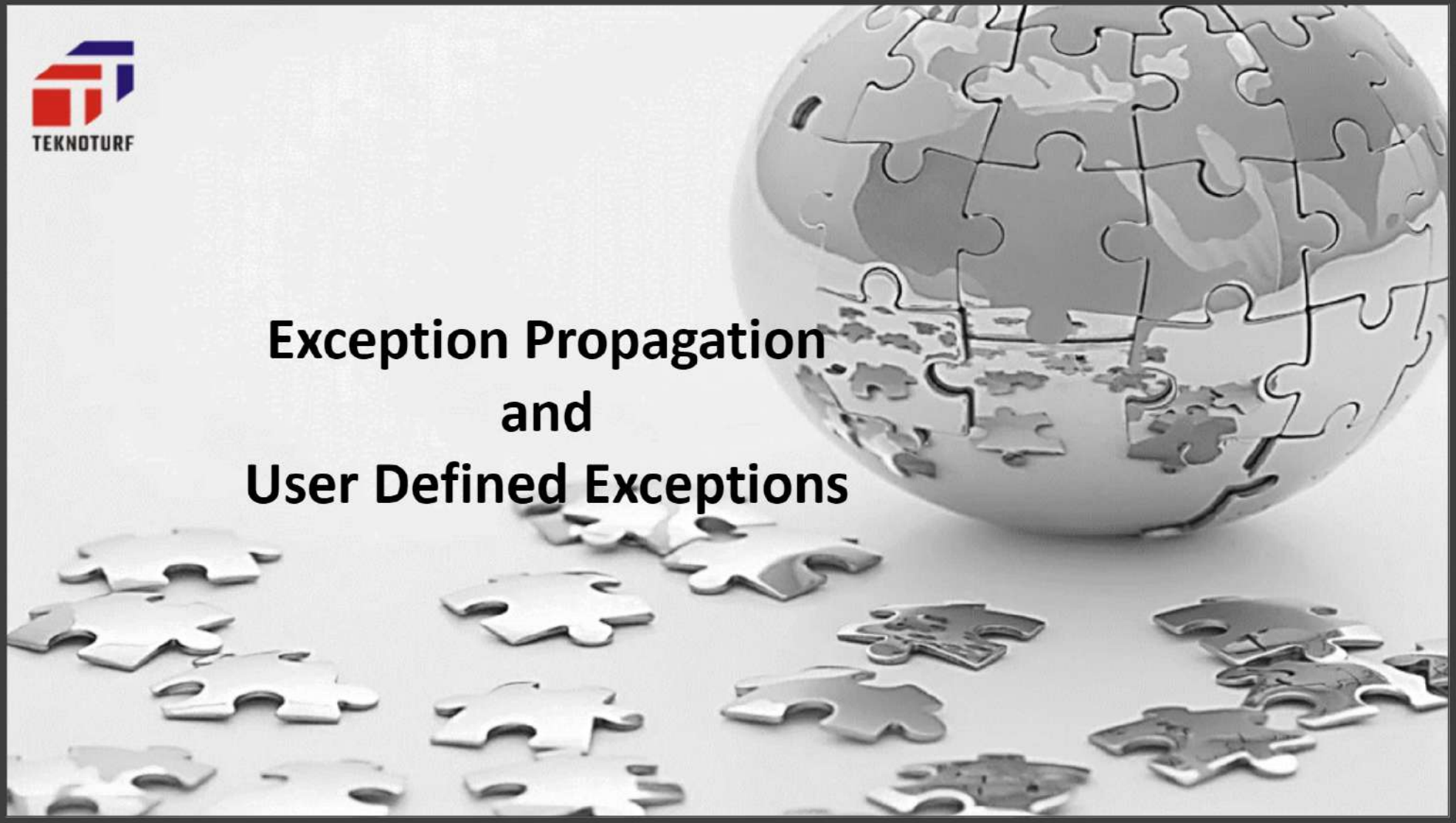

TEKNOTURF
Exception Propagation
and
User Defined Exceptions

In this module you will learn

TEKNOTURF

- Exception Propagation
- User defined Exceptions
- Try with resources

Nested try

TEKNOTURF

We can have nested try and catch blocks(a try block inside another try block)

If an inner try statement does not have a matching catch statement for a particular exception, the control is transferred to the next try statement's catch handlers

This continues until one of the catch statements succeeds, or until all of the nested try statements are done in

If none of the catch statements match, then the Java runtime system will handle the exception.

Nested try

TEKNOTURF

```
import java.io.*;
public class NestedTry{
 public static void main (String args[])throws IOException  {
   int num1=2,num2=0,res=0;
   try{
       FileInputStream fis=null;
       fis = new FileInputStream (new File (args[0]));
       try{
                res=num1/num2;
                System.out.println("The result is"+res);
       }
       catch(ArithmeticException e){
                System.out.println("divided by Zero");
       }
      }
```

```
catch (FileNotFoundException e){
       System.out.println("File not found!");
 }
catch(ArrayIndexOutOfBoundsException   e){
       System.out.println("Array index is Out
                        of bound! Argument required");
 }
 catch(Exception e){
   System.out.println("Error.."+e);
  }
 }
}
```

Nested try

TEKNOTURF

If the above program is executed without giving any file name, the output will be "Array index is Out of bound! Argument required " (available in outer try)

If the above program is executed by giving a file name that does not exist, the output will be "File not found! " (available in outer try)

If the above program is executed by giving a file name that exists, the output will be "divided by Zero " (available in inner try).

Call Stack Mechanism

TEKNOTURF

If an exception is not handled in the current try-catch block, it is thrown to the caller of that method

If the exception gets back to the main method and is not handled there, the program is terminated abnormally

A stack trace provides information on the execution history of the current thread and lists the names of the classes and methods that were called at the point when the exception occurred

The following code shows how to call the getStackTrace method on the exception object

```
catch (Exception cause) {
    StackTraceElement elements[] = cause.getStackTrace();
    for (int i = 0, n = elements.length; i < n; i++) {
        System.err.println(elements[i].getFileName()
            + ":" + elements[i].getLineNumber()
            + ">> "
            + elements[i].getMethodName() + "()");
    }
}
```

Rules for try-catch-finally

TEKNOTURF

For each try block there can be zero or more catch blocks but only one finally block

The catch blocks and finally block must always appear in conjunction with a try block

A try block must be followed by either at least one catch block or one finally block

The order of the exception handlers in the catch block must be from the most specific exception

throw and throws usage

TEKNOTURF

```
throw <exception reference>;
```

```
throw new ArithmeticException("Division attempt by 0");
```

```
Method() throws <ExceptionType_1>,..., <ExceptionType_n> {
        //statements
}
```

throws and throw - Example

TEKNOTURF

```
public class DivideByZeroException {

   public static void main(String[] args) {
      try{
                int result  = divide(100,10);
                result  = divide(100,0);
      System.out.println("result : "+result);
      }
      catch(ArithmeticException e){
                System.out.println("Exception : "+
      e.getMessage());
      }
   }

   public static int divide(int totalSum, int
      totalNumber) throws   ArithmeticException
   {
      int quotient = -1;
      System.out.println("Computing Division.");
```

```
try{
      if(totalNumber == 0){
          throw new ArithmeticException("Division attempt
                                                  by 0");
      }
      quotient  = totalSum/totalNumber;
}
 finally{
      if(quotient != -1){
      System.out.println("Finally Block Executes");
      System.out.println("Result : "+ quotient);
      }else{
      System.out.println("Finally Block Executes. Exception
                              Occurred");
      }
}
 return quotient;
}
}
```

Handle or Declare Rule

TEKNOTURF

Handle the exception by using the try-catch-finally block

Declare that the code causes an exception by using the throws clause

- void trouble() throws IOException { ... }
- void trouble() throws IOException, MyException { ... }

You do not need to declare runtime exceptions or errors

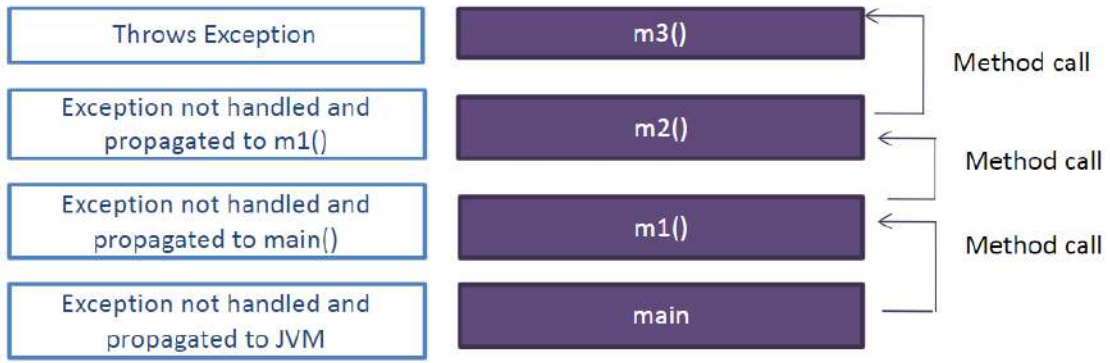
You can choose to handle runtime exceptions

Exception Propogation

TEKNOTURF

Exception propagation is a way of propagating exception from a method to the previous method in the call stack until it is caught.

If uncaught thrown to JVM and program gets terminated



Exception Propogation

TEKNOTURF

```
public class ExceptionPropogationDemo
{
   public static void main(String a[])  {
     m1(); //Line 4
   }
   public static void m1() {
     m2();  //Line 7
   }
   public static void m2() {
     m3();  //Line 10
   }
   public static void m3() {
    throw new ArithmeticException(); //Line13
   }
}
```

Output :

```
Exception in thread "main" java.lang.ArithmeticException
at
ExceptionPropogationDemo.m3(ExceptionPropogationDemo.java:13)
at
ExceptionPropogationDemo.m2(ExceptionPropogationDemo.java:10)
at ExceptionPropogationDemo.m1(ExceptionPropogationDemo.java:7)
at
ExceptionPropogationDemo.main(ExceptionPropogationDemo.java:4)
```

Exception Propogation

TEKNOTURF

```
public class ExceptionPropogationDemo
{
    public static void main(String a[]) throws IOException  {
        m1();
    }
    public static void m1() throws IOException {
        m2();
    }
    public static void m2() throws IOException {
        m3();
    }
    public static void m3() throws IOException {
        throw new IOException();
    }
}
```

```
Exception in thread "main" java.io.IOException
at ExceptionPropogationDemo.m3(ExceptionPropogationDemo.java:15)
at ExceptionPropogationDemo.m2(ExceptionPropogationDemo.java:12)
at ExceptionPropogationDemo.m1(ExceptionPropogationDemo.java:9)
at ExceptionPropogationDemo.main(ExceptionPropogationDemo.java:6)
```

Method Overriding and Exceptions

TEKNOTURF

The overriding method can throw

- No exceptions
- One or more of the exceptions thrown by the overridden method
- One or more subclasses of the exceptions thrown by the overridden method

The overriding method cannot throw

- Additional exceptions not thrown by the overridden method
- Super classes of the exceptions thrown by the overridden method

Method Overriding and Exceptions

TEKNOTURF

Example 1 :

```
class Parent  {
  public void method1() throws Exception
  {
      System.out.println("Parent");
  }
}
public class Child extends Parent  {
  public void method1() throws Exception
  {
      System.out.println("Child");

   }
   public static void main(String a[]) throws Exception
   {
       Parent p=new Child();
      p.method1();
    }
}
```

Example 2 :

```
class Parent {
  public void method1() throws Exception
  {
      System.out.println("Parent");
  }
}
public class Child extends Parent  {
 public void method1() throws ArithmeticException
  {
      System.out.println("Child");
   }
   public static void main(String a[]) throws Exception
   {
       Parent p=new Child();
      p.method1();
    }
}
```

Method Overriding and Exceptions

TEKNOTURF

Example 1 :

```
class Parent  {
  public void method1() throws Exception
  {
      System.out.println("Parent");
  }
}
public class Child extends Parent  {
  public void method1()
  {
      System.out.println("Child");

   }
   public static void main(String a[]) throws Exception
   {
       Parent p=new Child();
      p.method1();
    }
}
```

Example 2 :

```
class Parent {
  public void method1() throws ArithmeticException
  {
      System.out.println("Parent");
  }
}
public class Child extends Parent  {
 public void method1()  throws Exception
  {
      System.out.println("Child");
   }
   public static void main(String a[]) throws Exception
   {
       Parent p=new Child();
      p.method1();
    }
}
```

Compile Time Error

User Defined Exception

TEKNOTURF

Though Java provides an extensive set of in-built exceptions, there are cases in which we may need to define our own exceptions in order to handle the various application specific errors.

While defining a user defined exception, we need to extend the Exception class.

User Defined Exceptions - Example

TEKNOTURF

```
class InvalidAgeException extends Exception  {
   public InvalidAgeException(String message)
   {
      super(message);
   }
}

public class CustomException {
   public static void validateAge(int age) throws
                                    InvalidAgeException  {
        if(age < 18)
        {
              throw new InvalidAgeException("Not a
                                        valid Age to
    vote");
        }
```

```
      else {
        System.out.println("Eligible to vote");
      }
   }
   public static void main(String arg[])  {
      try  {
        validateAge(15);
      }
      catch(InvalidAgeException e)  {
        System.out.println(e.getMessage());
      }
   }
}
```