

CSEg2202

Object Oriented Programming

CHAPTER - 5

Exception Handling

2023

Content of the lecture



Exception Handling

- [redacted]
- [redacted]
  - [redacted]
  - [redacted]

Types of Exception Handling

- [redacted]
  - [redacted]
  - [redacted]

[redacted]

- [redacted]

[redacted]

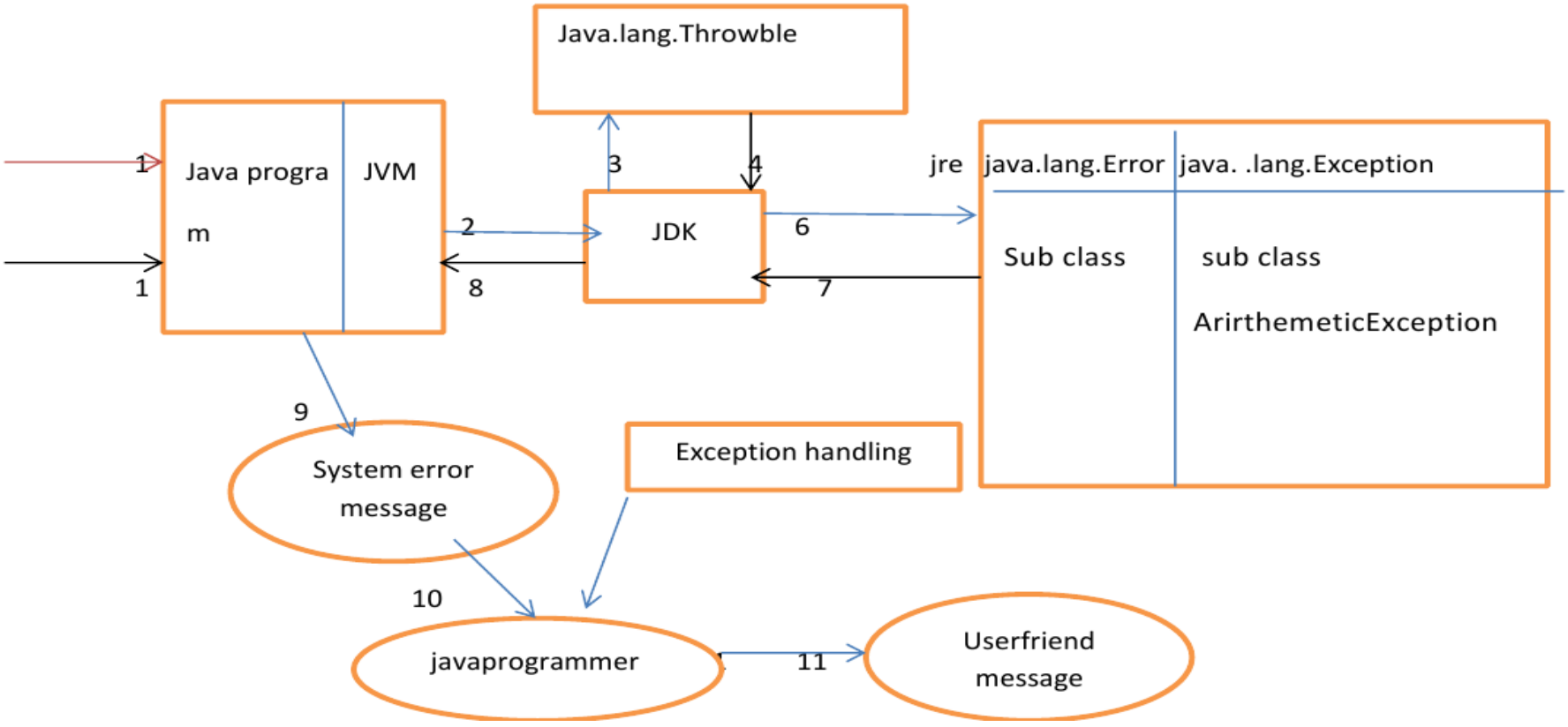
Types of Exception Handling

- [redacted]
  - [redacted]
  - [redacted]
- [redacted]
  - [redacted]
  - [redacted]

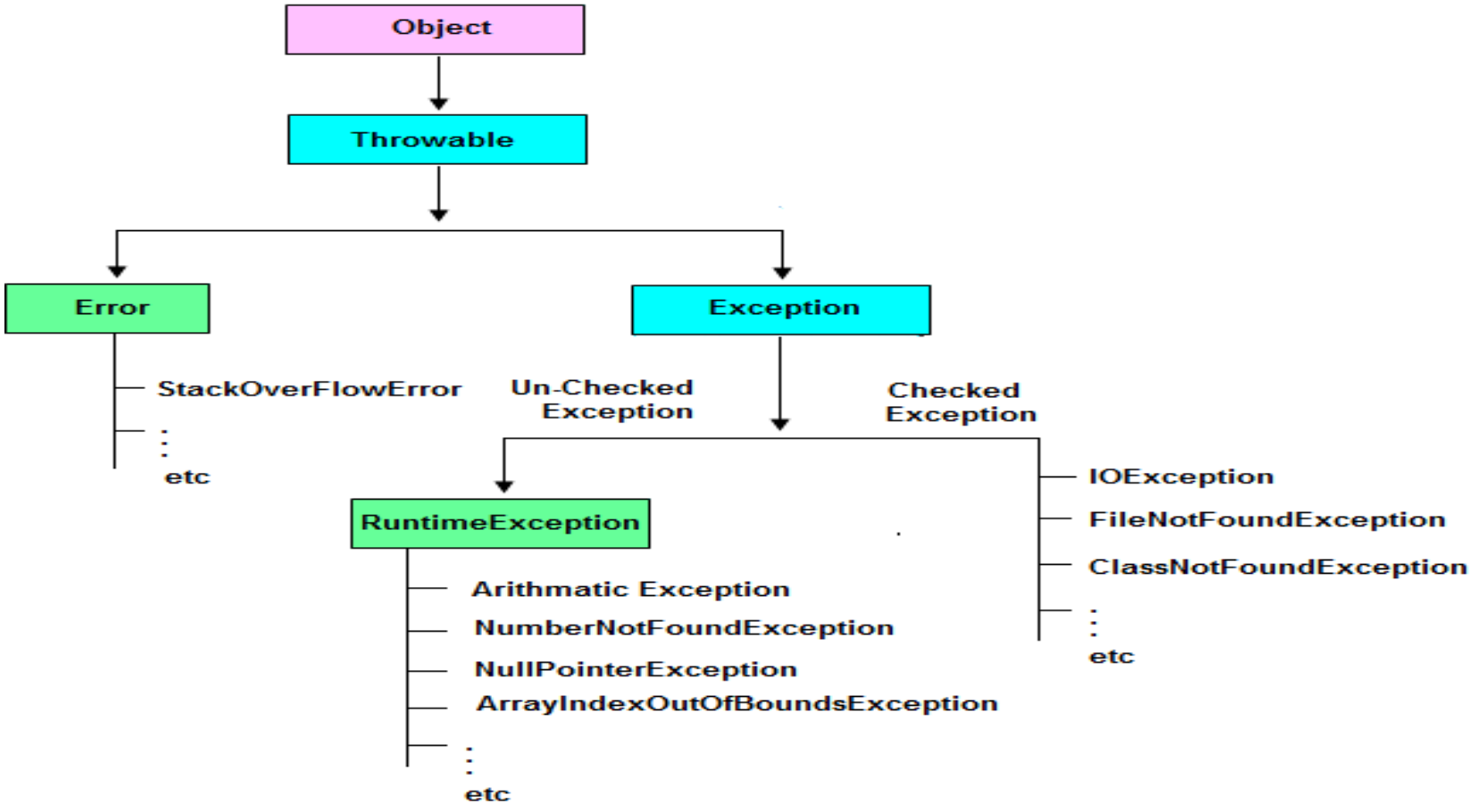
Types of Exception Handling

- [redacted]
- [redacted]

How exception occur in java run time environment



Hierarchy of Exception classes



Handling the Exception

- **Use Five keywords for Handling the Exception**
  - ✓ try
  - ✓ catch
  - ✓ finally
  - ✓ throws
  - ✓ throw

Syntax

```
try
{
  // statements causes problem at run time
}
catch(type of exception-1 object-1)
{
  // statements provides user friendly error message
}
catch(type of exception-2 object-2)
{
  // statements provides user friendly error message
}
finally
{
  // statements which will execute compulsory
}
```

Exception Message Details

Exception message format:

```
[exception class]: [additional description of
exception]
    at [class].[method]([file]:[line number])
```

Example:

```
java.lang.ArrayIndexOutOfBoundsException: 2
       at
ExceptionExample.main(ExceptionExample.java:4)
```

- ✓ What exception class? `ArrayIndexOutOfBoundsException`
- ✓ Which array index is out of bounds? `2`
- ✓ What method throws the exception? `main`
- ✓ What file contains the method? `ExceptionExample.java`
- ✓ What line of the file throws the exception? `4`

Example without Exception Handling

```
1  class ExceptionDemo
2  {
3      public static void main(String[] args)
4      {
5          int a=10, ans=0;
6          ans=a/0;
7          System.out.println("Denominator not be zero");
8      }
9  }
```

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Lab5.ExceptionDemo.main(Exception.java:17)
```

Example with Exception Handling

```
1  class ExceptionDemo {
2      public static void main(String[] args)
3      {
4          try{
5              int a=10, ans=0;
6              ans=a/0;
7              System.out.println("Denominator not be zero");
8          } catch(ArithmeticException e) {
9              System.out.println("Denominator can't be Zero");
10         }
11     }
12 }
```

Output:
Denominator can't be Zero

try and catch block

- Inside **try block** we write the block of statements which causes executions at run time in other words try block always contains problematic statements.
- **Important points about try block**
  - ✓ If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.
  - ✓ After executing appropriate catch block, even through we use run time statement, CPU control never goes to try block to execute the rest of the statements.
  - ✓ Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

Syntax

```
try
{
  .....
}
/* Here no other statements are allowed
between try and catch block */
catch()
{……..}
```

- Each and every try block must contains at least one catch block. But it is highly recommended to write multiple catch blocks for generating multiple user friendly error messages.
- One try block can contains another try block that is nested or inner try block can be possible.

catch block

❖Inside **catch** block we write the block of statements which will generates user friendly error messages.

❖**catch block important points**

- ✓ Catch block will execute when exception occurs in try block.
- ✓ You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.
- ✓ At a time only one catch block will execute out of multiple catch blocks.
- ✓ in catch block you declare an object of sub class and it will be internally referenced by JVM.

finally Block in Exception Handling

- Inside **finally** block we write the block of statements which will relinquish (released or close or terminate) the resource (file or database) where data store permanently.
- **finally block important points**
  - ✓ Finally block will execute compulsory
  - ✓ Writing finally block is optional.
  - ✓ You can write finally block for the entire java program
  - ✓ In some of the circumstances one can also write try and catch block in finally block.

Example

```
1  class ExceptionDemo {
2      public static void main(String[] args)
3      {
4          try {
5              int a=10, ans=0;
6              ans=a/0;
7              System.out.println("Denominator not be zero");
8          } catch(ArithmeticException e) {
9              System.out.println("Denominator can't be Zero");
10         } finally {
11             System.out.println("Finally happened");
12         }
13     }
14 }
```

Output

```
Denominator can't be Zero
Finally happened
```

Catching Multiple Exceptions

- [redacted]

Example

```
1  class TestMultipleCatchBlock1{
2    public static void main(String args[]){
3     try{
4      int a[]=new int[5];
5      a[5]=30/0;
6     }
7     catch(Exception e){
8         System.out.println("common task completed");
9     }
10    catch(ArithmeticException e){
11        System.out.println("task1 is completed");
12    }
13    catch(ArrayIndexOutOfBoundsException e){
14        System.out.println("task 2 completed");
15    }
16    System.out.println("rest of the code...");
17   }
18  }
```

Output

Compile Time Error

Example : Multi catch block

```
1  public class TestMultipleCatchBlock{
2    public static void main(String args[]) {
3     try{
4      int a[]=new int[5];
5      a[5]=30/0;
6     }
7     catch(ArithmeticException e) {
8          System.out.println("task1 is completed");
9     }
10    catch(ArrayIndexOutOfBoundsException e) {
11         System.out.println("task 2 completed");
12    }
13    catch(Exception e){
14         System.out.println("common task completed");
15    }
16    System.out.println("rest of the code...");
17   }
18 }
```

- If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

Output

task1 completed
rest of the code...

Java Nested try block

- **Java Nested try block: T**he try block within a try block is known as nested try block in java.
- **Why use nested try block:** Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax:

```
....
try
{  statement 1;
   statement 2;
    try
    {  statement 1;
       statement 2;
    }  catch(Exception e)  {
    } }
catch(Exception e)
{  }
....
```

Java nested try example

```
1  class TestNestedTryExample{
2      public static void main(String args[]){
3          try{
4              try{
5                  System.out.println("going to divide");
6                  int b=39/0;
7              } catch (ArithmeticException e){
8                  System.out.println("Division by zero");
9              }
10             try{
11                 int a[]=new int[5];
12                 a[5]=4;
13             }catch(ArrayIndexOutOfBoundsException e){
14                 System.out.println("Array Out of index");
15             }
16             System.out.println("other statement");
17         } catch(Exception e){
18             System.out.println("handeled");
19         }
20         System.out.println("normal flow..");
21     }
22 }
```

Throws

- The **Java throws keyword** is used to declare an exception. It gives an 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 the normal flow of the program can be maintained.
- **Syntax**

```
return_type method_name() throws exception_class_name{
  ...
}
```

Example

```
1  import java.io.IOException;
2  class Testthrows1{
3    void m()throws IOException{
4      throw new IOException("device error");//checked exception
5    }
6    void n()throws IOException{
7      m();
8    }
9    void p(){
10    try{
11     n();
12    }catch(Exception e){System.out.println("exception handled");}
13   }
14   public static void main(String args[]){
15    Testthrows1 obj=new Testthrows1();
16    obj.p();
17    System.out.println("normal flow...");
18   }
19 }
```

Throw

- Throw is a keyword in java language which is used to throw any user defined exception to the same signature of method in which the exception is raised.
- Throw keyword always should exist within method body.
- whenever method body contain throw keyword than the caller method should be followed by throws keyword.
- **Syntax:**

```
throw new exception_class("error message");
```

Example

```
public class TestThrow {
    public static void validate(int age) {
        if(age<18) {
            throw new ArithmeticException("Person is not eligible to
                vote");
        }
        else {
            System.out.println("Person is eligible to vote!!");
        }
    }
    public static void main(String args[]){
        validate(16);
        System.out.println("rest of the code...");
  }
}
```

Custom Exception in Java

- It is exception designed by programmer.
- **Rules to design user defined Exception**
  - ✓ Create a package with valid user defined name.
  - ✓ Create any user defined class.
  - ✓ Make that user defined class as derived class of Exception or RuntimeException class.
  - ✓ Declare parametrized constructor with string variable.
  - ✓ call super class constructor by passing string variable within the derived class constructor.
  - ✓ Save the program with public class name.java

Example

```
class MyCustomException extends Exception
{

}
// class that uses custom exception MyCustomException
public class TestCustomException2
{
    // main method
    public static void main(String args[])
    {
        try
        {
            // throw an object of user defined exception
            throw new MyCustomException();
        }
        catch (MyCustomException ex)
        {
            System.out.println("Caught the exception");
            System.out.println(ex.getMessage());
        }
        System.out.println("rest of the code...");
    }  }
```

Example

```
1  // class representing custom exception
2  class InvalidAgeException  extends Exception
3  {
4      public InvalidAgeException (String str)
5      {
6          // calling the constructor of parent Exception
7          super(str);
8      }
9  }
10
11 // class that uses custom exception InvalidAgeException
12 public class TestCustomException1
13 {
14      // method to check the age
15     static void validate (int age) throws InvalidAgeException{
16        if(age < 18){
17         // throw an object of user defined exception
18         throw new InvalidAgeException("age is not valid to vote");
19         }
20        else {
21         System.out.println("welcome to vote");
22         }
23      }
```

Example

```
24      // main method
25      public static void main(String args[])
26      {
27          try
28          {
29              // calling the method
30              validate(13);
31          }
32          catch (InvalidAgeException ex)
33          {
34              System.out.println("Caught the exception");
35              // printing the message from InvalidAgeException object
36              System.out.println("Exception occured: " + ex);
37          }
38          System.out.println("rest of the code...");
39      }
40  }
```

Difference between throw and throws in Java

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.



Adama Science and Technology University

School of Electrical Engineering And Computing

2023 G.C