Java - Exception Propagation

Exception Propagation

Exception propagation refers to movement of exception event from nested try or nested methods calls. A try block can be nested within another try block. Similarly a method can call another method where each method can handle exception independently or can throw checked/unchecked exceptions. Whenever an exception is raised within a nested try block/method, its exception is pushed to Stack. The exception propagates from child to parent try block or from child method to parent method and so on.

Syntax - Nested Try Block

The syntax for nested catch blocks looks like the following –

```
try { // parent try block
   try {  // child try block

   }
   catch(ExceptionType1 e1){  // child catch block

   }
} catch (ExceptionType2 e1) { // parent catch block

}
```

Syntax - Nested Method Calls

The syntax for nested method calls looks like the following –

```
method1(){ // parent method
   try { // parent try block
      method2();
   } catch (ExceptionType2 e1) { // parent catch block
}
method2(){ // child method
   // code to throw Exception
   // this exception will be handled by parent method
}
```

The previous statements demonstrate two try/catch blocks and methods, but you can have any number of them. If an exception occurs in the protected child code, the exception is thrown to the catch block of the child list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes up to the parent catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Rules for Exception Propagation in Java

- Child catch block should have specific exception for better code clarity. Parent catch block can have more generic exception handled so that if child catch block is not able to handle the exception then parent catch block can handle it.
- There in no restriction on exception hiearchy to be used in child vs parent catch block.
- If a exception is handled correctly in child catch block, then in parent, another exception can be raised and handled.

Java Exception Propagation Example

Here is code segment showing exception event propagation from child to parent. In this example, we're creating an error by dividing a value by 0 in a child method. The child method throws the exception. Now in parent method, in try block, we're handling the exception and printing the error message.

</> Open Compiler

```
package com.tutorialspoint;

public class ExcepTest {

   public static void main(String args[]) {
      int a = 3;
      int b = 0;
      try {
         System.out.println("result:" + divide(a,b));
      }catch(ArithmeticException e) {
         System.out.println(e.getMessage());
      }
```

```
    }

    private static int divide(int a, int b) {
        return a / b;
    }
}
```

Output

```
/ by zero
```

More Examples

Example 1

Here is code segment showing exception event propagation from child to parent. In this example, we're creating an error by dividing a value by 0 in a child method. The child method throws the exception. Now in parent method, we're not handling the exception. The JVM will intercept the exception and prints the error message.

</> Open Compiler

```
package com.tutorialspoint;

public class ExcepTest {

    public static void main(String args[]) {
        int a = 3;
        int b = 0;
        System.out.println("result:" + divide(a,b));
    }

    private static int divide(int a, int b) {
        return a / b;
    }
}
```

Output

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
   at com.tutorialspoint.ExcepTest.divide(ExcepTest.java:12)
   at com.tutorialspoint.ExcepTest.main(ExcepTest.java:8)
```

Example 2

Here is code segment showing exception event propagation to stop within child itself instead of flowing to parent. In this example, we're creating an error by dividing a value by 0 in a child method. The child method is handling the exception. Now in parent method, we're not getting any exception.

</> Open Compiler

```
package com.tutorialspoint;

public class ExcepTest {

   public static void main(String args[]) {
      int a = 3;
      int b = 0;
      System.out.println("result:" + divide(a,b));
   }

   private static int divide(int a, int b) {
      try {
         return a / b;
      }catch(ArithmeticException e) {
         System.out.println(e.getMessage());
      }
   return 0;
   }
}
```

Output

```
/ by zero
result:0
```