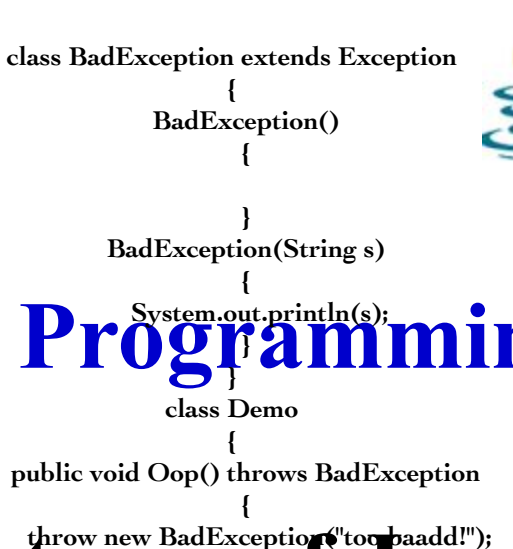

CS205 Object Oriented Programming in Java

Module 3 - More features of Java (Part 8)

PREPARED BY

Renetha J.B.
AP
Dept.of CSE,
Lourdes Matha College of Science and Technology

Topics

- **More features of Java :**
  - **Exception Handling:**
    - ***throw***
    - ***throws***
    - ***finally***

throw statement

- Our program can throw an exception explicitly, using the **throw** statement.
- The general form of **throw is shown here:**

```
throw ThrowableInstance;
```

  - *ThrowableInstance* must be an **object** of type ***Throwable*** or a ***subclass of Throwable.***
  - Primitive types, such as int or char, as well as non-Throwable classes, such as String and Object, cannot be used as exceptions.

throw(contd.)

- Two ways to obtain a **Throwable object:**

1. using a parameter in a **catch** clause, or
2. creating one with the **new** operator

1) Using a parameter in a catch clause

```
catch(ArrayIndexOutOfBoundsException ar)
{
    throw ar;
}
```

2) Creating one with the new operator

```
throw new ArrayIndexOutOfBoundsException();
```

throw statement(contd..)

- The *flow of execution* **stops** immediately after the **throw** statement.
  - Any statements after throw statement will not be executed.
- When exception is thrown using **throw** statement :-
  - the nearest enclosing try block is inspected to see if it has a catch statement that matches the type of exception thrown.
    - If that catch statement has a **matching exception type as the thrown exception**, control is transferred to that statement.
    - If **not matching**, then the *next enclosing try statement is inspected, and so on.*
    - If **no matching catch is found**, then the *default exception handler halts the program and prints the stack trace.*

throw Example 1

```
class ThrowDemo
{
static void show()
{
    try
    {throw new NullPointerException("demoexception");
    }
    catch(NullPointerException e)
    {
     System.out.println("Caught inside show");
     throw e; // rethrow the exception
          }
}
public static void main(String args[])
{
      try {
      show();
          }
      catch(NullPointerException e)
      { System.out.println("Recaught in main: " + e);
      }
} }
```

Here first **throw** in show is caught by matching catch is in show function.

Next throw has no immediate catch So since the exception matches with catch in the **main** function(that calls show), the exception is caught by that matching catch in main.

```
java ThrowDemo
Caught inside show
Recaught in main: java.lang.NullPointerException: demoexception
```

throw with matching catch in calling function

```
class ThrowDemo2
{
   static void show()
   {
     throw new NullPointerException("demoxception");
   }
   public static void main(String args[])
   {
         try
         {
         show();
         } catch(NullPointerException e)
         { System.out.println("Caught in main: " + e);
           }
     }
}
```

Here no matching catch for throw is in show function
So since the exception matches with **catch** in the main function(that calls show), the exception is caught by that matching catch

OUTPUT

Caught in main: java.lang.NullPointerException: demoexception

throw with NO matching catch

```
class ThrowDemo2
{
   static void show()
   {
     throw new NullPointerException("demoxception");
   }
   public static void main(String args[])
   {
         try
         {
         show();
         }
          catch(ArithmeticException e)
         { System.out.println("Caught in main: " + e);
           }
     }
}
```

Here no matching catch is in show function.
So since the exception does not matches with catch in the main function(that calls show)
also, the exception is not caught in the program
the ***default exception handler halts the program and prints the stack trace***

OUTPUT

```
Exception in thread "main" java.lang.ArithmeticException: demoxception
        at ThrowDemo2.show(ThrowDemo2.java:3)
        at ThrowDemo2.main(ThrowDemo2.java:9)
```

throw(contd.)

- Many of Java's built-in run-time exceptions have at least two constructors:
  - one with no parameter and
  - one that takes a string parameter.
- When constructor with string parameter is used, the argument specifies a **string that describes the exception**.
  - This string is displayed when the object is printed using **print( ) or println( ).**
  - It can also be obtained by a call to getMessage( ), which is defined by Throwable.

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

- Here the string **demoxception** inside the constructor of ***NullPointerException*** is the name of the exception.

throws

- A **throws** clause lists the types of exceptions that a **method(function) might throw**.
- **throws** keyword is used with the method signature(header)
- If a method has an exception and it does not handle that exception, it must specify this using **throws** , so that callers of the method can guard themselves against that exception.
- **throws** is necessary for all exceptions, except those of type **Error** or **RuntimeException** or any of their subclasses

throws (contd.)

- All other **exceptions** that a **method can throw** must be declared in the **throws** clause.
  - If they are not, a compile-time error will result.
- General form of a method declaration that includes a **throws clause:**

type method-name(parameter-list) ***throws*** *exception-list*

```
{
   // body of method
}
```

Separated by ,

throw statement but no throws in method-ERROR

```
public class ThrowsEg {
  static void vote(int age) {
    if (age < 18) {
      throw new IllegalAccessException("You must be at least 18 years old.");
                } else {
            System.out.println(“ You can vote!");
                  }
  }
  public static void main(String[] args)
  {
     vote(15);
  }
}
```

```
D:\RENETHAJB\OOP>javac ThrowsEg.java
ThrowsEg.java:4: unreported exception java.lang.IllegalAccessException; must be
caught or declared to be thrown
      throw new IllegalAccessException("You must be at least 18 years old.");
      ^
1 error
```

COMPILE ERROR

This program tries to throw an exception that it does not catch.

Because the program does not specify a **throws clause to declare this exception to be thrown,** the program will not compile.

Include **throws** in method and **try catch** in calling function.

Using throws

```
public class ThrowsEg {
  static void vote(int age) throws IllegalAccessException{
    if (age < 18) {
      throw new IllegalAccessException("You must be at least 18 years old.");
                    } else {
                          System.out.println(“ You can vote!");
                          }
 }
  public static void main(String[] args)
 {
   try{
       vote(15);
         }
         catch(Exception e)
         {
        System.out.println("Exception: "+e);
         }
 }
}
```

OUTPUT
Exception: java.lang.ArithmeticException: You must be at least 18 years.

```
import java.io.*;
class Sample{
 void show() throws IOException{
  throw new IOException("Thrown IO error");
 }
}
public class Testthrows{
   public static void main(String args[]){
    try{
     Sample s=new Sample();
     s.show();
    }
catch(Exception e){System.out.println("Exception handled. "+e);}

    System.out.println("Normal program flow");
  }
}
```

Output

```
Exception handledjava.io.IOException: Thrown IO error
Normal program flow
```

finally

- **finally** creates a block of code that will be executed **after** a try/catch block has completed and **before** the control goes out from the try/catch block.

```
try {
// block of code to monitor for errors
      }
catch (ExceptionType1 exOb)
{
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{   // exception handler for ExceptionType2
}
// ...
finally
{   // block of code to be executed after try block ends
}
```

Why finally is needed?

- When exceptions are thrown, execution in a method takes a nonlinear path and changes the normal flow through the method.
  - Sometime exception causes the method to return prematurely.
  - This may cause problems in some cases.
  - E.g a method opens a file upon entry and closes it upon exit, then we will not want the code that closes the file to be bypassed by the exception-handling mechanism.
    - In such situations the code for closing that file and other codes that should not be bypassed should be written inside **finally** block
    - This will ensure that necessary codes are not skipped because of exception handling.

finally(contd.)

- The **finally** block **will execute** whether or not an exception is thrown.
  - If an **exception is thrown**, the **finally** block will execute *even if no catch statement matches the exception*.
  - Any time a method is about to return to the caller from inside a **try/catch block**, (via an uncaught exception or an explicit return statement). the **finally clause is also executed just before the method returns.**
- *If* a **finally** block is associated with a **try**, the finally block will be executed upon conclusion of the try.

try mandatory

finally(contd.)

- The finally clause is optional. However, each try statement requires **at least one catch or a finally clause**

```
try
{
//monitor exception
}
finally
{
}
```

```
try
{
//monitor exception
}
catch(ExceptionType1 ob)
{
}
```

```
try
{
//monitor exception
}
catch(ExceptionType1 ob)
{
}
catch(ExceptionType2 ob)
{
}
//
finally
{
}
```

finally example

```
class FinallyTry
{
public static void main(String[] args)
    {
        try
        {
        int a=5/0;
        }
        catch(ArithmeticException ae)
        {
        System.out.println("Exception is "+ae);
        }
        finally
        {
        System.out.println("Inside finally");
        }
    System.out.println("After try - catch -finally");
    }
}
```

OUTPUT

Exception is java.lang.ArithmeticException: / by zero
Inside finally
After try - catch -finally

Here int a=5/0; inside try causes ArithmeticException
And it is caught by **catch(ArithmeticException ae)**
And prints the message
Exception is ***details about exception***
Then it enters finally block and prints ***Inside finally***
Then it comes out from try catch finally block
and prints the message
After try - catch -finally

finally example

```
class FinallyTry
{
public static void main(String[] args)
    {
        try
        {
        int a=5/2;
        }
        catch(ArithmeticException ae)
        {
        System.out.println("Exception is "+ae);
        }
        finally
        {
        System.out.println("Inside finally");
        }
    System.out.println("After try - catch -finally");
    }
}
```

OUTPUT
Inside finally
After try - catch -finally

Here int a=5/2; inside try does not cause exception
(So it is not caught by catch(ArithmeticException ae))
Then it enters finally block and prints ***Inside finally***
Then it comes out from try catch finally block
and prints the message
After try - catch -finally

finally Example

```
class Sample{
  void show(int n)
  {  int c=10;
     try
     {
     System.out.println("inside try");
     c=10/n;
     }
     catch(Exception e)
     {
     System.out.println("Exception  caught"+e);
     }

     finally
     {
     System.out.println("Finally done");
     }
  }


}
```

```
class Finally1
{
public static void main(String[] args)
      {
      Sample ob=new Sample();
      ob.show(1);
      ob.show(0);

      System.out.println("Finished");
      }
 }
```

```
inside try
Finally done
inside try
Exception caughtjava.lang.ArithmeticException: / by zero
Finally done
Finished
```

```
class FinallyDemo {
static void procA() {
try {
System.out.println("inside procA");
throw new RuntimeException("demo");
} finally {
System.out.println("procA's finally");
      }
}
static void procB() {
try {
System.out.println("inside procB");
return;
} finally {
System.out.println("procB's finally");
      }
}
// Execute a try block normally.
static void procC() {
try {
System.out.println("inside procC");
} finally {

System.out.println("procC's finally");
      }
}
public static void main(String args[]) {
try {
procA();
} catch (Exception e) {
System.out.println("Exception caught");
}
procB();
procC();
}
}
```

```
inside procA
procA's finally
Exception caught
inside procB
procB's finally
inside procC
procC's finally
```


Java

Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.