



NGÔN NGỮ LẬP TRÌNH JAVA

|| Nội dung

Cơ bản về ngôn ngữ lập trình Java

Lập trình hướng
đối tượng

Biến, từ khoá,
kiểu dữ liệu

Biểu thức, các
cấu trúc điều
khiển

Dữ liệu kiểu
mảng

Các khía cạnh nâng cao của lập trình hướng đối tượng

Thiết kế lớp

Thiết kế lớp
nâng cao

Xử lý ngoại lệ

Xây dựng ứng dụng Java

Tạo giao diện đồ
họa

Xử lý các sự
kiện trên giao
diện đồ họa

Xây dựng ứng
dụng đồ họa

Lập trình Java nâng cao

Luồng

Vào / Ra

Lập trình mạng

Lập trình với
CSDL



XỬ LÝ NGOẠI LỆ VÀ CƠ CHẾ DEBUG

|| Nội dung

1. Định nghĩa ngoại lệ - `exception`
2. Sử dụng câu lệnh `try - catch - finally`
3. Cơ chế xử lý ngoại lệ
4. Danh mục ngoại lệ, một số ngoại lệ hay gặp
5. Nắm bắt và xử lý ngoại lệ
6. Cách sử dụng cơ chế `Assertion` trong Java
7. Debugging

1. Ngoại lệ là gì?

- ❑ Đó là những tình huống bất thường xảy ra trong khi thực thi chương trình không phải do lỗi logic
- ❑ Các ngôn ngữ như C/C++ có nhược điểm khi phải xử lý ngoại lệ.

|| Ví dụ: một LTV cần mở một file để xử lý

1. LTV không quan tâm đến việc xử lý ngoại lệ: mở một file chưa tồn tại
2. LTV có quan tâm đến xử lý ngoại lệ.
 - Ưu tiên logic chính của chương trình, viết trình xử lý ngoại lệ “sau”
 - Tuy nhiên, việc xử lý ngoại lệ sau đó ko bao giờ được thực hiện
 - LTV ko thực hiện việc viết trình xử lý ngoại lệ cùng với logic chính
3. LTV quyết định viết trình xử lý ngoại lệ cùng với logic chính của chương trình
 - Trình xử lý ngoại lệ lẫn lộn với logic chính của chương trình
 - Khó theo dõi logic chính chương trình và toàn bộ chương trình rất khó đọc

Ví dụ : LTV quyết định viết trình xử lý ngoại lệ
cùng với logic chính của chương trình

```
if (file exists) {  
    open file;  
    while (there is more records to be processed) {  
        if (no IO errors) {  
            process the file record  
        } else {  
            handle the errors  
        }  
    }  
    if (file is opened) close the file;  
} else {  
    report the file does not exist;  
}
```

Java cung cấp cơ chế xử lý ngoại lệ

1. Việc xử lý ngoại lệ được khai báo cùng với khai báo phương thức

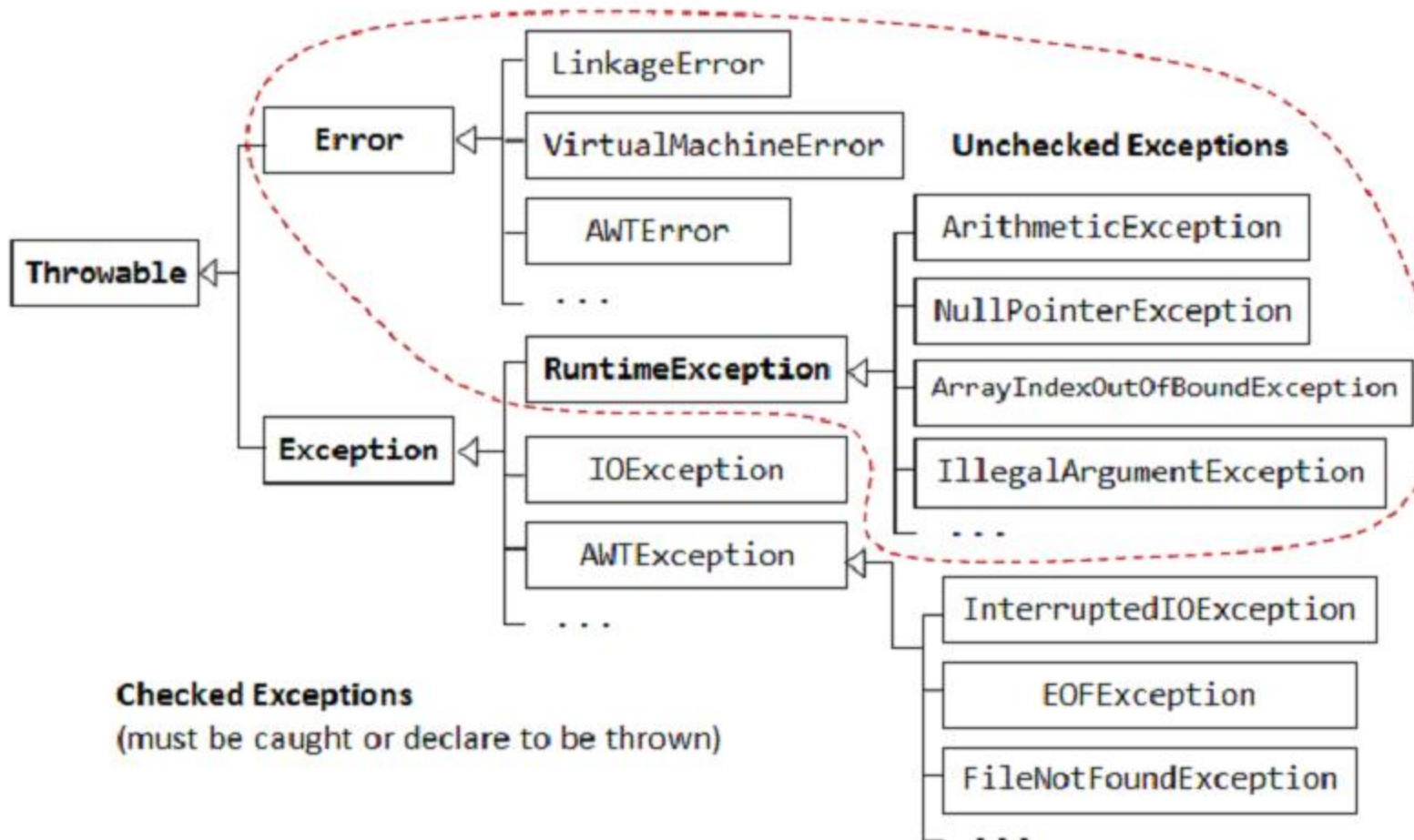
- ▣ LTV được thông báo về khả năng xuất hiện một ngoại lệ khi gọi một phương thức

2. Bắt buộc phải xử lý ngoại lệ khi viết logic chính của chương trình

- ▣ Chương trình ko thể biên dịch nếu ko xử lý ngoại lệ

3. Trình xử lý ngoại lệ được tách biệt với logic chính

- ▣ Thông qua cấu trúc try-catch-finally



Trong Java, ngoại lệ (exception) chia làm 2 loại

❑ unchecked exception :

- ❑ các sự cố nghiêm trọng xảy ra do lỗi tiềm ẩn của chương trình.
- ❑ được xử lý trong lớp `Error`, `RuntimeException`

❑ checked exception :

- ❑ các tình huống xảy ra do lỗi của người sử dụng, do môi trường... trong khi thực thi một chương trình đúng.
- ❑ được xử lý trong các lớp như `IOException`, `AWTException`

2. Câu lệnh try-catch-finally

```
try {  
    // main logic, uses methods that may throw Exceptions  
    ...  
} catch (Exception1 ex) {  
    // error handler for Exception1  
    ...  
} catch (Exception2 ex) {  
    // error handler for Exception1  
    ...  
    ...  
} finally { // finally is optional  
    // clean up codes, always executed regardless of exceptions  
    ...  
}
```

|| Ví dụ : try-catch-finally

```
public Scanner(File source) throws FileNotFoundException;
```

Để sử dụng một phương thức đã khai báo ngoại lệ :

1. Cung cấp trình xử lý ngoại lệ thông qua try-catch hoặc try-catch-finally
2. Không xử lý ngoại lệ, ném ngoại lệ lên phương thức cao hơn trong callstack

Ví dụ : try-catch-finally

```
public Scanner(File source) throws FileNotFoundException;
```

Cách 1 : xử lý ngoại lệ

```
1  import java.util.Scanner;
2  import java.io.File;
3  import java.io.FileNotFoundException;
4  public class TryCatchFinally {
5      public static void main(String[] args) {
6          try {          // main logic
7              System.out.println("Start of the main logic");
8              System.out.println("Try opening a file ...");
9              Scanner in = new Scanner(new File("test.in"));
10             System.out.println("File Found, processing the file ...");
11             System.out.println("End of the main logic");
12         } catch (FileNotFoundException e) {    // error handling separated from the main logic
13             System.out.println("File Not Found caught ...");
14         } finally {    // always run regardless of exception status
15             System.out.println("finally-block runs regardless of the state of exception");
16         }
17         // after the try-catch-finally
18         System.out.println("After try-catch-finally");
19     }
20 }
```

Ví dụ : try-catch-finally

```
public Scanner(File source) throws FileNotFoundException;
```

Cách 2 : ném ngoại lệ ra callstack

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class ScannerFromFileWithThrow {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner in = new Scanner(new File("test.in"));    // this method may
        // main logic here ...
    }
}
```

- Quyết định không xử lý ngoại lệ
- Ném ngoại lệ ra cho phương thức cao hơn trong call stack

III 3. Cơ chế xử lý ngoại lệ

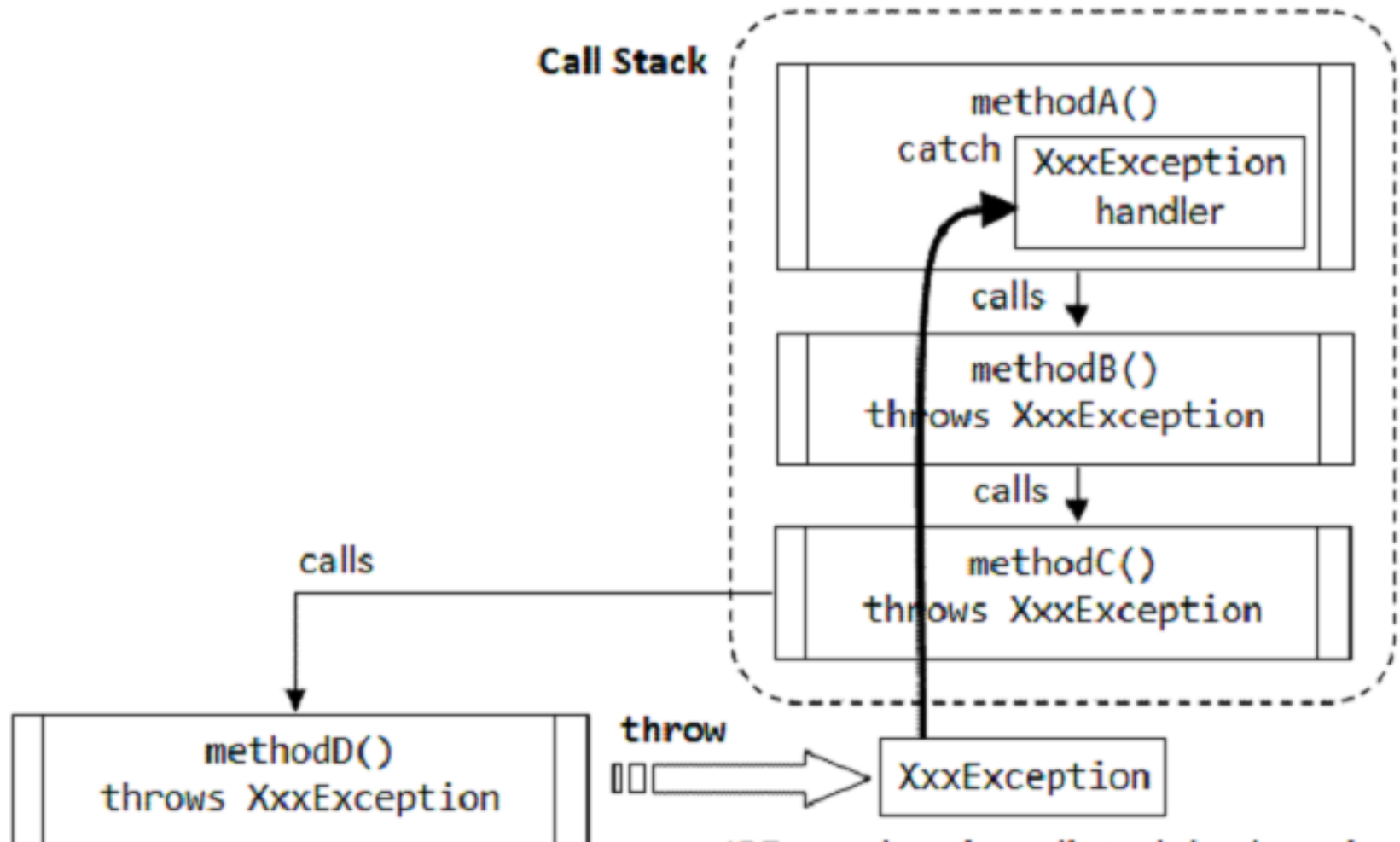
Khi có một ngoại lệ xảy ra trong một phương thức:

- ❑ Phương thức tạo ra 1 đối tượng ngoại lệ Exception và gửi nó tới JRE

- ❑ JRE tìm đoạn mã tương ứng để xử lý ngoại lệ trên

- ❑ JRE tìm ngược trong “call stack” cho đến khi nó tìm thấy trình xử lý ngoại lệ
 - ❑ Nếu ko tìm thấy trình xử lý ngoại lệ trong toàn bộ “call stack”, JRE sẽ kết thúc chương trình.

Call Stack



JRE searches the call stack *backward* for a *matching* exception handler

|| Ví dụ về Exception

```
1 public class AddArguments {
2     public static void main(String args[]) {
3         int sum = 0;
4         for ( String arg : args ) {
5             sum += Integer.parseInt(arg);
6         }
7         System.out.println("Sum = " + sum);
8     }
9 }
```

```
java AddArguments 1 2 3 4
Sum = 10
```

```
java AddArguments 1 two 3.0 4
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "two"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:447)
    at java.lang.Integer.parseInt(Integer.java:497)
    at AddArguments.main(AddArguments.java:5)
```


|| Ví dụ về Exception

```
1  public class AddArguments2 {
2      public static void main(String args[]) {
3          try {
4              int sum = 0;
5              for ( String arg : args ) {
6                  sum += Integer.parseInt(arg);
7              }
8              System.out.println("Sum = " + sum);
9          } catch (NumberFormatException nfe) {
10             System.err.println("One of the command-line "
11                               + "arguments is not an integer.");
12         }
13     }
14 }
```

java AddArguments2 1 two 3.0 4

One of the command-line arguments is not an integer.

|| Ví dụ về Exception

```
1  public class AddArguments3 {
2      public static void main(String args[]) {
3          int sum = 0;
4          for ( String arg : args ) {
5              try {
6                  sum += Integer.parseInt(arg);
7              } catch (NumberFormatException nfe) {
8                  System.err.println "[" + arg + "] is not an integer"
9                      + " and will not be included in the sum.");
10             }
11         }
12         System.out.println("Sum = " + sum);
13     }
14 }
```

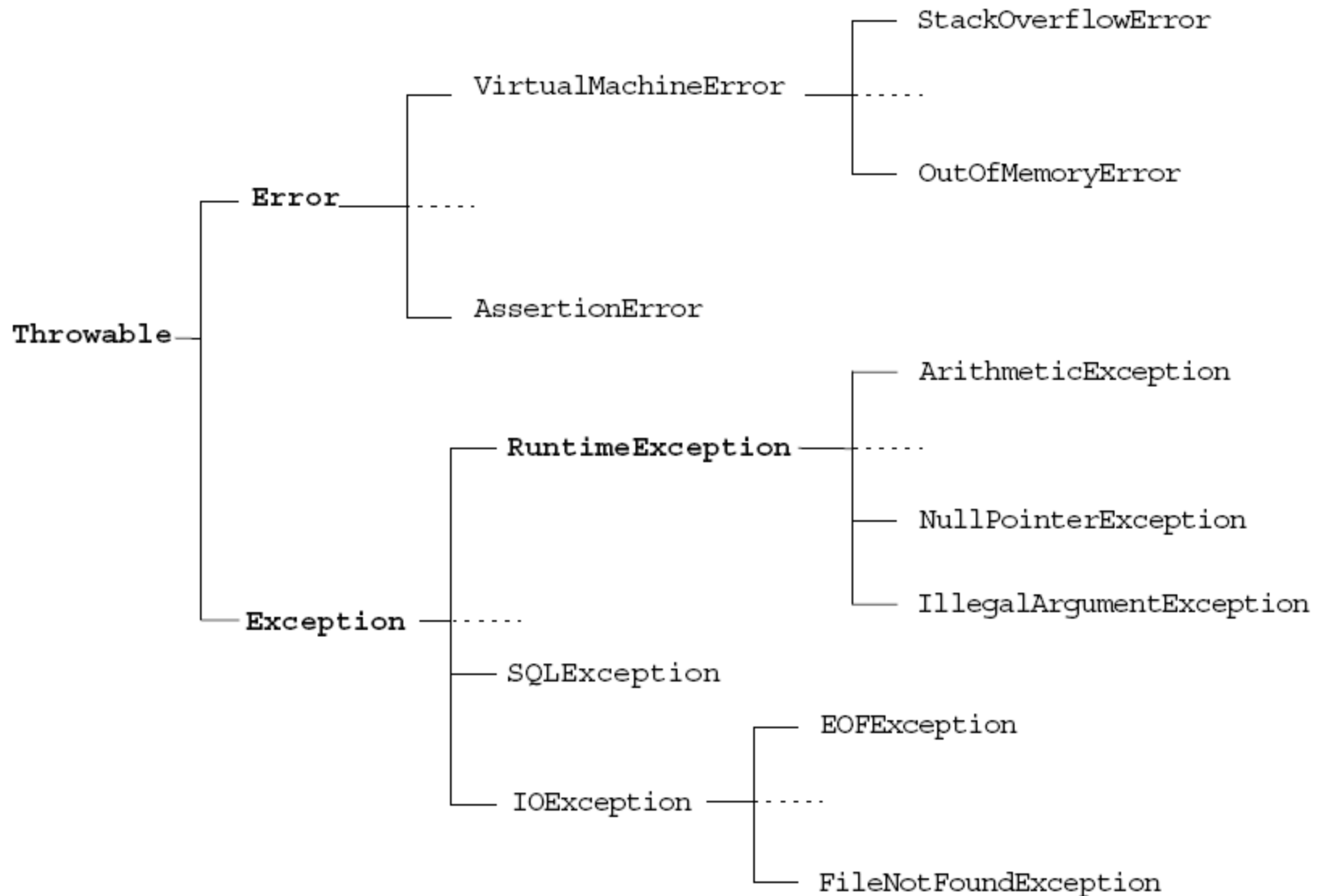
java AddArguments3 1 two 3.0 4

[two] is not an integer and will not be included in the sum.

[3.0] is not an integer and will not be included in the sum.

Sum = 5

4. Danh mục Exception



Unchecked RuntimeException

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.
UnsupportedOperationException	An unsupported operation was encountered.

Checked Exception

Exception	Description
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

|| Một số ngoại lệ phổ biến

- `NullPointerException`
- `FileNotFoundException`
- `NumberFormatException`
- `ArithmeticException`
- `SecurityException`



|| Nội dung

1. Định nghĩa ngoại lệ - `exception`
2. Sử dụng câu lệnh `try - catch - finally`
3. Cơ chế xử lý ngoại lệ
4. Danh mục ngoại lệ, một số ngoại lệ hay gặp
5. **Nắm bắt và xử lý ngoại lệ**
6. Cách sử dụng cơ chế Assertion trong Java
7. Debugging

5. Khai báo xử lý ngoại lệ cho phương thức

❑ Khi bên trong một phương thức có khả năng phát sinh ngoại lệ và không được xử lý, phương thức đó cần được khai báo xử lý ngoại lệ

❑ Từ khoá: throws (không phải throw)

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

❑ Ý nghĩa của việc khai báo xử lý ngoại lệ là thông báo cho phương thức khác sử dụng phương thức này sẽ xử lý ngoại lệ do phương thức này sinh ra.

|| Kỹ thuật đề hàm và khai báo exception

☐ Phương thức cài đề có thể:

- ☐ Không ném ra ngoại lệ
- ☐ Ném ra một hay nhiều ngoại lệ được khai báo ở phương thức bị cài đề
- ☐ Ngoại lệ con của ngoại lệ được khai báo ở phương thức bị cài đề

☐ Phương thức cài đề không thể:

- ☐ Ném ra thêm các ngoại lệ khác chưa được khai báo ở phương thức bị cài đề
- ☐ Ném ra ngoại lệ cha của ngoại lệ được khai báo ở phương thức bị cài đề

|| Kỹ thuật đề hàm và khai báo exception

```
1 public class TestA {  
2     public void methodA() throws IOException {  
3         // do some file manipulation  
4     }  
5 }
```

```
1 public class TestB1 extends TestA {  
2     public void methodA() throws EOFException {  
3         // do some file manipulation  
4     }  
5 }
```

```
1 public class TestB2 extends TestA {  
2     public void methodA() throws Exception { // WRONG  
3         // do some file manipulation  
4     }  
5 }
```

Tạo ngoại lệ mới

```
1  public class ServerTimeoutException extends Exception {  
2      private int port;  
3  
4      public ServerTimeoutException(String message, int port) {  
5          super(message);  
6          this.port = port;  
7      }  
8  
9      public int getPort() {  
10         return port;  
11     }  
12 }
```

Sử dụng ngoại lệ mới

[illegible]

|| Sử dụng ngoại lệ mới

```
1  public void findServer() {  
2      try {  
3          connectMe(defaultServer);  
4      } catch (ServerTimeoutException e) {  
5          System.out.println("Server timed out, trying alternative");  
6          try {  
7              connectMe(alternativeServer);  
8          } catch (ServerTimeoutException e1) {  
9              System.out.println("Error: " + e1.getMessage() +  
10                  " connecting to port " + e1.getPort());  
11          }  
12      }  
13  }
```

6. Cơ chế kiểm tra bằng assertion

❑ Cho phép phát hiện lỗi sớm bằng cách kiểm tra pre-condition, post-condition, và các bất biến của lớp (class invariants).

❑ Cú pháp

```
assert <boolean_expression> ;  
assert <boolean_expression> : <detail_expression> ;
```

❑ Nếu biểu thức <boolean_expression> trả về false, AssertionError sẽ được nhả ra

❑ Nội dung của detail_expression sẽ được chuyển sang dạng xâu và in ra màn hình.

Ví dụ

❑ Tình huống:

```
1  if (x > 0) {  
2    // do this  
3  } else {  
4    // do that  
5  }
```

❑ Giải pháp:

```
1  if (x > 0) {  
2    // do this  
3  } else {  
4    assert ( x == 0 );  
5    // do that, unless x is negative  
6  }
```

Ví dụ

```
1  switch (suit) {
2      case Suit.CLUBS: // ...
3          break;
4      case Suit.DIAMONDS: // ...
5          break;
6      case Suit.HEARTS: // ...
7          break;
8      case Suit.SPADES: // ...
9          break;
10     default: assert false : "Unknown playing card suit";
11         break;
12 }
```


|| Kích hoạt chế độ debug

❑ Ở chế độ mặc định, cơ chế debug không được kích hoạt. Để kích hoạt nó cho từng chương trình, ta sử dụng cú pháp sau:

```
java -enableassertions MyProgram
```

or:

```
java -ea MyProgram
```

7. Debugging

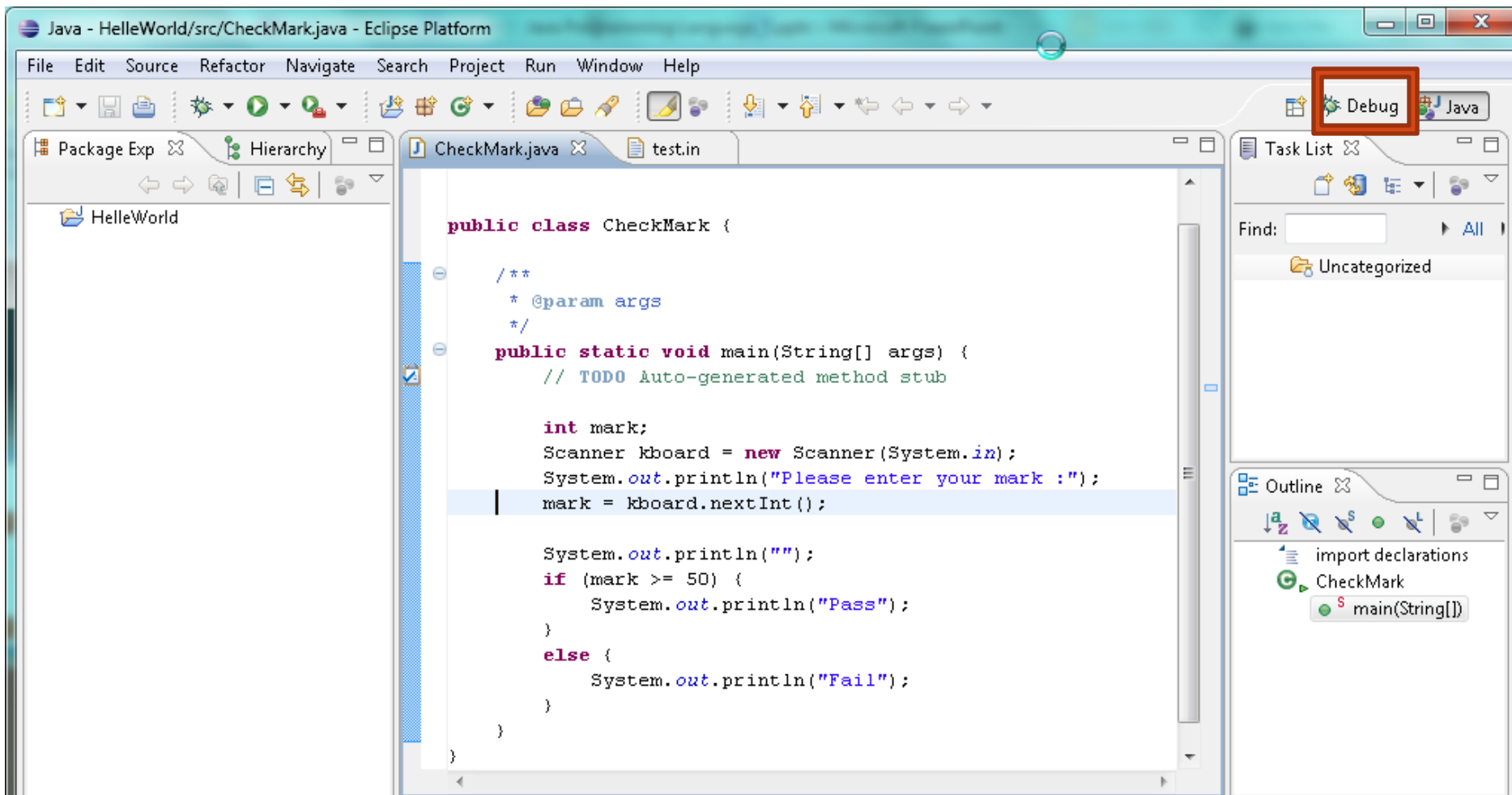
❑ Debugging (chế độ gỡ lỗi) là gì?

- ❑ Debugging cho phép chạy ứng dụng ở chế độ tương tác, đồng thời cho phép xem mã nguồn và các biến trong khi thực hiện
- ❑ Sử dụng các “stop points” để thiết lập điểm dừng trong khi chạy ứng dụng
 - ❑ Breakpoints (điểm ngắt) : thiết lập một điểm trong mã nguồn xác định nơi sự thực thi chương trình nên dừng lại
 - ❑ Watchpoints (điểm quan sát) : là một breakpoint + dừng chương trình chỉ khi một trường được đọc hay bị thay đổi

❑ Debugging trong Eclipse:

- ❑ Debug perspective

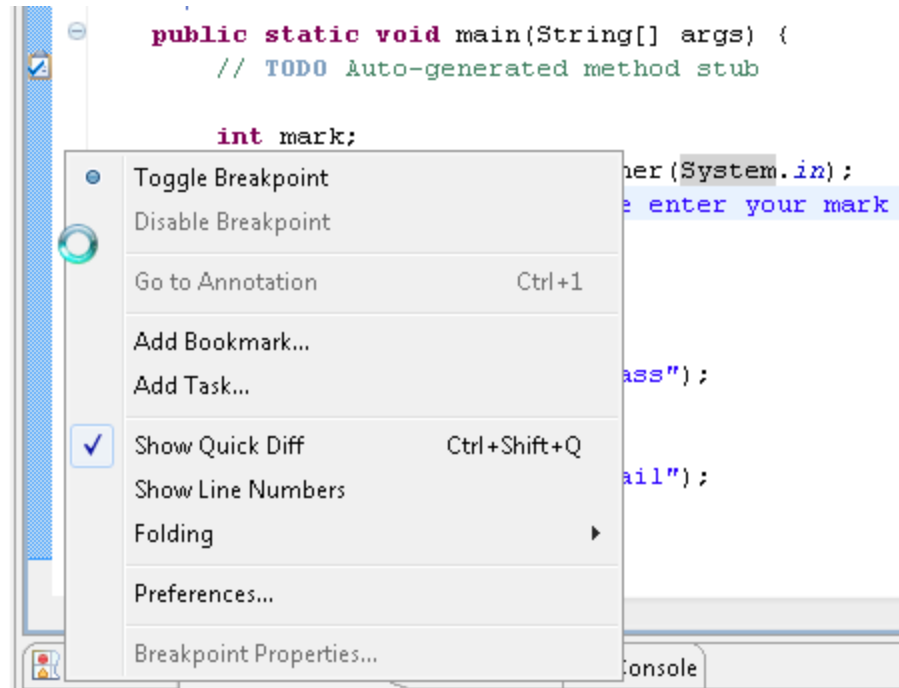
Debug perspective



Window / Open Perspective / Debug

Thiết lập breakpoints

- ❑ Chuột trái vào lề trái của trình soạn thảo code
- ❑ Toggle breakpoints



- ❑ Chạy ứng dụng ở chế độ debug
 - ❑ 1. Chuột trái vào tên file
 - ❑ Chọn Debug as -> Java Application
 - ❑ 2. Debug perspective

Debug view/perspective

Debug - HelloWorld/src/CheckMark.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Debug

CheckMark [Java Application]

- CheckMark at localhost:60423
 - Thread [main] (Suspended (breakpoint at line 13 in CheckMark))
 - CheckMark.main(String[]) line: 13

C:\Program Files (x86)\Java\jre7\bin\javaw.exe (24 févr. 14 03:14:20)

Current stack

Variables

Name	Value
args	String[0] (id=16)

CheckMark.java test.in

```
/**
 * @param args
 */
public static void main(String[] args) {
    // TODO Auto-generated method stub

    int mark;
    Scanner kboard = new Scanner(System.in);
    System.out.println("Please enter your mark :");
    mark = kboard.nextInt();
}
```

Outline

- import declarations
- CheckMark
 - main(String[])

Console

CheckMark [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (24 févr. 14 03:14:20)

II Điều khiển sự thực thi chương trình



F8 : resume

Stop

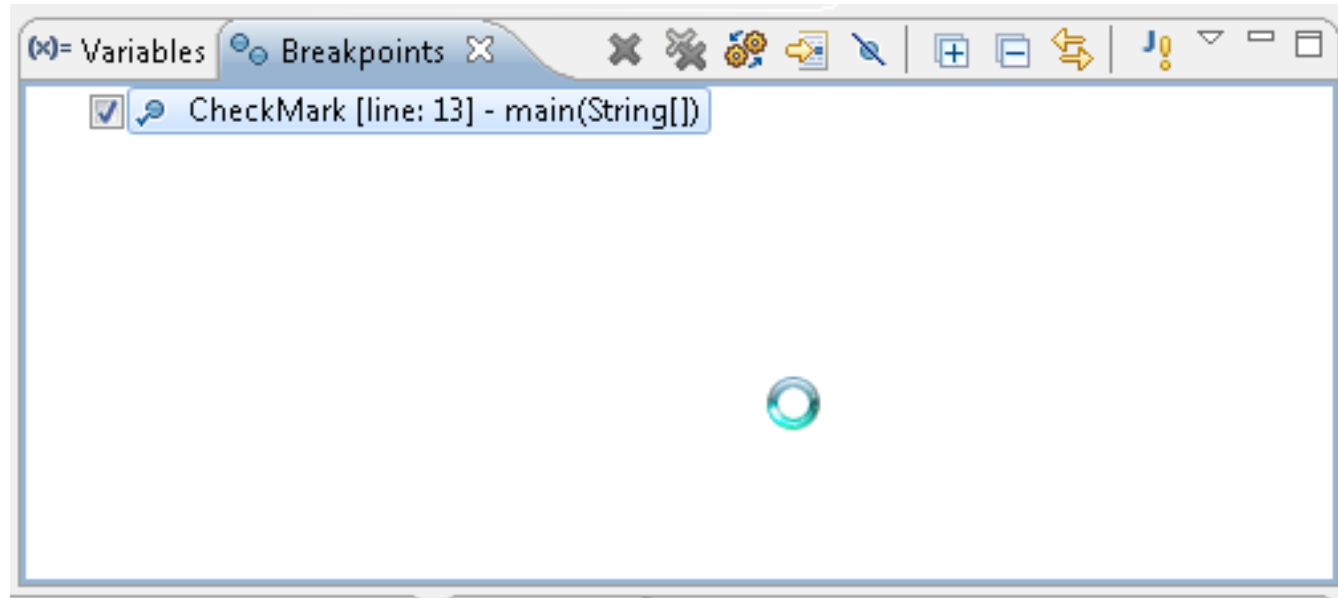
F5 - step into : nhảy vào trong mã thực thi của 1 phương thức





F6 - step over : thực thi 1 phương thức và trả về kết quả

F7 - step return : thoát ra và trở về phương thức cha gọi đến phương thức hiện tại

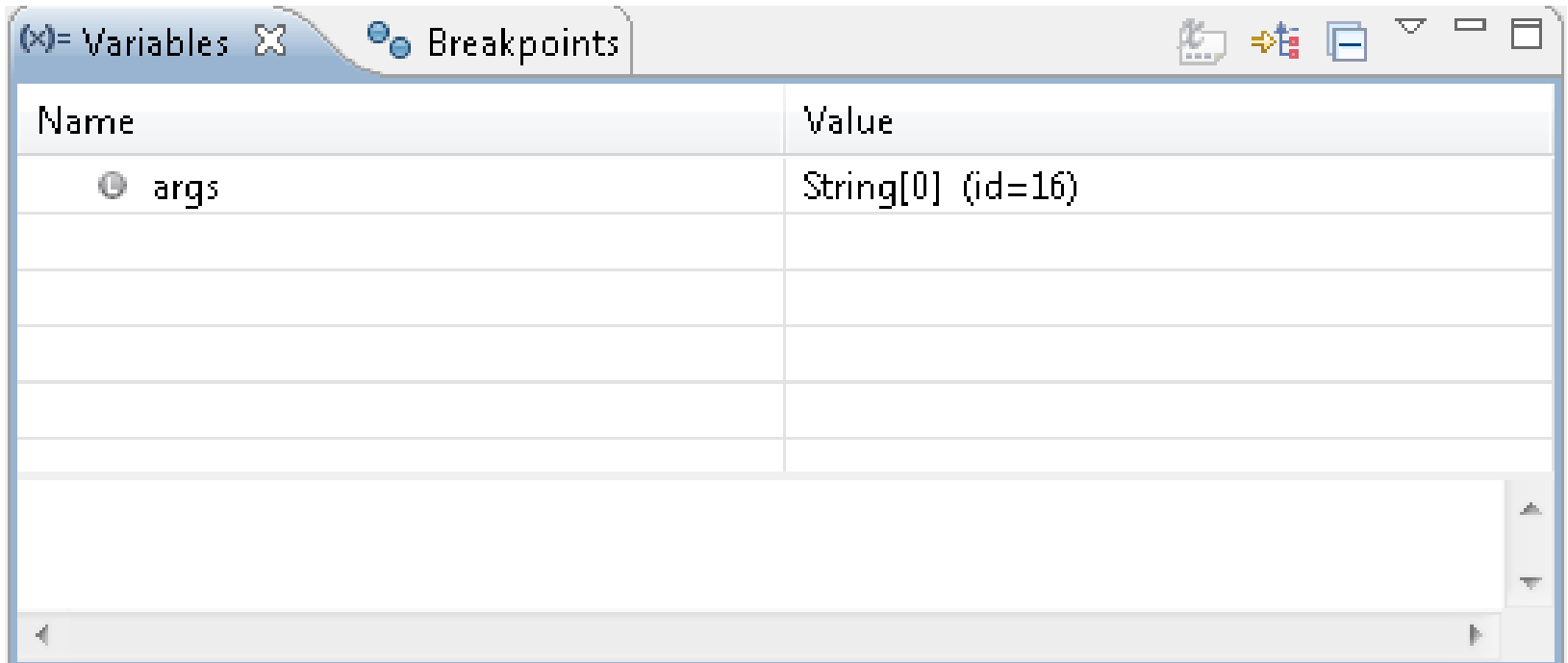
Ctrl + R : chạy tới dòng lệnh hiện tại (có con trỏ nhấp nháy)

Breakpoints view / Xoá breakpoints



-  Xoá breakpoints đang được lựa chọn
-  Xoá tất cả breakpoints
-  Bỏ kích hoạt tất cả các breakpoints
-  Bỏ kích hoạt 1 breakpoint

Variable view



Hiển thị thông tin về các trường và biến nội bộ

Thay đổi giá trị các biến trong khi thực thi chương trình

Variable view

The screenshot shows a debugger interface with two tabs: 'Variables' (selected) and 'Breakpoints'. The 'Variables' tab displays a table with two columns: 'Name' and 'Value'. The table contains two rows: 'args' with value 'String[0] (id=15)' and 'sum' with value '40'. The '40' is highlighted with a red box. A red arrow points from the text 'Change value here' to the '40'.

Name	Value
args	String[0] (id=15)
sum	40

Change value here

40

Thay đổi giá trị các biến trong khi thực thi chương trình

Example 1 : Counter

```
public class Main {  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        Counter counter = new Counter();  
        counter.count();  
        System.out.println("We have counted " + counter.getResult());  
    }  
}
```

```
public class Counter {  
    private int result = 0;  
    public int getResult() {  
        return result;  
    }  
  
    public void count() {  
        for (int i=0; i<100; i++) {  
            result += i+1;  
        }  
    }  
}
```

Câu hỏi:
i = 49 => result = ?

Example 2 : Fibonacci

Dãy Fibonacci là dãy vô hạn các số tự nhiên bắt đầu bằng hai phần tử 1 và 1, các phần tử sau đó được thiết lập theo quy tắc mỗi phần tử luôn bằng tổng hai phần tử trước nó. Công thức truy hồi của dãy Fibonacci là:

$$F(n) := \begin{cases} 1, & \text{khi } n = 1; \\ 1, & \text{khi } n = 2; \\ F(n-1) + F(n-2) & \text{khi } n > 2. \end{cases}$$

Example 2 : Fibonacci

```
import java.util.Scanner;
public class Fibonacci {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner scanner = new Scanner(System.in);
        int n = 0;
        do {
            System.out.print("Enter a number (n>2): ");
            n = scanner.nextInt();
        } while (n < 2);

        printArray(getFiboArray(n));
    }

    private static int[] getFiboArray(int n) {
        int[] f = new int[n];
        f[0] = 1;
        f[1] = 1;
        for (int i = 2; i < n; i++)
            f[i] = f[i - 1] + f[i - 2];
        return f;
    }

    private static void printArray(int[] fiboArray) {
        for (int i = 0; i < fiboArray.length; i++) {
            System.out.printf("%4d", fiboArray[i]);
        }
    }
}
```

1. Set breakpoint tại 2 điểm
printArray(getFiboArray(n));
f[i] = ...

2. Chạy chương trình
Nhập n = 19
f[i=13] = ?

3. Trong quá trình debug, thay
đổi f[13] = 1000
f[19] = ?

Example 3 : Fibonacci Recursion

```
import java.util.Scanner;
public class FibonacciRecursion {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Enter number upto which Fibonacci series to print: ");
        int number = new Scanner(System.in).nextInt();

        System.out.println("Fibonacci series upto " + number + " numbers : ");

        for(int i=1; i<=number; i++){
            System.out.print(fibonacci(i) + " ");
        }

        public static int fibonacci(int number){
            if(number == 1 || number == 2){
                return 1;
            }

            return fibonacci(number-1) + fibonacci(number-2); //tail recursion
        }
    }
}
```

Với $n = 97$, độ sâu của call stack là bao nhiêu?