Object Oriented Programming using Java

Prepared By:
Suyel, PhD
Assistant Professor
Dept. of CSE, NIT Patna

Outline

1. Exception
2. Constructors of Throwable Class
3. Methods of Throwable Class
4. Types of Exception
5. Exception Handling
6. Exception and Inheritance
7. Redirecting and Rethrowing Exception

Exception

- An exception is an unwanted or unexpected event, which occurs during the execution of a program.
- It disrupts the normal flow of the program's execution.
- Exceptions are thrown by a program, and may be caught and handled by another part of the program.
- A program can be separated into a normal execution flow and an exception execution flow.

Exception (Cont…)

- **Hierarchy of Exception**

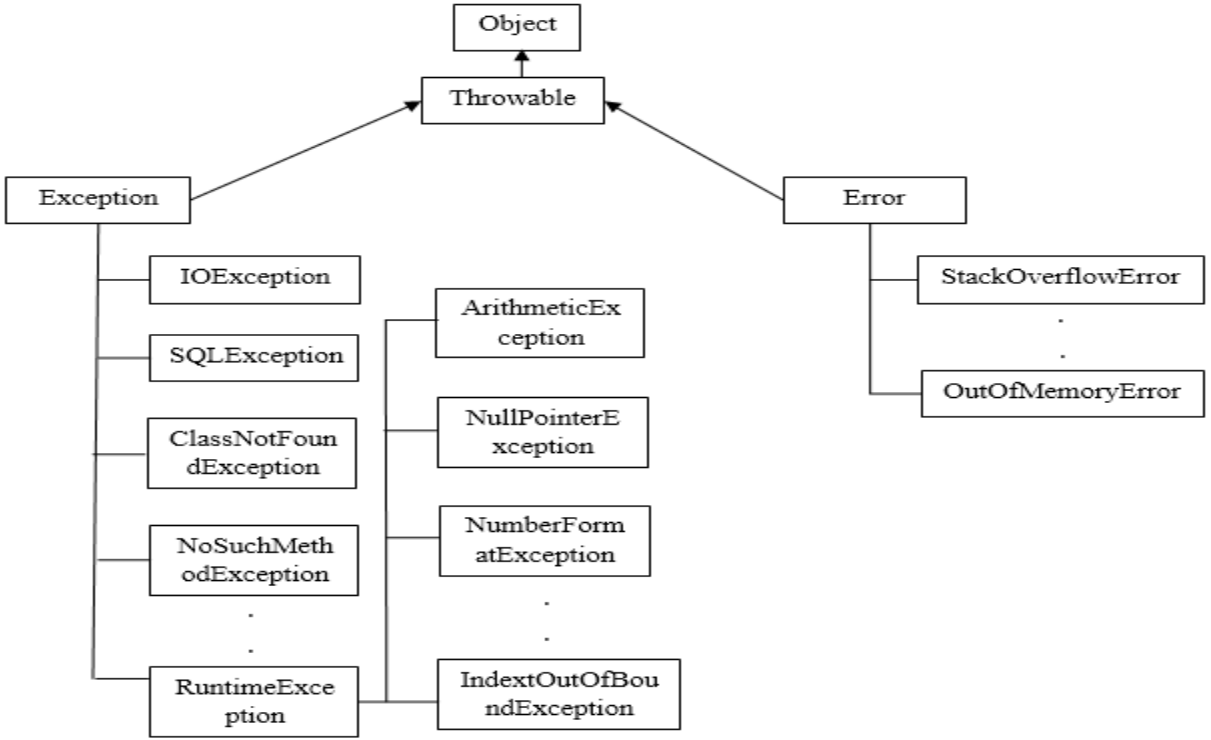


Fig. 1: Hierarchy of exception

Exception (Cont…)

- **Difference between Exception and Error**

Exception	Error
Exceptions are defined in package, namely java.lang.Exception.	Errors are defined in the package, namely java.lang.Error.
Exceptions can be both Checked and Unchecked exceptions.	Errors can be only Unchecked type.
Programs are recoverable from Exceptions by handling them appropriately.	Programs are irrecoverable from Errors once they occur.
Exceptions are mainly caused by the application itself.	Errors are mostly caused by the environment in which application is running.
Example: NullPointerException	Example: OutOfMemoryError

Constructors of Throwable Class

Constructor	Description
Throwable()	This constructs a new throwable with null as its detail message.
Throwable(String message)	It constructs a new throwable with the specified detail message.
Throwable(String message, Throwable cause)	This constructs a new throwable with the specified detail message and cause.
Throwable(Throwable cause)	It constructs a new throwable with the specified cause and a detail message of (cause == null ? null : cause.toString()) (which typically contains the class and detail message of cause).

Methods of Throwable Class

Method	Description
String getMessage()	It returns the detail message of this Throwable, or null, if this Throwable does not have a detail message.
String getLocalizedMessage()	This creates a localized description of this Throwable. Subclasses may override this method in order to produce a local-specific message.
String toString()	It returns a short description of this Throwable.
void printStackTrace()	This method prints this Throwable and its backtrace to the standard error stream.
void printStackTrace(PrintStream s)	It prints this Throwable and its backtrace to the specified print stream.

Methods of Throwable Class (Cont…)

Method	Description
void printStackTrace(PrintWriter s)	This method prints this Throwable and its backtrace to the specified print writer.
void setStackTrace(StackTraceElement [] stackTrace)	It sets the stack trace elements that is returned by getStackTrace() and printed by printStackTrace() and related methods.
Throwable fillInStackTrace()	This fills in the execution stack trace. This method is useful, when an application is re-throwing an error or exception.
Throwable initCause (Throwable cause)	It initializes the cause of this throwable to the specified value.

Methods of Throwable Class (Cont…)

Mthod	Description
Throwable getCause ()	This method returns the cause of this throwable, or null, if the cause is nonexistent or unknown.
getStackTrace ()	It provides programmatic access to the stack trace information printed by printStackTrace().

Types of Exception

- **Checked Exception:**
  - All the exceptions other than **Runtime Exceptions** are known as **Checked Exceptions**.
  - Compiler checks them during compilation.
  - If these exceptions are not handled, we get compilation error.
  - These exceptions directly inherit Throwable class.
- **Unchecked Exception:**
  - **Runtime Exceptions** are also known as **Unchecked Exceptions**.
  - These exceptions are not checked at compile time.
  - It is the responsibility of the programmer to handle these exceptions.

Types of Exception (Cont…)

- **Example of Checked Exception**

```
import java.io.File;
import java.io.FileReader;

public class Checked
{
   public static void main(String args[])
   {
      File fl = new File("E://file.txt");
      FileReader fr = new FileReader(fl);
   }
}
```

Types of Exception (Cont…)

- **Example of Checked Exception (Cont…)**
  - **Output**

```
Checked.java:9: error: unreported exception FileNotFoundException; must be caugh
t or declared to be thrown
      FileReader fr = new FileReader(f1);
                      ^
1 error
```

Types of Exception (Cont…)

- **Example of Unchecked Exception**

```
public class Unchecked
{
   public static void main(String args[])
   {
      int num[] = {1, 2, 3, 4, 5};
      System.out.println(num[10]);
   }
}
```

Types of Exception (Cont…)

- **Example of Unchecked Exception (Cont…)**
  - **Output**

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 ou
t of bounds for length 5
        at Unchecked.main(Unchecked.java:6)
```

Exception Handling

- **Exception Handling** in Java is one of the powerful mechanisms to handle the runtime errors, so that normal flow of the application can be maintained.
- **Normal Scenario:**

```
Statement 1;
Statement 2;
Statement 3;    //Exception occur
Statement 4;
Statement 5;
```

Exception Handling (Cont…)

Keyword	Description
Try	The try block contains the code that might throw/raise an exception. It contains at least one catch block or finally block.
Catch	A catch block must be declared after try block. It contains the error handling code.
Finally	The code present in finally block will always be executed. It must be followed by try or catch block and used to execute some important code.
Throw	Programmer can also throw/raise exception explicitly at runtime based on some condition. The throw keyword is used to explicitly throw our own exception.
Throws	throws keyword is used to declare which exceptions can be thrown from a method. Throws is always added after the method signature.

Exception Handling (Cont…)

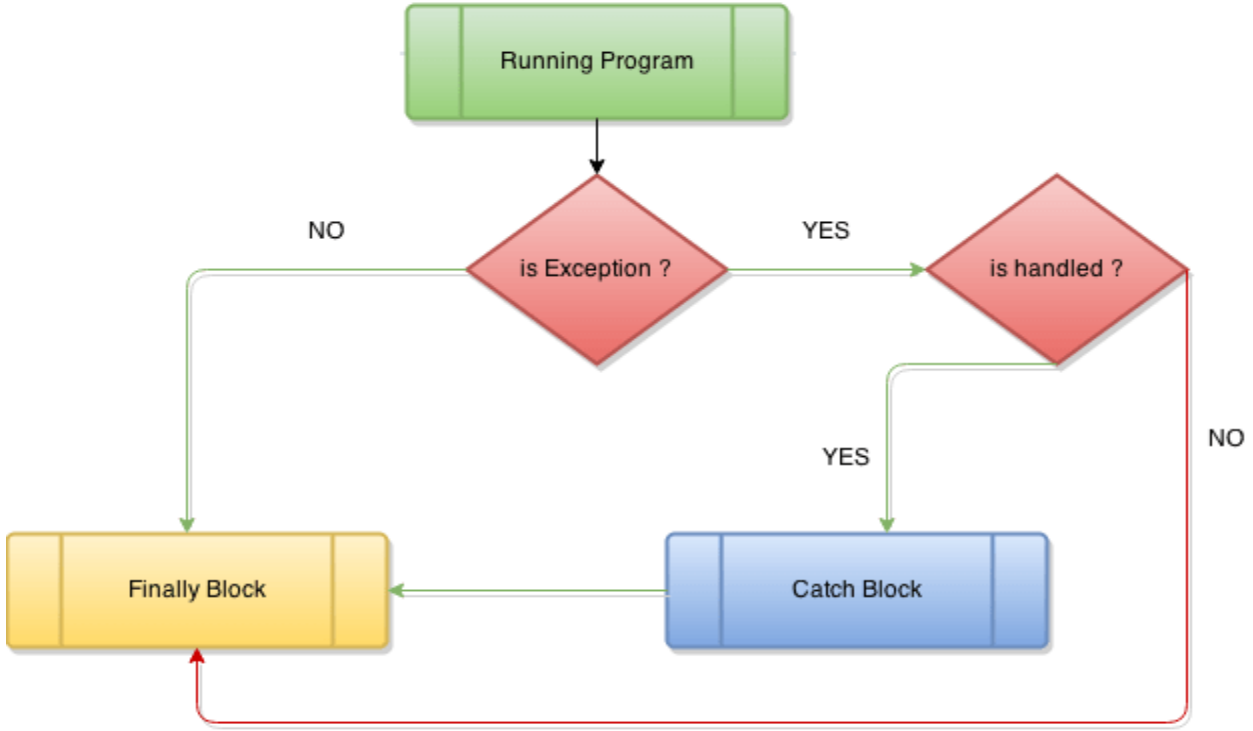


Fig. 2: Flowchart of exception handling

Exception Handling (Cont…)

- **Try-Catch Block**

```
public class Exception1
{
   public static void main(String args[])
   {
      try
      {
         int num[] = {1, 2, 3, 4, 5};
         System.out.println(num[10]);
      }
      catch(Exception e)
      {
         System.out.println("Something is wrong here");
      }
   }
}
```

Exception Handling (Cont…)

- **Try-Catch Block (Cont…)**
  - Output

```
Something is wrong here
```

Exception Handling (Cont…)

- **Try-Catch Block (Cont…)**

```
public class Exception2
{
   public static void main(String[] args)
   {
      try
      {
         int a[] = new int[4];
         a[4] = 100/0;
      }
      catch(ArithmeticException e)
      {
         System.out.println("Arithmetic Exception occurs");
      }
      catch(ArrayIndexOutOfBoundsException e)
      {
         System.out.println("ArrayIndexOutOfBounds Exception occurs");
      }
      catch(Exception e)
      {
         System.out.println("Parent Exception occurs");
      }
      System.out.println("Rest of the code");
   }
}
```

Exception Handling (Cont…)

- **Try-Catch Block (Cont…)**
  - Output

```
Arithmetic Exception occurs
Rest of the code
```

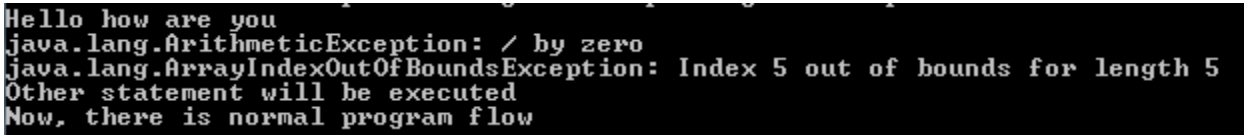
Exception Handling (Cont…)

- **Try-Catch Block (Cont…)**

```
class Exception3
{
   public static void main(String args[])
   {
      try{
            try
            {
               System.out.println("Hello how are you");
               int a = 100/0;
            }catch(ArithmeticException e)
             {
                System.out.println(e);
             }
            try
            {
               int b[] = new int[5];
               b[5] = 400;
            }catch(ArrayIndexOutOfBoundsException e)
             {
                System.out.println(e);
             }
            System.out.println("Other statement will be executed");
          }catch(Exception e)
           {
              System.out.println("Exception is Handeled");
           }
      System.out.println("Now, there is normal program flow");
   }
}
```

Exception Handling (Cont…)

- **Try-Catch Block (Cont…)**
  - Output



Exception Handling (Cont…)

- **Try-Catch Block (Cont…)**

```
class Exception22
{
   public static void main(String ar[])
   {
      try
      {
         try
         {
            System.out.println("Hello How are you?");
            int a = 100/0;
         }catch(ArithmeticException e)
          {
             System.out.println("Arithmetic Error" + e.getMessage());
          }
         try
         {
            int a[] = new int[4];
            a[3] = 40;
            int b = a[3]/0;
         }catch(ArrayIndexOutOfBoundsException e)
          {
             System.out.println("Array Error" + e.getMessage());
          }
       }catch(ArithmeticException e)
        {
           System.out.println("Arithmetic Error" + e.getMessage());
        }
     }
}
```

Exception Handling (Cont…)

- **Try-Catch Block (Cont…)**
  - Output

```
Hello how are you?
Arithmetic Error/ by zero
Arithmetic Error/ by zero
```

Exception Handling (Cont…)

- **Finally Block**

```
class Exception5
{
   public static void main(String args[])
   {
      try
      {
         int A = 50/0;
         System.out.println(A);
      }
      catch(NullPointerException e)
      {
         System.out.println(e);
      }
      finally
      {
         System.out.println("The Block of Finally is always executed");
      }
      System.out.println("Rest of the code, if any");
   }
}
```

Exception Handling (Cont…)

- **Finally Block (Cont…)**
  - Output

```
The Block of Finally is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exception5.main(Exception5.java:8)
```

Exception Handling (Cont…)

- **Finally Block (Cont…)**

```
class Exception6
{
   public static void main(String args[])
   {
      try
      {
         int A = 50/0;
         System.out.println(A);
      }
      catch(ArithmeticException e)
      {
         System.out.println(e);
      }
      finally
      {
         System.out.println("The Block of Finally is always executed");
      }
      System.out.println("Rest of the code, if any");
   }
}
```

Exception Handling (Cont…)

- **Finally Block (Cont…)**
  - Output

```
java.lang.ArithmeticException: / by zero
The Block of Finally is always executed
Rest of the code, if any
```

Exception Handling (Cont…)

- **Throw Keyword**

```
public class Exception7
{
   void pass(int mark)
   {
      if(mark < 15)
        throw new ArithmeticException("5 mark deduct in the next exam");

      else
        System.out.println("Normal case and continue");
   }
   public static void main(String args[])
   {
      Exception7 obj = new Exception7();
      obj.pass(12);
      System.out.println("I hope all of you will get good marks");
   }
}
```

Exception Handling (Cont…)

- **Throw Keyword (Cont…)**
  - Output

```
Exception in thread "main" java.lang.ArithmeticException: 5 mark deduct in the n
ext exam
        at Exception7.pass(Exception7.java:7)
        at Exception7.main(Exception7.java:15)
```

Exception Handling (Cont…)

- **Throw Keyword (Cont…)**

```
public class Exception25
{
   static void pass()
   {
      try
      {
         throw new ArithmeticException("How are you?");
      }
      catch(ArithmeticException e)
      {
         System.out.println("Exception caught inside method");
      }
   }

   public static void main(String args[])
   {
      pass();
   }
}
```

Exception Handling (Cont…)

- **Throw Keyword (Cont…)**
  - Output

```
Exception caught inside method
```

Exception Handling (Cont…)

- **Throw Keyword (Cont…)**

```
public class Exception26
{
   static void pass()
   {
      try
      {
         throw new ArithmeticException("How are you");
      }
      catch(ArithmeticException e)
      {
         System.out.println("Exception caught inside pass()");
         throw e;                    //Rethrowing the exception
      }
   }

   public static void main(String args[])
   {
      try
      {
         pass();
      }
      catch(NullPointerException e)
      {
         System.out.println("Exception caught inside main");
      }
   }
}
```

Exception Handling (Cont…)

- **Throw Keyword (Cont…)**
  - Output

```
Exception caught inside pass()
Exception in thread "main" java.lang.ArithmeticException: How are you
        at Exception26.pass(Exception26.java:8)
        at Exception26.main(Exception26.java:21)
```

Exception Handling (Cont…)

Throws Keyword

- Exception Propagating using Throws Keyword

```
class Exception8
{
   void NITP_method1() throws ArithmeticException
   {
      throw new ArithmeticException("This is how we calculation error");
   }
   void NITP_method2() throws ArithmeticException
   {
      NITP_method1();
   }
   void NITP_method3()
   {
      try{
            NITP_method2();
         }
         catch(ArithmeticException e)
         {
            System.out.println("ArithmeticException handled");
         }
   }
   public static void main(String args[])
   {
      Exception8 obj = new Exception8();
      obj.NITP_method3();
      System.out.println("Try to understand");
   }
}
```

Exception Handling (Cont…)

- **Throws Keyword (Cont…)**
  - Output

    ArithmeticException handled

    Try to understand

Exception Handling (Cont…)

Throws Keyword (Cont…)

- There are two cases:
  - **Case1:** We handle the exception using try-catch block.
  - **Case2:** We declare the exception, i.e. using throws with the method.
    - ✓When there is no exception
    - ✓When exception is occurred

Exception Handling (Cont…)

Throws Keyword (Cont…)

- There are two cases (Cont…):
  - **Case1: We handle the exception using try-catch block.**

```
import java.io.*;
class Hello
{
   void method() throws IOException
   {
      throw new IOException("Ther is an error");
   }
}

public class Exception9
{
   public static void main(String args[])
   {
      try
      {
         Hello obj = new Hello();
         obj.method();
      }catch(Exception e)
       {
          System.out.println("We have handled exception");
       }
      System.out.println("Normal flow of the program");
   }
}
```

Exception Handling (Cont…)

- **Throws Keyword (Cont…)**
  - Output

    We have handled exception

    Normal flow of the program

Exception Handling (Cont…)

Throws Keyword (Cont…)

- There are two cases (Cont…):
  - **Case2: We declare the exception, i.e. using throws with the method.**
    - ✓**When there is no exception**

```
import java.io.*;
class Hello
{
   void method() throws IOException
   {
      System.out.println("If Exception is not occurred");
   }
}

class Exception10
{
   public static void main(String args[]) throws IOException    //declaring exception
   {
      Hello obj = new Hello();
      obj.method();
      System.out.println("Normal Flow of the program");
   }
}
```

Exception Handling (Cont…)

- **Throws Keyword (Cont…)**
  - Output

    If Exception is not occurred

    Normal Flow of the program

Exception Handling (Cont…)

Throws Keyword (Cont…)

- There are two cases (Cont…):
  - **Case2: We declare the exception, i.e. using throws with the method.**
    - ✓**When exception is occurred**

```
import java.io.*;
class Hello
{
   void method() throws IOException
   {
      throw new IOException("there is an error");
   }
}

class Exception11
{
   public static void main(String args[]) throws IOException
   {
      Hello obj = new Hello();
      obj.method();
      System.out.println("Normal Flow of the program");
   }
}
```

Exception Handling (Cont…)

- **Throws Keyword (Cont…)**
  - Output

```
Exception in thread "main" java.io.IOException: there is an error
        at Hello.method(Exception11.java:7)
        at Exception11.main(Exception11.java:16)
```

Exception Handling (Cont…)

Throw Keyword	Throws Keyword
Throw is used to explicitly throw an exception.	Throws is used to declare an exception.
It is followed by an instance.	It is followed by class.
It is used within the method.	It is used with method signature.
We cannot throw multiple exceptions.	We can declare multiple exceptions. For example: public void method() throws IOException, SQLException.

Exception Handling (Cont…)

User Defined Exception

- Java provides us facility to create our own exceptions.
- **User Defined Exception** or **Custom Exception** is creating our own exception class.
- We throw that exception using ‘throw’ keyword.
- This can be done by extending the class Exception or RuntimeException.

Exception Handling (Cont…)

- **User Defined Exception (Cont...)**

```
class User_defined extends Exception
{
    int X;
    User_defined(int Y)
    {
      X=Y;
    }
    public String toString()
    {
      return ("Exception Number is =  " + X);
    }
}

class Exception12
{
    public static void main(String args[])
    {
      try{
        throw new User_defined(155551);
      }catch(User_defined e)
       {
         System.out.println(e);
       }
    }
}
```

Exception Handling (Cont…)

- **User Defined Exception (Cont...)**
  - Output

```
Exception Number is = 155551
```

Exception Handling (Cont…)

- **User Defined Exception (Cont...)**

```
class User_defined extends Exception
{
   String strA;
   User_defined(String strB)
   {
        strA=strB;
   }
   public String toString()
   {
        return ("User_defined Exception Occurred: "+ strA);
   }
}

class Exception13
{
   public static void main(String args[])
   {
        try{
                System.out.println("This is the starting of try block");
                throw new User_defined("This is the error Message");
        }
        catch(User_defined e){
                System.out.println("This is the starting of Catch Block");
                System.out.println(e);
        }
   }
}
```

Exception Handling (Cont…)

- **User Defined Exception (Cont...)**
  - Output

```
This is the starting of try block
This is the starting of Catch Block
User_defined Exception Occurred: This is the error Message
```

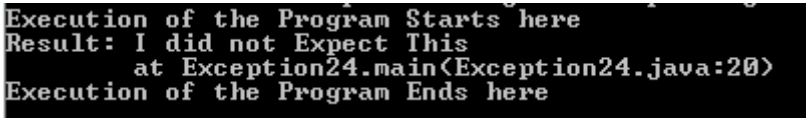
Exception Handling (Cont…)

- **User Defined Exception (Cont...)**

```
class Result extends RuntimeException
{
   Result()
   {
      super("Please focus on Career");
   }
   Result(String msg)
   {
      super(msg);
   }
}
class Exception24
{
   public static void main(String args[])
   {
      int mark = 50;
      System.out.println("Execution of the Program Starts here");
      try{
            if(mark < 70)
               throw new Result("I did not Expect This");
            else
               System.out.println("You are a good student");
         }catch(Result e)
          {
             e.printStackTrace();
          }
      System.out.println("Execution of the Program Ends here");
   }
}
```

Exception Handling (Cont…)

- **User Defined Exception (Cont...)**
  - Output



Exception and Inheritance

Introduction to Method Overriding

- When a method in a subclass has the same name, same parameters or signature and same return type as a method defined in its super-class, then the method in the subclass is said to override the method in the super-class.

- Overriding allows a subclass or child class to provide a specific implementation of a method that is already defined by one of its super-classes (parent classes).

Exception and Inheritance (Cont...)

- There are mainly two problems:
  - Problem 1: When Superclass doesn't declare an exception
    - ✓ **Case 1:** Subclass declares checked exception
    - ✓ **Case 2:** Subclass declares unchecked exception
  - Problem 2: When Superclass declares an exception
    - ✓ **Case 1:** Subclass declares exceptions other than the child exception of the Superclass declared exception.
    - ✓ **Case 2:** Subclass declares an child exception of the Superclass declared exception.
    - ✓ **Case 3:** Subclass declares without exception.

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 1: **Case 1: Subclass declares checked exception**

```
import java.io.*;
class Superclass
{
   void method()                        //Superclass doesn't declare any exception
   {
      System.out.println("This is our Superclass without exception");
   }
}

class Exception14 extends Superclass
{
   void method() throws IOException  //Subclass declares Checked Exception i.e. IOException
   {
      System.out.println("This is our Subclass");
   }
   public static void main(String args[])
   {
      Superclass obj = new Exception14();
      obj.method();
   }
}
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 1: **Case 1: Subclass declares checked exception (Cont…)**

    Output

```
Exception14.java:13: error: method() in Exception14 cannot override method() in
Superclass
   void method() throws IOException  //Subclass declares Checked Exception i.e.
IOException
        ^
  overridden method does not throw IOException
1 error
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 1: **Case 2: Subclass declares unchecked exception**

```
class Superclass
{
   void method()                                  //Superclass doesn't declare any exception
   {
      System.out.println("This is our Superclass without exception");
   }
}

class Exception15 extends Superclass
{
   void method() throws ArithmeticException  //Subclass declares unchecked Exception
   {
      System.out.println("This is our Subclass");
   }
   public static void main(String args[])
   {
      Superclass obj = new Exception15();
      obj.method();
   }
}
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 1: **Case 2: Subclass declares unchecked exception (Cont..)**

Output

```
This is our Subclass
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 1: Subclass declares exceptions other than the child exception of the Superclass declared exception**

```
class Superclass     //Superclass with Exception
{
   void method() throws RuntimeException
   {
      System.out.println("This is our Superclass with Exception");
   }
}

class Exception16 extends Superclass
{
   void method() throws Exception  //Exception is not child of RuntimeException
   {
      System.out.println("This is our Subclass");
   }
   public static void main(String args[])
   {
      Superclass obj = new Exception16();
      obj.method();
   }
}
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 1: Subclass declares exceptions other than the child exception of the Superclass declared exception (Cont…)**

Output

```
Exception16.java:12: error: method() in Exception16 cannot override method() in
Superclass
   void method() throws Exception  //Exception is not child of RuntimeException
        ^
  overridden method does not throw Exception
1 error
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 2: Subclass declares a child exception of the Superclass declared exception**

```
class Superclass     //Superclass with Exception
{
   void method() throws RuntimeException
   {
      System.out.println("This is our Superclass with exception");
   }
}

class Exception17 extends Superclass
{
   void method() throws ArithmeticException  //ArithmeticException is child exception of RuntimeException
   {
      System.out.println("This is our Subclass");
   }
   public static void main(String args[])
   {
      Superclass obj = new Exception17();
      obj.method();
   }
}
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 2: Subclass declares a child exception of the Superclass declared exception (Cont…)**

Output

This is our Subclass

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 3: Subclass declares without exception**

```
class Superclass     //Superclass with Exception
{
   void method() throws RuntimeException
   {
      System.out.println("This is our Superclass with exception");
   }
}

class Exception18 extends Superclass
{
   void method()    //Subclass without Exception
   {
      System.out.println("This is our Subclass");
   }
   public static void main(String args[])
   {
      Superclass obj = new Exception18();
      obj.method();
   }
}
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 3: Subclass declares without exception (Cont…)**

Output

```
This is our Subclass
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 3: Subclass declares without exception (Cont…)**

```
import java.io.*;
class Superclass     //Superclass declares exception
{
   void method() throws IOException
   {
      System.out.println("This is our Superclass with exception");
   }
}

class Exception19 extends Superclass
{
   void method()     //Subclass without exception
   {
      System.out.println("This is our Subclass");
   }
   public static void main(String args[])
   {
      Superclass obj = new Exception19();
      try
      {
         obj.method();
      }catch (IOException e)
       {
          e.printStackTrace();
       }
   }
}
```

Exception and Inheritance (Cont...)

- There are mainly two problems (Cont…):
  - Problem 2: **Case 3: Subclass declares without exception (Cont…)**

    Output

    This is our Subclass

Redirecting Exception

- When there is no appropriate catch block to handle the exception (checked) that was thrown by an object, the compiler does not compile the program.
- To overcome the above issue, Java allows us to redirect exceptions by using the “throws” keyword that have been raised up the call stack.
- Thus, an exception thrown by a method can be handled either in the method itself or passed to a different method in the call stack.

Redirecting Exception (Cont...)

```
public class Exception23
{
   static void Method1(String s1, String s2) throws Exception
   {
      int a = Integer.parseInt(s1);
      int b = Integer.parseInt(s2);
      int c = Method2(a, b);
      System.out.println("Our Result is: " + a + "/" + b + "=" + c );
   }
   static int Method2(int x, int y) throws Exception
   {
      if (y == 0)
      {
         throw new Exception("Divide by Zero is not Possible");
      }
      return x/y;
   }
   public static void main(String args[])
   {
      System.out.println("Program is Executed From this Line");
      try
      {
         Method1(args[0], args[1]);
      }catch(Exception e)
       {
          System.out.println (e.getMessage ());
          e.printStackTrace ();
       }
      System.out.println("Program Ends here");
   }
}
```

Redirecting Exception (Cont...)

- **Output**

```
C:\Users\SUYEL\Desktop\Java\Program\Exception>javac Exception23.java

C:\Users\SUYEL\Desktop\Java\Program\Exception>java Exception23 30 6
Program is Executed From this Line
Our Result is: 30/6=5
Program Ends here

C:\Users\SUYEL\Desktop\Java\Program\Exception>java Exception23 30 0
Program is Executed From this Line
Divide by Zero is not Possible
java.lang.Exception: Divide by Zero is not Possible
        at Exception23.Method2(Exception23.java:15)
        at Exception23.Method1(Exception23.java:8)
        at Exception23.main(Exception23.java:24)
Program Ends here

C:\Users\SUYEL\Desktop\Java\Program\Exception>java Exception23 30 gh
Program is Executed From this Line
For input string: "gh"
java.lang.NumberFormatException: For input string: "gh"
        at java.base/java.lang.NumberFormatException.forInputString(Num
Exception.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:660)
        at java.base/java.lang.Integer.parseInt(Integer.java:778)
        at Exception23.Method1(Exception23.java:7)
        at Exception23.main(Exception23.java:24)
Program Ends here
```

Rethrowing Exception

- If a catch block cannot handle the particular exception it has caught, we can rethrow the exception.
- As the exception has already been caught at the scope in which the rethrow expression occurs, it is rethrown out to the next enclosing try block. So, it cannot be handled by catch blocks at the scope in which the rethrow expression occurred.
- Any catch blocks for the enclosing try block have an opportunity to catch the exception.

Rethrowing Exception (Cont...)

```
public class Exception20
{
   public static void Hello1() throws Exception
   {
      System.out.println("The Exception present in method Hello1");
      throw new Exception("thrown from method Hello1");
   }
   public static void Hello2() throws Throwable
   {
      try{
         Hello1();
      }catch(Exception e){
         System.out.println("Inside method Hello2");
         throw e;
       }
   }
   public static void main(String[] args) throws Throwable
   {
      try{
         Hello2();
      }catch(Exception e){
         System.out.println("Caught in main method");
       }
   }
}
```

Rethrowing Exception (Cont...)

- **Output**

  The Exception present in method Hello1

  Inside method Hello2

  Caught in main method


thank
you . . .

Slides are prepared from various sources, such as Book, Internet Links and many more.