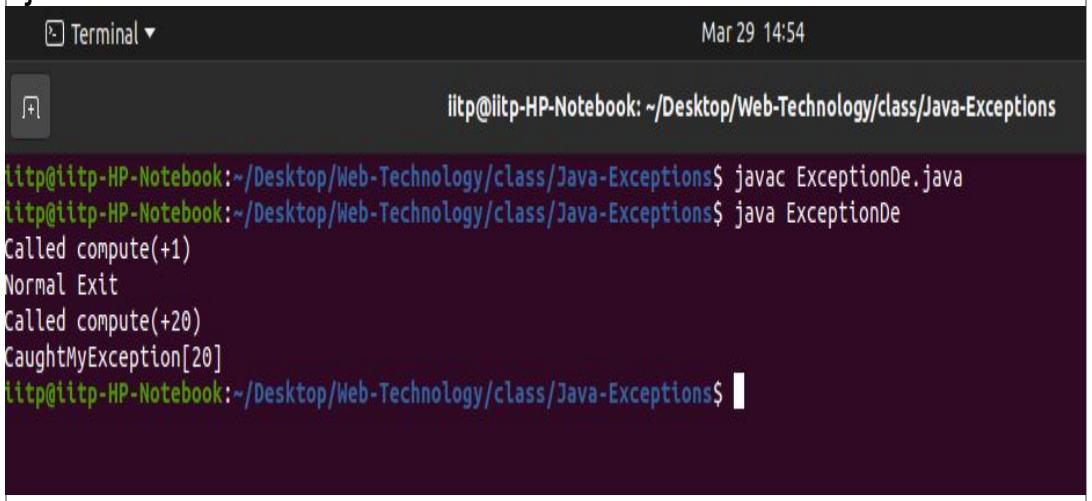
```
class MyException extends Exception
{
	private int detail;
	MyException(int a)
	{
		detail = a;
	}
	public String toString()
	{
		return "MyException["+detail+"]";
	}
}
class ExceptionDe
{
	static void compute(int a) throws MyException
	{
		System.out.println("Called compute(+"+a+")");
		if(a>10)
		{
			throw new MyException(a);
		}
		System.out.println("Normal Exit");
	}
```

```
public static void main(String args[])
	{
		try
		{
			compute(1);
			compute(20);
		}
		catch(MyException e)
		{
			System.out.println("Caught"+e);
		}
	}
}
```

User Defined Exceptions

```
class MyException extends Exception
{
	private int detail;
	MyException(int a)
	{
		detail = a;
	}
	public String toString()
	{
		return "MyException["+detail+"]";
	}
}
class ExceptionDe
{
	static void compute(int a) throws MyException
	{
		System.out.println("Called compute(+"+a+")");
		if(a>10)
		{
			throw new MyException(a);
		}
		System.out.println("Normal Exit");
	}
```

```
public static void main(String args[])
	{
		try
		{
			compute(1);
			compute(20);
		}
		catch(MyException e)
		{
			System.out.println("Caught"+e);
		}
	}
}
```



References

1. https://www.javatpoint.com/nested-interface
2. http://etutorials.org/cert/java+certification/Chapter+6.+Object-oriented+Programming/6.7+Polymorphism+and+Dynamic+Method+Lookup/#:~:text=Dynamic%20method%20lookup%20is%20the,instance%20method%20is%20not%20polymorphic.
3. http://coding-guru.com/polymorphism-java/
4. https://coderanch.com/t/378538/java/Dynamic-method-lookup
5. https://www.baeldung.com/java-inner-interfaces
6. https://cs-fundamentals.com/java-programming/java-static-nested-or-inner-interfaces