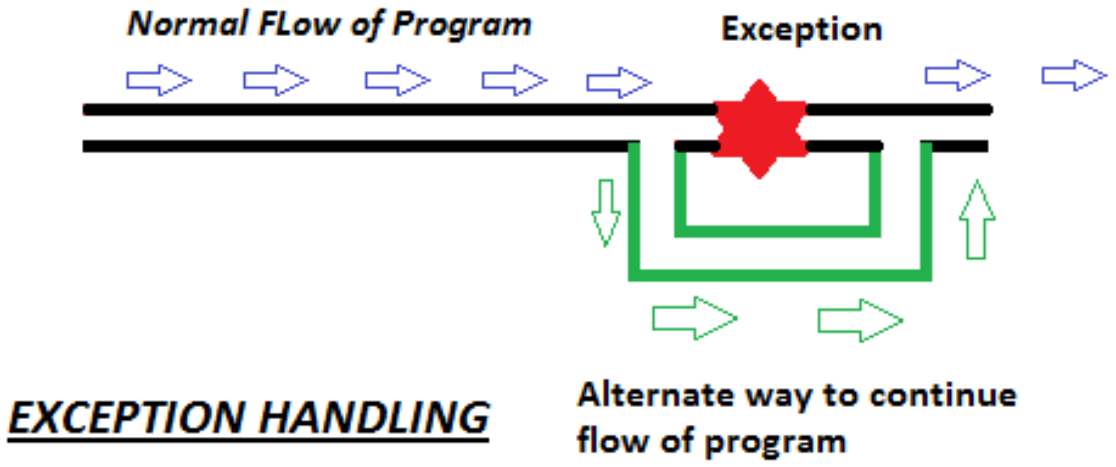
Bài 11
Ngoại lệ

Mục tiêu

- Giới thiệu Ngoại lệ - Exception
- Quản lý ngoại lệ
- Khối ngoại lệ



Giới thiệu

- Java là ngôn ngữ lập trình **mạnh mẽ** và **hiệu quả**.
- Các tính năng như lớp, đối tượng, thừa kế… mang lại khả năng **linh hoạt và an toàn**.
- Tuy nhiên, trong điều kiện nhất định, mã lệnh có thể **gây lỗi** hoặc **cư xử sai** với các nghiệp vụ đề ra.
- Những tình huống này có thể là **mong đợi** hoặc **không mong muốn**.

Giới thiệu

- **Nạn nhân** khi gặp tình huống này chính là **người dùng**.
- Java cung cấp các khái niệm về xử lý ngoại lệ, thay vì chương trình **dừng đột ngột** có thể hiển thị **thông báo** thích hợp.

Giới thiệu

Một ngoại lệ là một sự kiện hoặc một điều kiện bất thường diễn ra trong thời gian thực hiện chương trình dẫn đến sự gián đoạn luồng bình thường.

Một ngoại lệ có thể xuất hiện bởi các nguyên nhân:

- Khi dữ liệu không hợp lệ
- Khi file cần mở không tồn tại
- Kết nối mạng bị mất trong quá trình đang truyền thông điệp
- JVM quả tải bộ nhớ

Giới thiệu

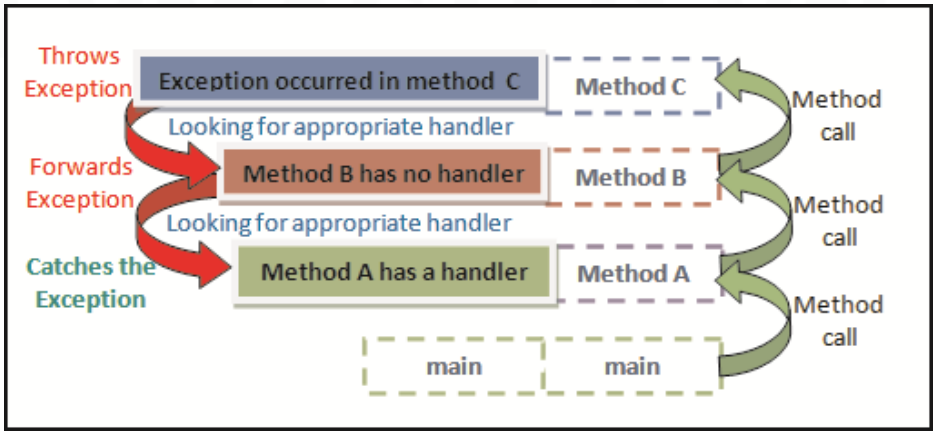
Quá trình tạo một đối tượng ngoại lệ và đưa nó và hệ thống khi chương trình hoạt động gọi là ném một ngoại lệ.

- Khi ngoại lệ được **"ném ra"** bởi phương thức, hệ thống runtime sẽ cố gắng tìm khối mã dành xử lý nó.
- Khối mã **phù hợp** với ngoại lệ tương ứng có thể xử lý hàng loạt phương thức khác để giải quyết lỗi đó.

Giới thiệu

Các lời gọi phương thức có thể được thực thi liên tiếp dạng chồng. Khi ngoại lệ xảy ra sẽ được xử lý trả về theo trình tự.

Sau đây hình ảnh là ví dụ các phương thức gọi chồng (stack).



Giới thiệu

Một ngoại lệ có thể được đẩy ra với các lý do:

- Một lệnh **throw** statement bên trong một phương thức đã kích hoạt.
- Một bất thường trong quá trình thực thi bị phát hiện bởi JVM như là:
  - Vi phạm ngữ nghĩa thông thường như là thực thi biểu thức một số nguyên đem chia cho 0.
  - Lỗi xuất hiện trong khi liên kết, tải, hoặc khởi tạo một phần chương trình sẽ ném thể hiện của lớp con như là LinkageError.
  - JVM bị ngăn cản thực thi mã do một lỗi hoặc nguồn tài nguyên giới hạn nội bộ mà sẽ ném thể hiện của lớp con như VirtualMachineError.
- Một ngoại lệ bất đồng bộ xảy ra.

Giới thiệu

Sử dụng ngoại lệ để quản lý lỗi có các ưu điểm sau:

Giới thiệu

Java cung cấp 2 kiểu Ngoại lệ:

Checked Exceptions

- Là ngoại lệ **được kiểm tra** tại thời gian biên dịch (compile-time)

Unchecked Exceptions

- Là ngoại lệ **không được kiểm tra** tại thời gian biên dịch (compile-time)

Checked Exceptions

- Xem xét tình huống chương trình **cần đọc** dữ liệu từ file trên thiết bị (ổ cứng, đĩa….).
- Có thể file, đường dẫn tới tệp tin **không tồn tại**.
- **Java Compile buộc** lập trình viên phải viết mã kiểm soát tại thời gian biên dịch, một ngoại lệ phải được xử lý như ***java.io.FileNotFoundException***.

```
try {
   String input = reader.readLine();
   System.out.println("You typed : "+input); // Exception prone area
   } catch (IOException e) {
    e.printStackTrace();
}
```

Giới thiệu

Một số **Checked Exceptions**:

- FileNotFoundException
- ParseException
- ClassNotFoundException
- CloneNotSupportedException
- InstantiationException
- InterruptedException
- NoSuchMethodException
- NoSuchFieldException

Unchecked Exceptions

- Xem xét tình huống ngoại lệ xuất phát từ lỗi lập trình, lỗi logic hoặc cách sử dụng API không đúng.
- Lỗi do phần cứng hoặc hệ thống bị trục trặc khiến việc truy cập tập tin trong quá trình runtime bị cản trở.

Unchecked Exceptions

- Chương trình truy cập đến đối tượng mà do một tình huống đặc biệt nó bị *null*. Lỗi toán học trong biểu thức, lỗi truy cập phần tử tập hợp…

```
package com.beingjavaguys.core;

public class ExceptionTest {

public static void main(String[] args) {
int i = 10/0;
}
}
```

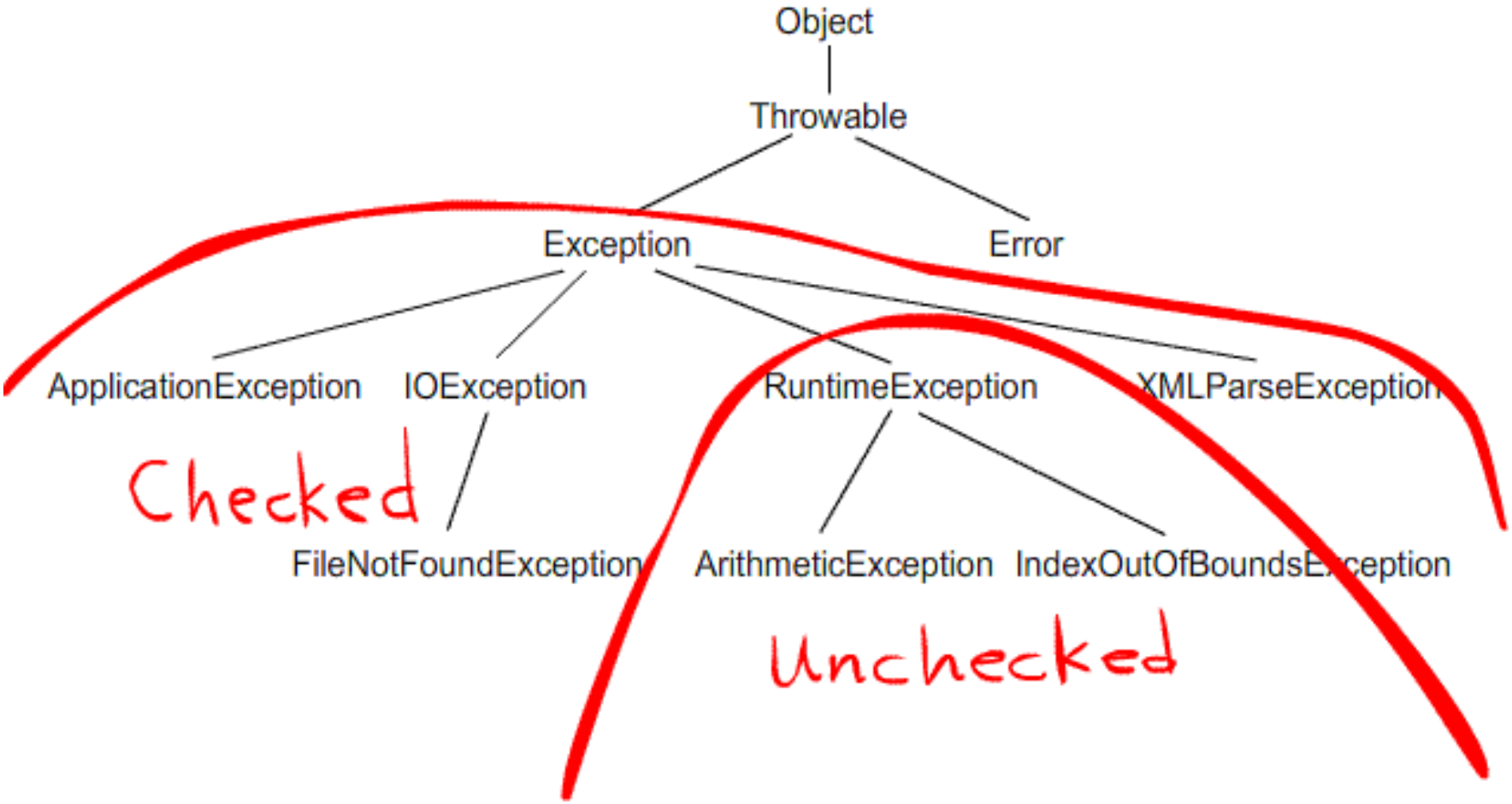
```
package com.beingjavaguys.core
public class ExceptionTest {

public static void main(String[] args) {
int arr[] = {'0','1','2'};
System.out.println(arr[4]);
}
}
```

```
package com.beingjavaguys.core;
import java.util.ArrayList;

public class ExceptionTest {

public static void main(String[] args) {
String string = null;
System.out.println(string.length());
}
}
```

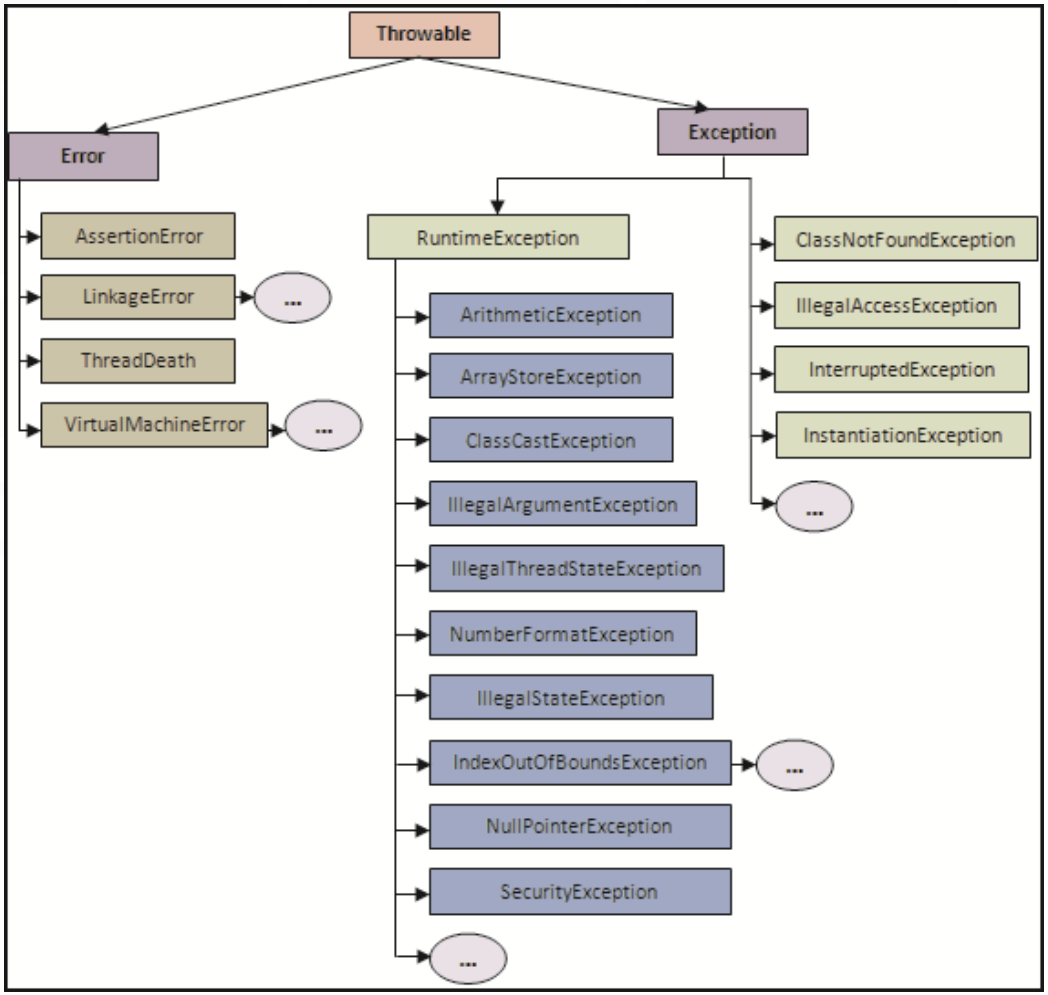
Giới thiệu

- **Error:** Là trường hợp xảy ra **bên ngoài** ứng dụng, ứng dụng thường không thể dự đoán hoặc phục hồi từ lỗi. VD: đường dẫn tới tệp tin tồn tại nhưng không thể đọc được do lỗi phần cứng hoặc hệ thống trục trặc.
- **Exception:** Là trường hợp xảy ra **bên trong** ứng dụng, ứng dụng thường không thể dự đoán hoặc phục hồi từ ngoại lệ. VD: khai báo và lấy về đối tượng trả từ thư viện API, tuy nhiên giá trị lại là *null*, mã nguồn tiếp theo sử dụng đối tượng nên sinh ngoại lệ….

Giới thiệu


Object
Throwable
Exception
Error
ApplicationException
IOException
RuntimeException
XMLParseException
Checked
FileNotFoundException
ArithmeticException
IndexOutOfBoundsException
Unchecked

Dòng họ class Throwable



Quản lý ngoại lệ

DEVMASTER.VN

- Các phương có tiềm ẩn lỗi về tham số truyền vào, giá trị trả về cần cài đặt ném ngoại lệ.
- Mã nguồn gọi phương thức phải nhận thức về ngoại lệ mà phương thức có thể ném ra.
- Điều này giúp người gọi quyết định làm thế nào để xử lý khi chúng xảy ra
- Sẽ có nhiều hơn một ngoại lệ xảy ra ở bất kỳ đâu trong một chương trình.
- Cần bổ sung thêm mã xử lý ngoại lệ runtime trong mỗi khai báo phương thức có thể làm giảm tính rõ ràng của chương trình.
- Trình biên dịch không yêu cầu người dùng phải bắt hoặc chỉ định ngoại lệ runtime, tuy vậy để đảm bảo chương trình rõ ràng cần phải xử lý.

Khối ngoại lệ

DEVMASTER.VN

Syntax

```
try{
    // statements that may raise exception
    // statement 1
    // statement 2
}
catch(<exception-type> <object-name>){
    // handling exception
    // error message
}
```

Khối ngoại lệ

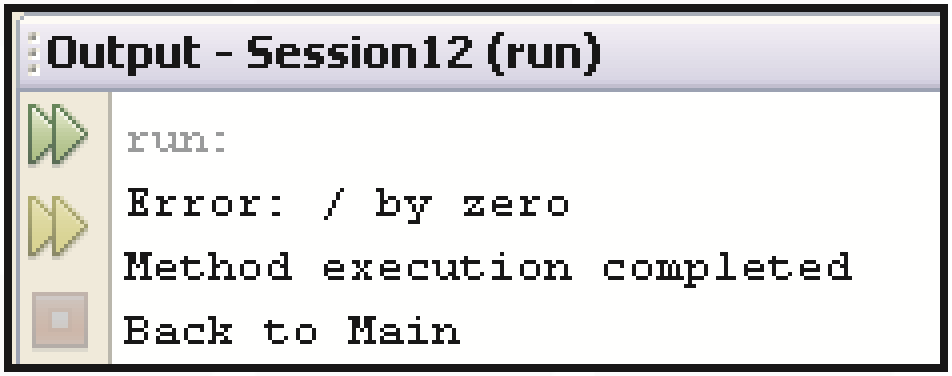
```
package session12;
class Mathematics {
    /**
     * Divides two integers
     * @param num1 an integer variable storing value of first number
     * @param num2 an integer variable storing value of second number
     * @return void
     */
    public void divide(int num1, int num2) {
        // Create the try block
        try {
            // Statement that can cause exception
            System.out.println(“Division is: “ + (num1/num2));
        }
        catch(ArithmeticException e){ //catch block for ArithmeticException
            // Display an error message to the user
            System.out.println(“Error: “+ e.getMessage());
        }
        // Rest of the method
        System.out.println(“Method execution completed”);
    } }
```

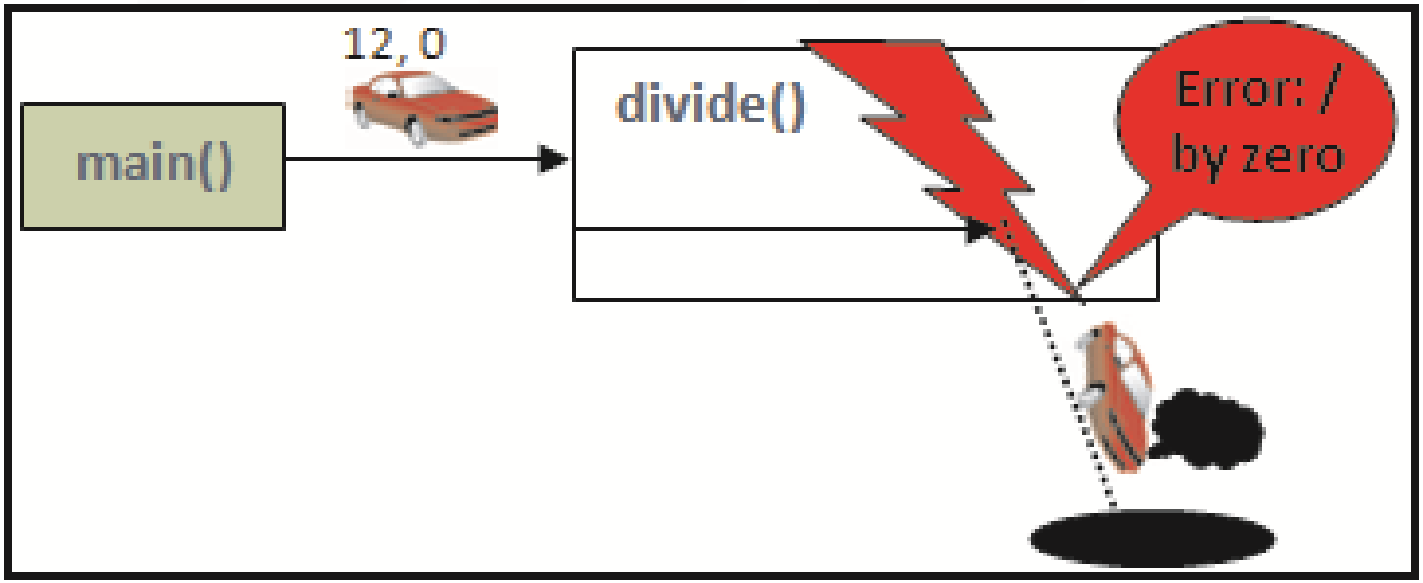
Khối ngoại lệ

```
/**
 * Define the TestMath.java class
 */
public class TestMath {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        // Check the number of command line arguments
        if(args.length==2) {
            // Instantiate the Mathematics class
            Mathematics objMath = new Mathematics();
            // Invoke the divide(int,int) method
            objMath.divide(Integer.parseInt(args[0]),
             Integer.parseInt(args[1]));
        }
        else {
            System.out.println("Usage: java Mathematics <number1> <number2>");
        }
    }
}
```

Khối ngoại lệ

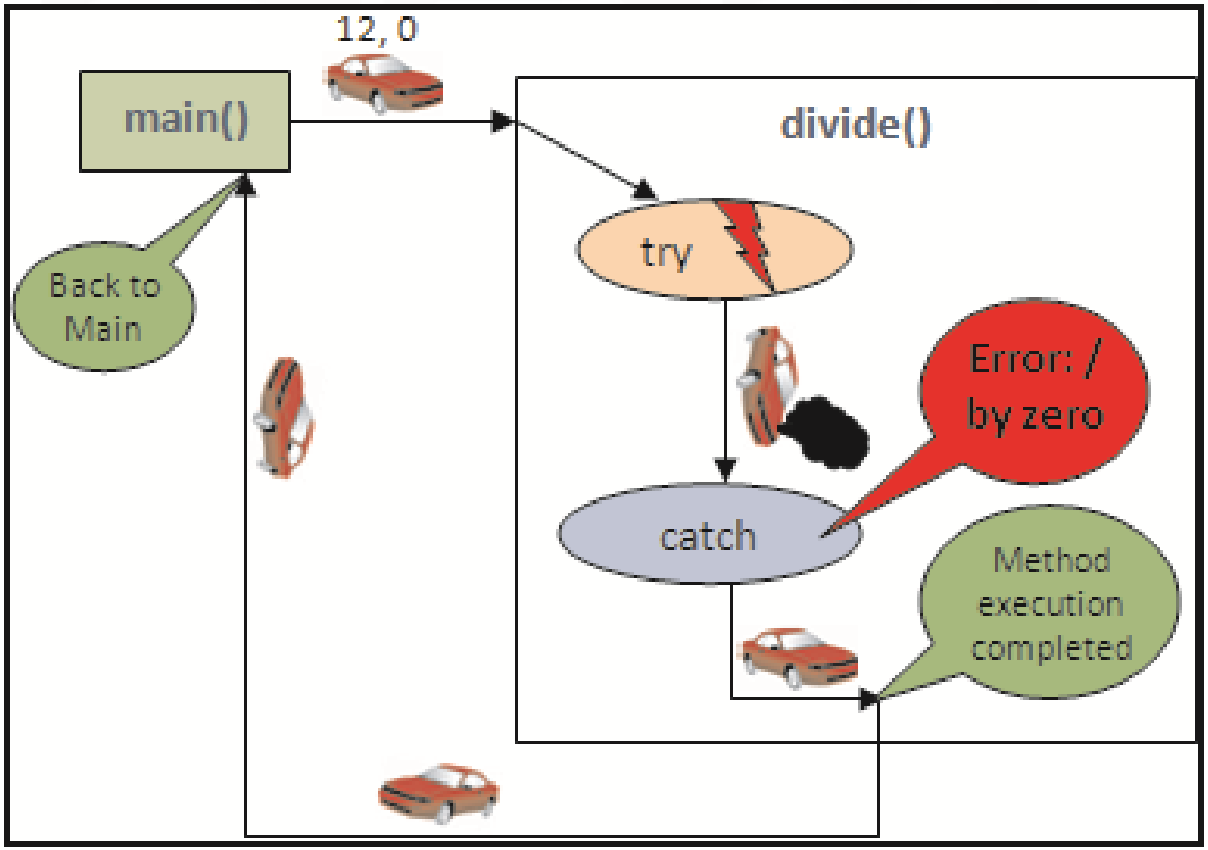
Kết quả:





Khối ngoại lệ

Xử lý các tình huống (dự đoán):



Khối ngoại lệ

Throws và **Throw**:

- Từ khóa **Throws**: cho biết một phương thức có thể ném ra ngoại lệ.
- Từ khóa **Throw**: ném ngoại lệ từ bên trong phương thức

```
package session12;
class Mathematics {

/**
 * Divides two integers, throws ArithmeticException
 * @param num1 an integer variable storing value of first number
 * @param num2 an integer variable storing value of second number
 * @return void
 */
public void divide(int num1, int num2) throws ArithmeticException {
```

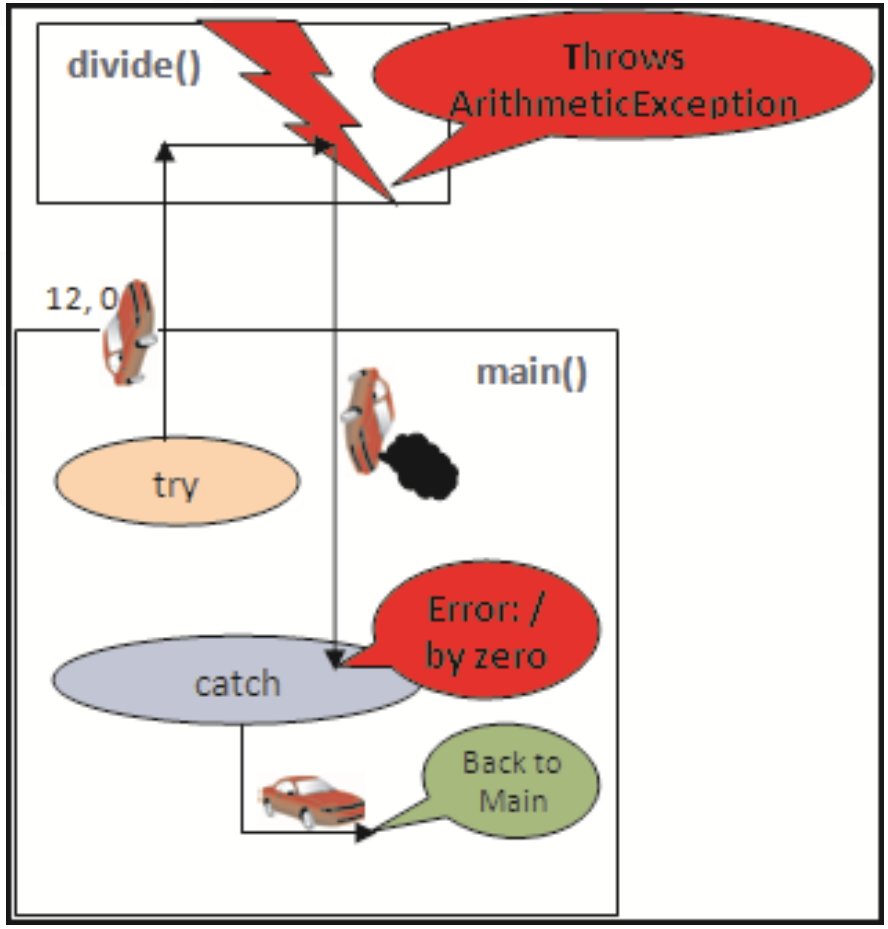
Khối ngoại lệ

```
    // Check the value of num2
    if(num2==0) {
        // Throw the exception
        throw new ArithmeticException("/ by zero");
    }
    else {
        System.out.println("Division is: " + (num1/num2));
    }

    // Rest of the method
    System.out.println("Method execution completed");
    }
}

/**
 * Define the TestMath.java class
 */
public class TestMath {
    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
```

Khối ngoại lệ

```
    // Check the number of command line arguments
    if(args.length==2) {
        // Instantiate the Mathematics class
        Mathematics objMath = new Mathematics();
        try {
            // Invoke the divide(int,int) method
            objMath.divide(Integer.parseInt(args[0]),
            Integer.parseInt(args[1]));
        }
        catch(ArithmeticException e) {
            // Display an error message to the user
            System.out.println("Error: "+ e.getMessage());
        }
    }
    else{
        System.out.println("Usage: java Mathematics <number1> <number2>");
    }
    System.out.println("Back to Main");
    }
}
```

Khối ngoại lệ



Khối ngoại lệ

Khối "catch" lồng:

Syntax

```
try
{…}
catch (<exception-type> <object-name>)
{…}
catch (<exception-type> <object-name>)
{…}
```

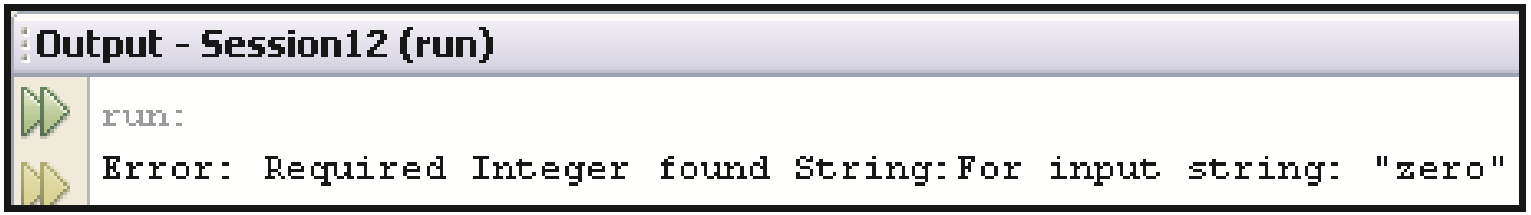
Khối ngoại lệ

- VD:

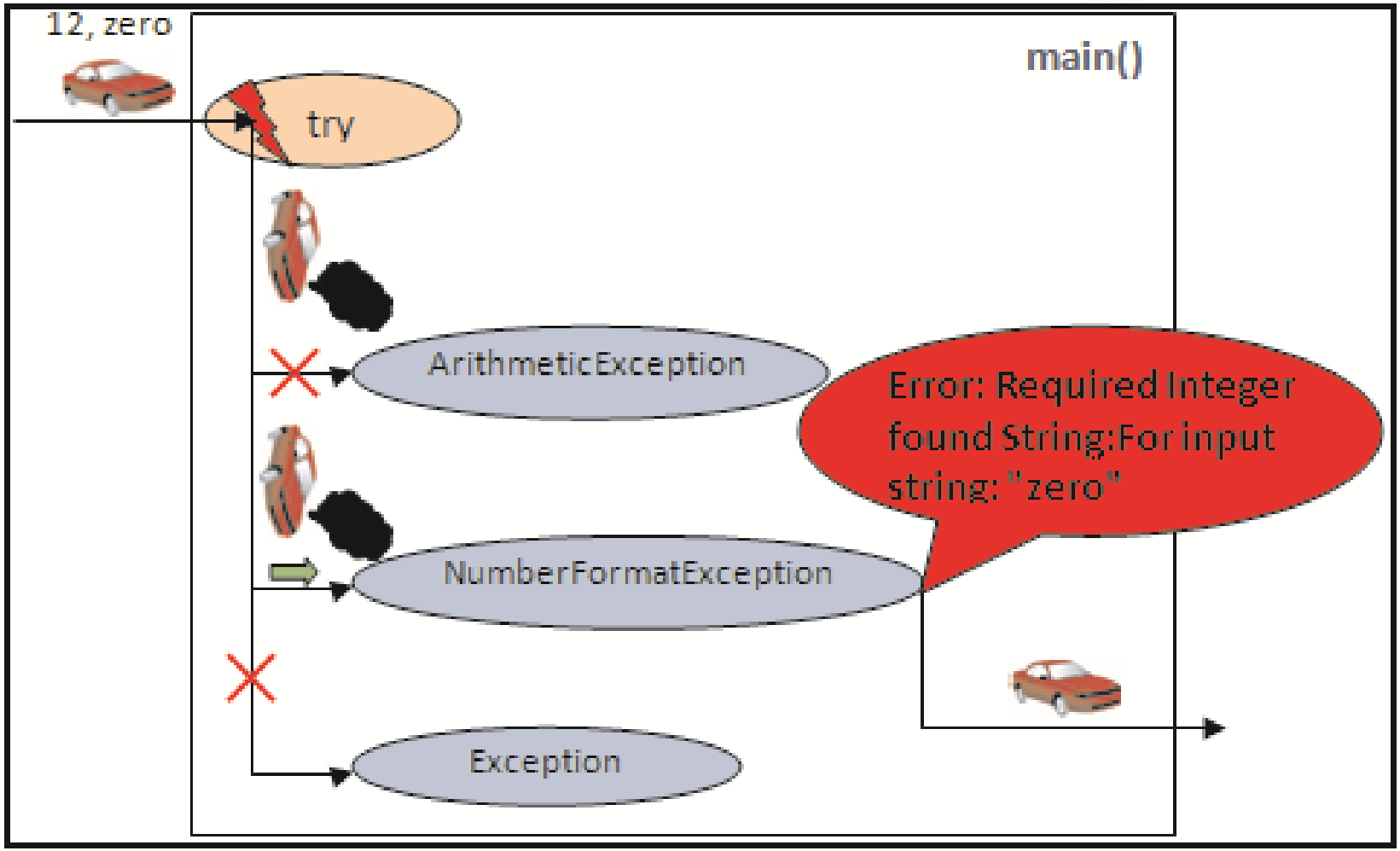
```
package session12;
public class Calculate {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Check the number of command line arguments
        if (args.length == 2){
            try {
                // Perform the division operation
                int num3 = Integer.parseInt(args[0]) / Integer.parseInt(args[1]);
                System.out.println("Division is: "+num3);
            }
            catch (ArithmeticException e) { // Catch the ArithmeticException
                System.out.println("Error: " + e.getMessage());
            }
            catch (NumberFormatException e){ // Catch the NumberFormatException
                System.out.println("Error: Required Integer found String:" +
                 e.getMessage());
            }
```

Khối ngoại lệ

```
            catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
            }
        }
        else {
            System.out.println("Usage: java Calculate <number1> <number2>");
        }
    }
}
```



Khối ngoại lệ



Khối ngoại lệ

DEVMASTER.VN

Khối **"finally"**:

- Java cung cấp khối finally để đảm bảo thực hiện khối lệnh cho dù có/không ngoại lệ xảy ra.
- Khối `finally` luôn được thực hiện cuối cùng trong toàn khối ngoại lệ.
- Điều này đảm bảo mã không bị bỏ qua bởi lệnh `return`, `break`, or `continue`.
- Khối `finally` được sử dụng chủ yếu như công cụ để ngăn chặn rò rỉ tài nguyên.

Khối ngoại lệ

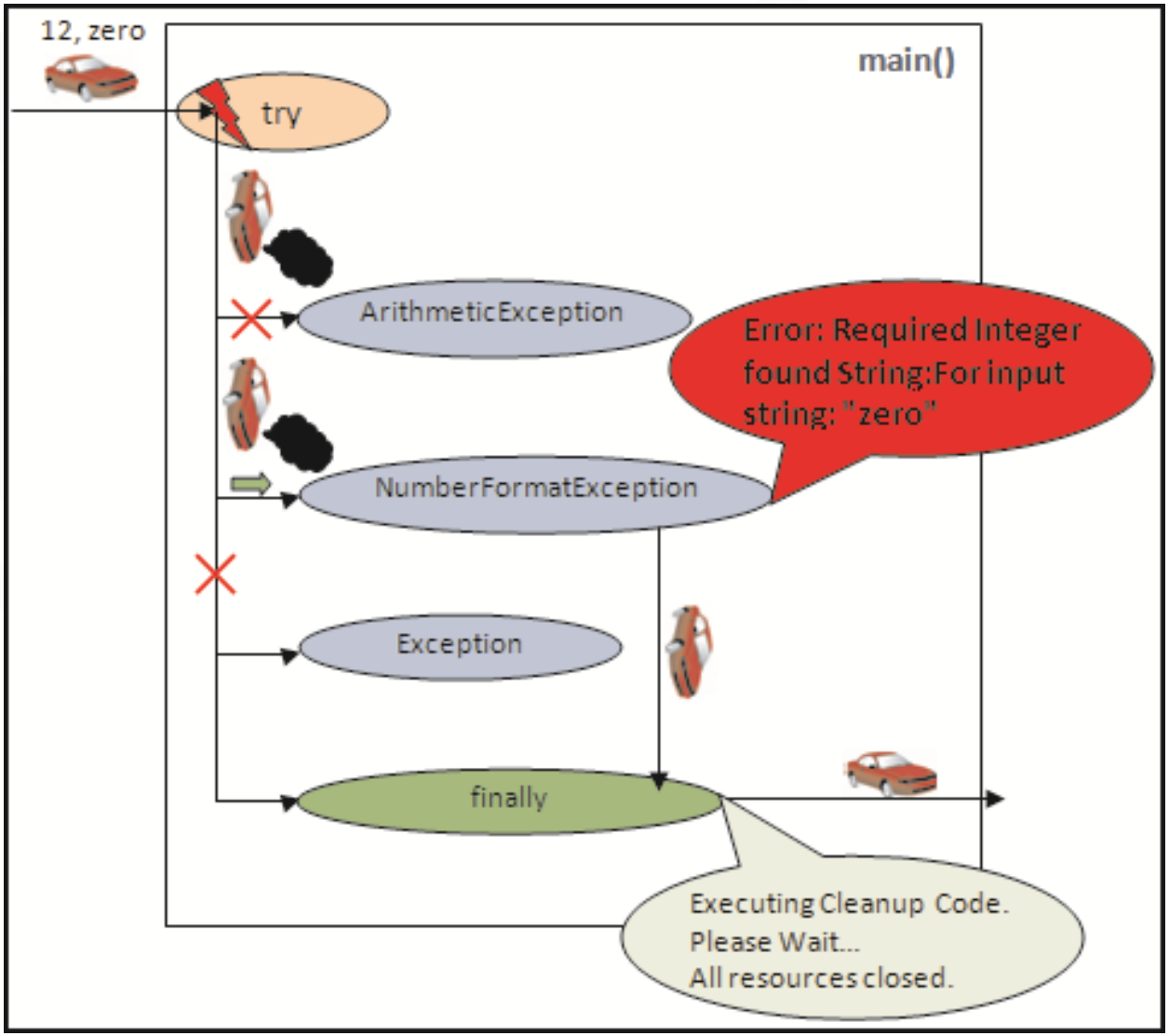
Khối **"finally"**:

Syntax

```
try
{
    // statements that may raise exception
    // statement 1
    // statement 2
}
catch(<exception-type> <object-name>)
{
    // handling exception
    // error message
}
finally
{
    // clean-up code
    // statement 1
    // statement 2
}
```

Khối ngoại lệ

Khối **"finally"**:



Tóm tắt bài học

- ✓ **Ngoại lệ** là sự kiện hoặc điều kiện bất thường trong chương trình diễn ra trong thời gian thực hiện dẫn tới làm gián đoạn dòng chảy bình thường của chương trình.
- ✓ Một phương thức có thể khai báo **xử lý** và **ném** ngoại lệ.
- ✓ Một chương trình mã nguồn tốt là phải **kiểm soát tối đa** tình huống có thể phát sinh ngoại lệ và xử lý nó.
- ✓ Khối lệnh **try{...}catch()** sử dụng để kiểm soát ngoại lệ trong mã lệnh.
- ✓ Có thể lồng nhiều khối **catch()**.
- ✓ Khối **finally** luôn đảm bảo thực hiện mã, ngay cả khi có/không có ngoại lệ xảy ra.

Thank for watching!