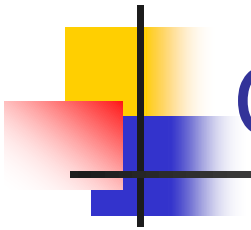
# CÔNG NGHỆ JAVA

Trường: Đại học Giao thông vận Tải

Khoa: Công nghệ thông tin



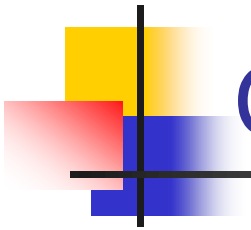
# CÔNG NGHỆ JAVA

Chương 4:

## Kiểm soát lỗi trong Java

# Nội dung

- Nội dung
  - Các khái niệm về kiểm soát lỗi
    - Khái niệm lỗi
    - Phân loại lỗi trong Java
    - Các cơ chế kiểm soát lỗi
  - Cơ chế kiểm soát lỗi
    - Kiểm soát lỗi với try … catch … finally
    - Tạo và lan truyền lỗi với throw … new và throws
    - Định nghĩa exceptions
    - …
  - Quản lý và giải phóng bộ nhớ trong Java
    - Quản lý bộ nhớ với heap và stack
    - Cơ chế giải phóng bộ nhớ

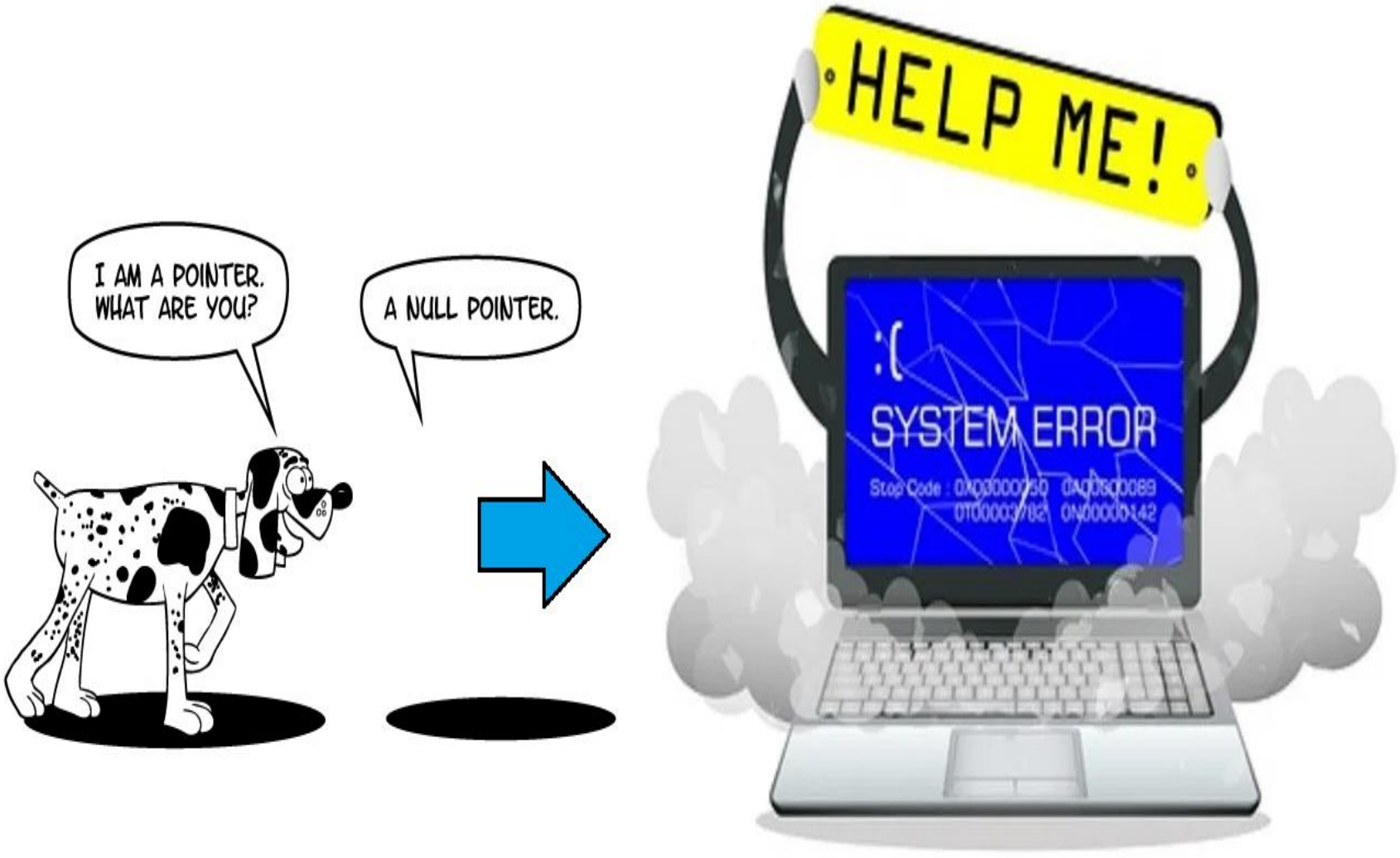


# CÔNG NGHỆ JAVA

Chương 4:

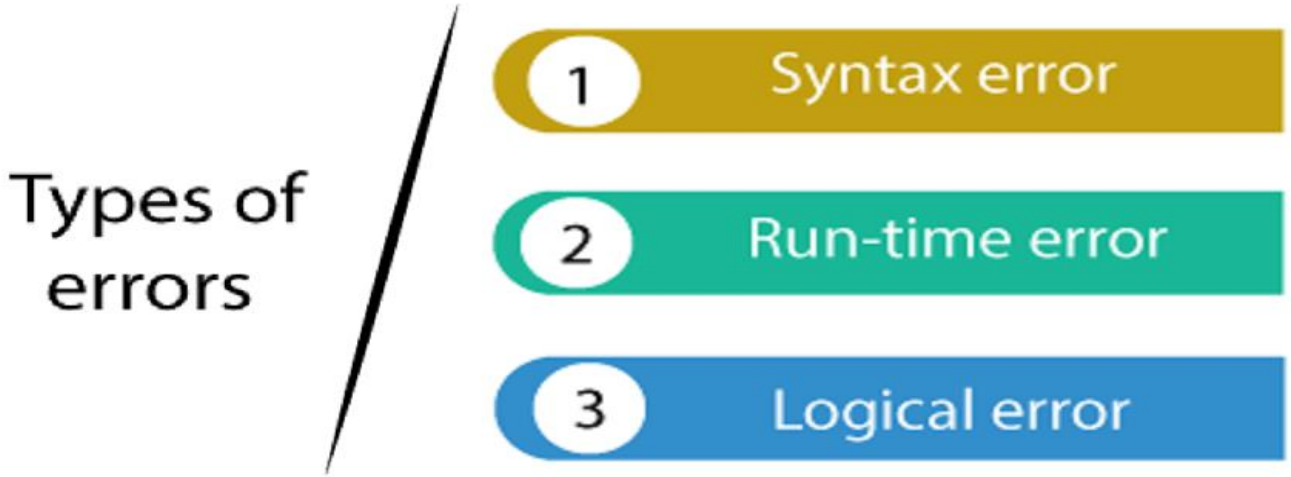
## 1. Lỗi và các cơ chế kiểm soát

# Khái niệm về lỗi



# Khái niệm về lỗi - exception

- Lỗi là vấn đề xảy ra trong quá trình phát triển và thực thi chương trình bao gồm:
  - **Compile error** (lỗi biên dịch - syntax error): lỗi xảy ra trong quá trình viết chương trình.
  - **Run-time error** (lỗi thực thi): lỗi xảy ra trong quá trình thực thi chương trình.
  - **Logical error** (lỗi logic): lỗi xảy ra trong quá trình viết chương trình dẫn đến khi thực thi kết quả không như yêu cầu.



# Khái niệm về lỗi - exception

- Với các lỗi biên dịch – **compile error**, đây là lỗi do các lập trình viên gây ra với một số nguyên nhân xảy ra lỗi:
  - Viết sai các cú pháp lệnh, sai câu lệnh, thiếu thư viện, …
  - Truy cập vào các phương thức trái với phạm vi cho phép.
  - Chưa kiểm soát các lỗi được lan truyền khi sử dụng hàm con.
  - …
  - ➔ Các lỗi này được trình soạn thảo IDE hỗ trợ kiểm soát và được thông báo trước khi chạy chương trình.
  - ➔ Các lỗi này có thể định nghĩa thông qua định nghĩa compiler của trình soạn thảo.
  - ➔ ***Chú ý:*** khi chúng ta thay đổi các định nghĩa lỗi biên dịch thông qua compiler sẽ dẫn đến tạo thành lỗi khi thực thi chương trình.

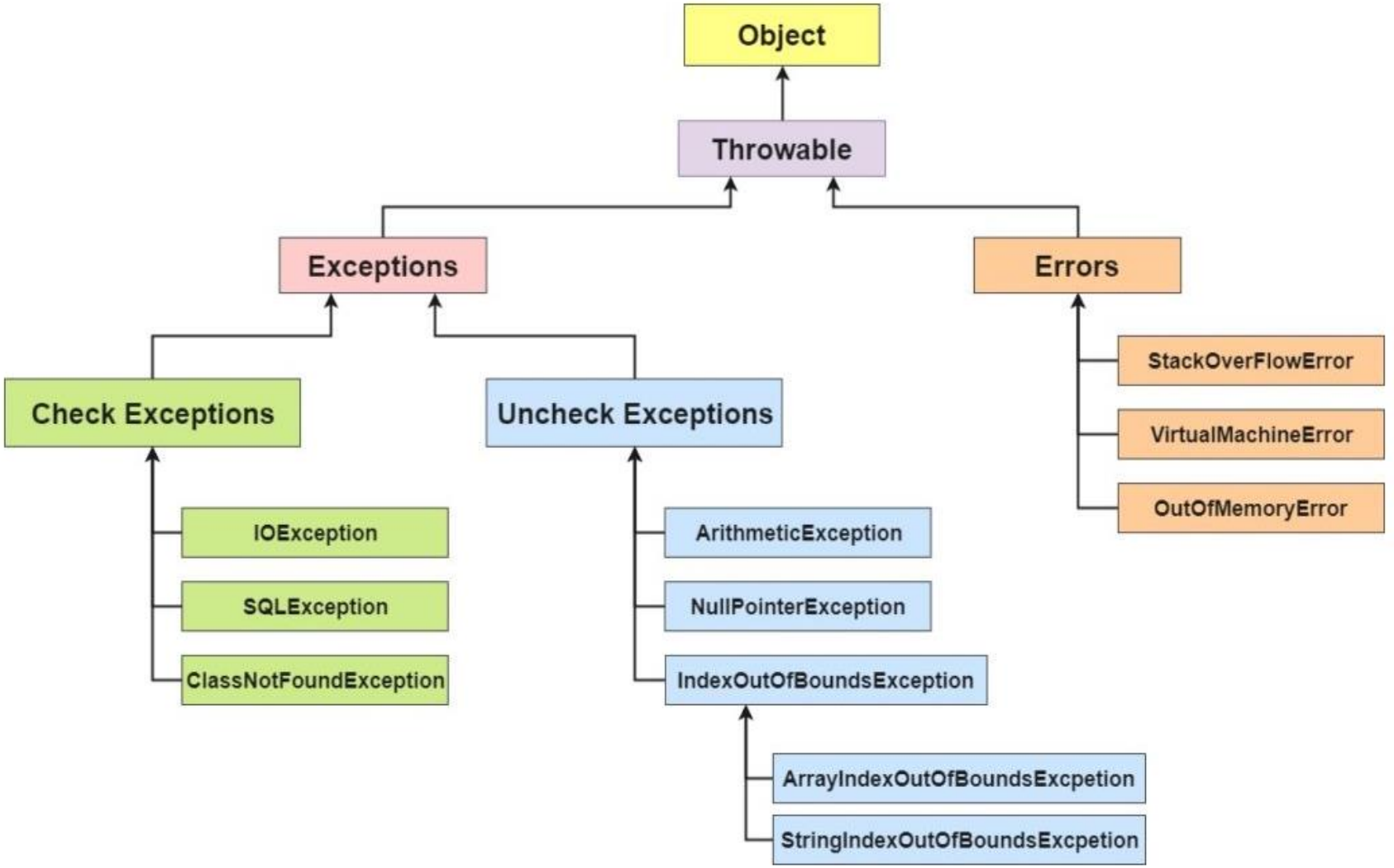
# Khái niệm về lỗi - exception

- Với các lỗi logic – **logical error**, đây là lỗi do nhóm phát triển gây ra với một số nguyên nhân xảy ra lỗi:
  - Khảo sát và phân tích yêu cầu không đúng như mong muốn của người dùng ➔ đưa ra nghiệp vụ sai.
  - Khi phát triển, lập trình viên đã thực hiện sai yêu cầu như nhầm phép tính, gọi hàm sai yêu cầu ➔ kết quả sai.
  - ➔ Các lỗi này không thể kiểm soát được bởi các trình soạn thảo cũng như các công cụ bắt lỗi khi dịch hoặc chạy chương trình.
  - ➔ Các lỗi này chỉ có thể định xác định khi thực hiện kiểm thử ➔ khó xác định được nguyên nhân để sửa lỗi.
  - ➔ ***Chú ý:*** phát triển phần mềm bao giờ cũng tiềm ẩn các lỗi logic ngay từ khi bắt đầu triển khai ➔ kiểm thử là việc bắt buộc và chiếm 30% khối lượng công việc.

# Khái niệm về lỗi - exception

- Với các lỗi thực thi – **run-time error**, đây là lỗi do cả con người và hệ thống có thể gây ra với một số nguyên nhân xảy ra lỗi:
  - Phần cứng hoặc các thiết bị có lỗi dẫn đến chương trình xảy ra lỗi khi đang chạy ➔ **Error**.
    - Các lỗi này không thể kiểm soát được bởi các trình soạn thảo cũng như các lập trình viên.
  - Khi phát triển, lập trình viên đã không kiểm soát các trường hợp như file không tồn tại, không có kết nối với CSDL, chia cho 0, … ➔ **Exception**.
    - Một số lỗi dạng này không thể kiểm soát được bởi các trình soạn thảo cũng như các công cụ bắt lỗi khi dịch chương trình mà chỉ có thể định xác định khi thực hiện chương trình nhưng chúng ta có thể kiểm soát được tất cả các lỗi này bằng chương trình.
    - ➔ *Lập trình viên cần phải kiểm soát các lỗi exception.*

# Phân loại lỗi trong Java



# Error vs Exception

| Error | Exception |
|---|---|
| Lỗi gây ra nguyên do từ hệ thống. | Lỗi gây ra từ các đoạn mã chương trình. |
| Lỗi không thể khắc phục và không kiểm soát được. | Lỗi cần phải kiểm soát bằng chương trình. |
| Khi xảy ra lỗi, chương trình sẽ kết thúc bất thường. | Khi xảy ra lỗi, chương trình có thể gây ra kết quả sai, có thể gây kết thúc chương trình bất thường. |
| StackOverFlow, OutOfMemory, … | ClassNotFound, FileNotFound, IndexOutOfBound … |

```
public class ErrorExample {

    public static void main(String[] args) {
        double count = 1.0;
        while(count>0){
            count ++;
        }
        System.out.println("Example: " + count);
    }
}
```

```
public class ExceptionExample {

    public static void main(String[] args) {
        int[] i_lists = { 1, 5, 8, 19 };
        int total = i_lists[0] + i_lists[4];
        System.out.println("Total: " + total);
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
        at week_08.ExceptionExample.main(ExceptionExample.java:7)
```

# Check vs Uncheck

| Checked exception | Unchecked exception |
|---|---|
| Lỗi gây ra từ các đoạn mã chương trình. | Lỗi gây ra từ các đoạn mã chương trình. |
| Các lỗi này mặc định có thể dự báo từ trước. | Các lỗi này có thể không xảy ra, phụ thuộc code của lập trình viên. |
| Lỗi được kiểm soát ngay tại trình biên dịch (compile-time exception) nguyên nhân là do các ngoại lệ được lan truyền từ phương thức được gọi bên trong ➔ ngoại lệ **yêu cầu** phải kiểm soát [xử lý hoặc lan truyền tiếp]. | Lỗi không thể kiểm soát bởi trình biên dịch mà chỉ xảy ra trong quá trình thực thi (run-time exception) do sai về bởi các đoạn mã chương trình ➔ ngoại lệ bắt buộc phải kiểm soát. |
| FileNotFoundException, ClassNotFoundException, … | ArithmeticException, ArrayIndexOutOfBoundsException … |

```
public class UncheckExceptionExample {

    public static void main(String[] args) {
        File f = new File("C:\\test.txt");
        FileReader fr = new FileReader(f);
        System.out.println(fr.ready());
    }
}
```

Unhandled exception type FileNotFoundException

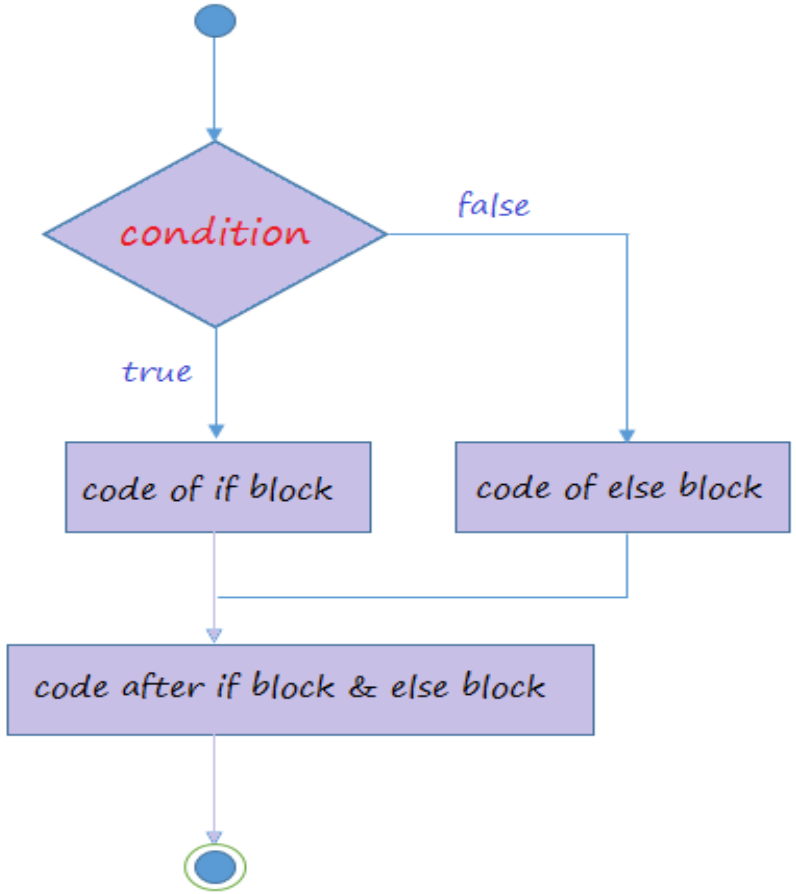
2 quick fixes available:

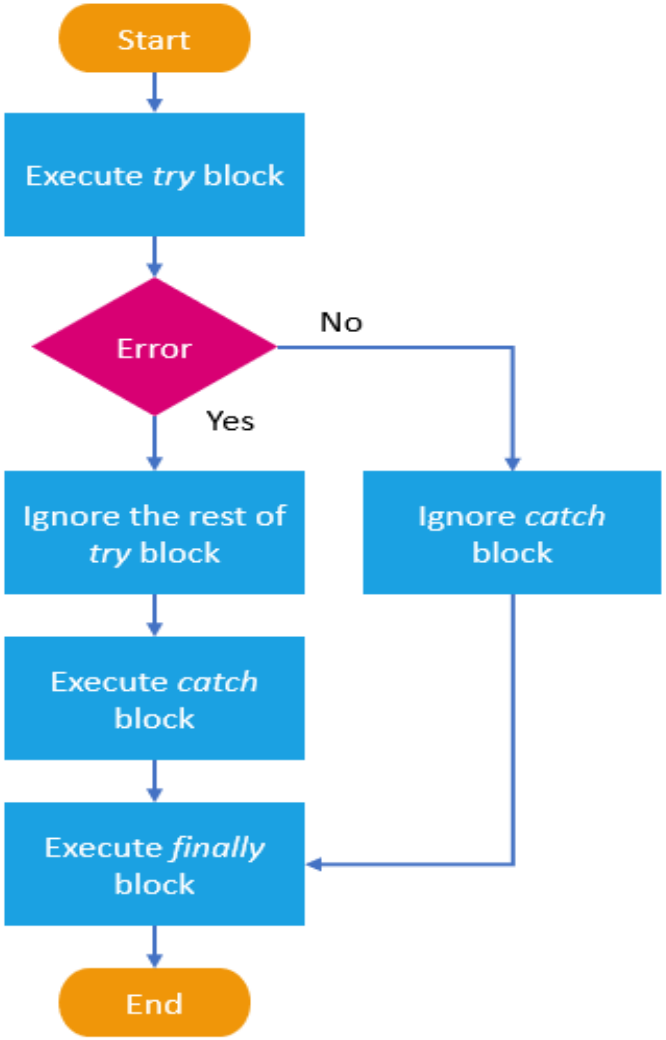
Add throws declaration

Surround with try/catch

```
public class CheckExceptionExample {

    public static void main(String[] args) {
        int[] i_lists = { 1, 5, 8, 19 };
        int total = i_lists[0] + i_lists[4];
        System.out.println("Total: " + total);
    }
}
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
        at week_08.ExceptionExample.main(ExceptionExample.java:7)
```

# Cơ chế kiểm soát lỗi



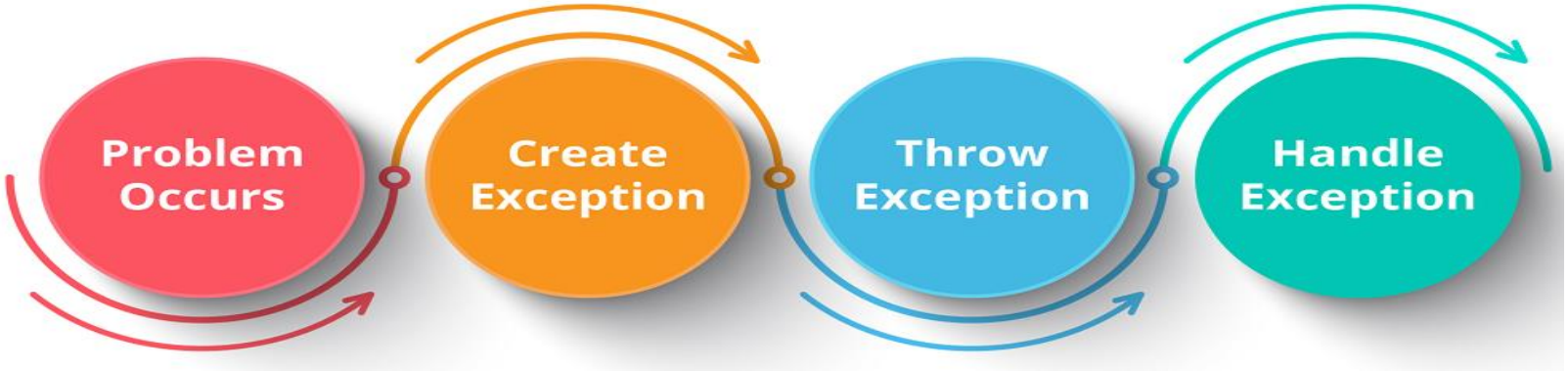



Chương 4:

## 2. Cơ chế kiểm soát lỗi trong Java

# Cơ chế kiểm soát lỗi

- Lỗi cần phải được kiểm soát trong chương trình. Cơ chế chung của kiểm soát lỗi:

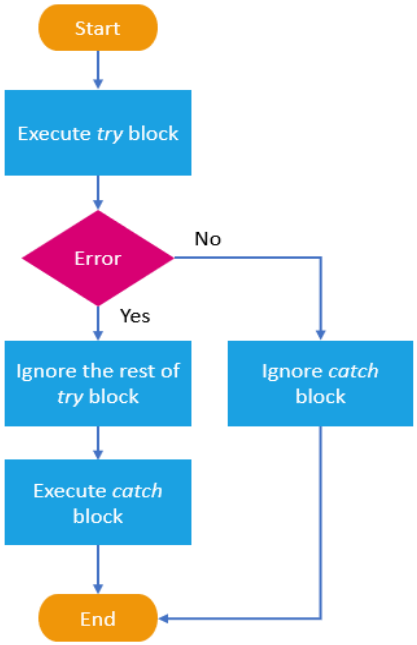


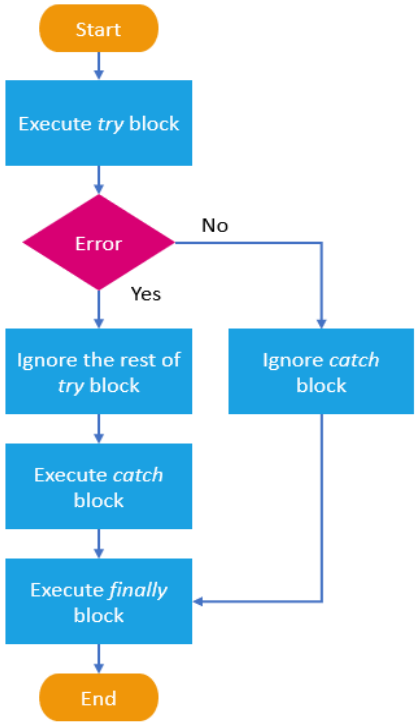

- Khi gặp lỗi, chúng ta có thể:
  - Xử lý lỗi ngay tại đoạn chương trình gây ra lỗi: handle exception.
  - Tạo lỗi và tự động lan truyền lỗi lên trên để tầng trên xử lý: create/throw exception.
    - Có thể tạo ra lỗi mới để lan truyền.
    - Có thể sử dụng lỗi do Java định nghĩa để lan truyền.

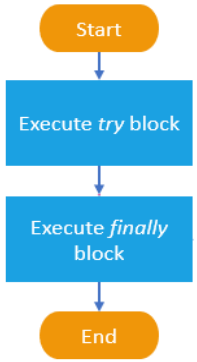
➔ Việc xử lý lỗi bắt buộc phải thực hiện tại một tầng nào đó trong chương trình.

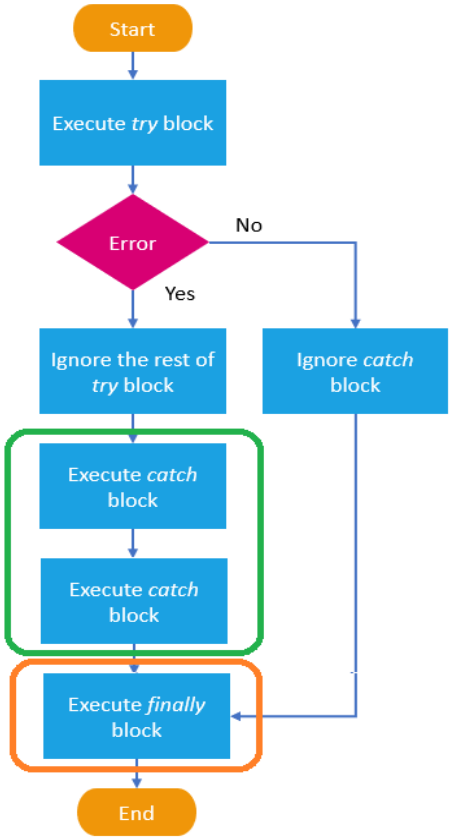
# Cơ chế kiểm soát lỗi

- **Các cơ chế kiểm soát lỗi với try … catch … finally**
  - Cho phép kiểm soát các lỗi compile-time và run-time exception.
  - Java hỗ trợ các cách tiếp cận:



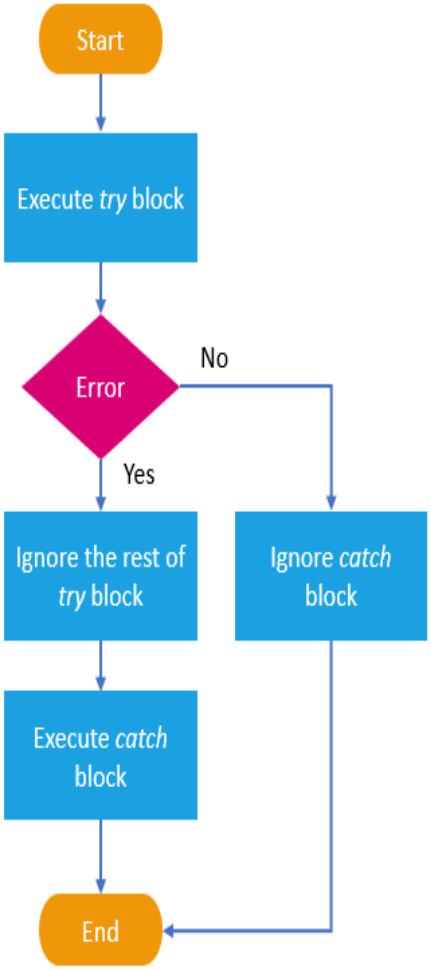



# Cơ chế kiểm soát lỗi

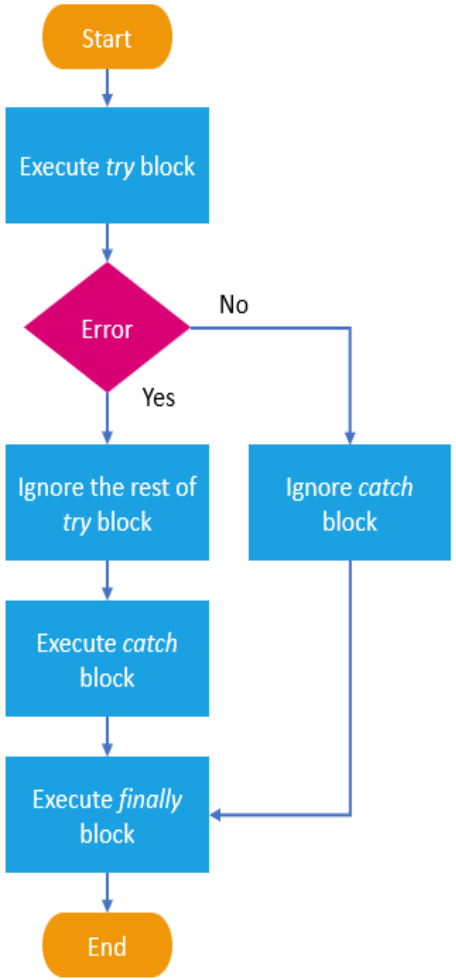
- **Cấu trúc kiểm soát lỗi với try … catch**




```
public class SimpleTryCatch {

    public static void main(String[] args) {
        int[] i_lists = { 1, 5, 8, 19 };
        try{
            // Thực hiện các câu lệnh
            int total = i_lists[0] + i_lists[4];
            System.out.println("Total: " + total);
        }catch (ArrayIndexOutOfBoundsException e){
            // Nếu có lỗi, sẽ xử lý lỗi
            System.out.println("Có lỗi khi truy cập sai chỉ số mảng.");
        }
    }
}
```

# Cơ chế kiểm soát lỗi

- **Cấu trúc kiểm soát lỗi với try … catch … finally**




```java
public static void main(String[] args) {
    int[] i_lists = { 1, 5, 8, 19 };
    try {
        // Thực hiện các câu lệnh
        int total = i_lists[0] + i_lists[4];
        System.out.println("Total: " + total);
    } catch (ArrayIndexOutOfBoundsException e) {
        // Nếu có lỗi, sẽ xử lý lỗi
        System.out.println("Có lỗi khi truy cập sai chỉ số mảng.");
    } finally{
        System.out.println("Kết thúc chương trình ở đây.");
    }
}
```

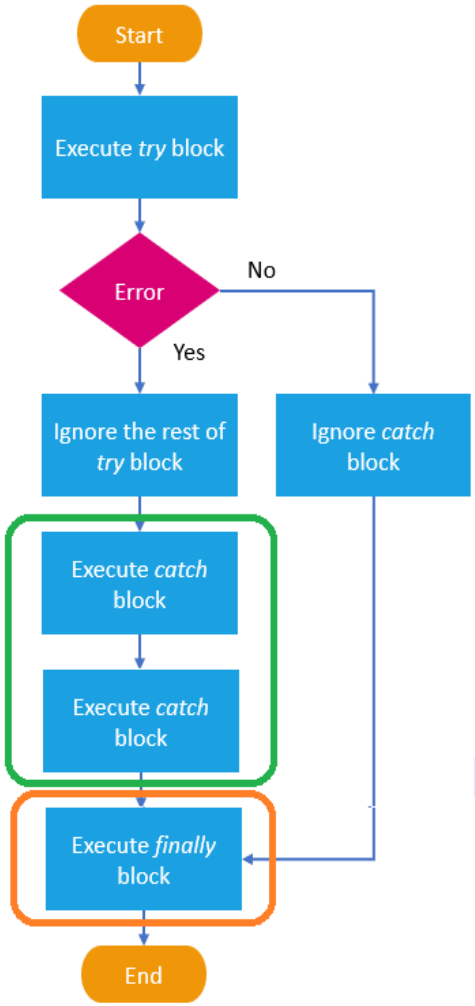
Markers | Properties | Problems | Console | Debug

<terminated> SimpleTryCatch [Java Application] C:\Program Files\Java\jdk1.8.0_271\bin\javaw.exe (Mar 14, 2021, 9:19:56 PM)

```
Có lỗi khi truy cập sai chỉ số mảng.
Kết thúc chương trình ở đây.
```

# Cơ chế kiểm soát lỗi

- **Cấu trúc kiểm soát lỗi với try … catch (multi-catch)**

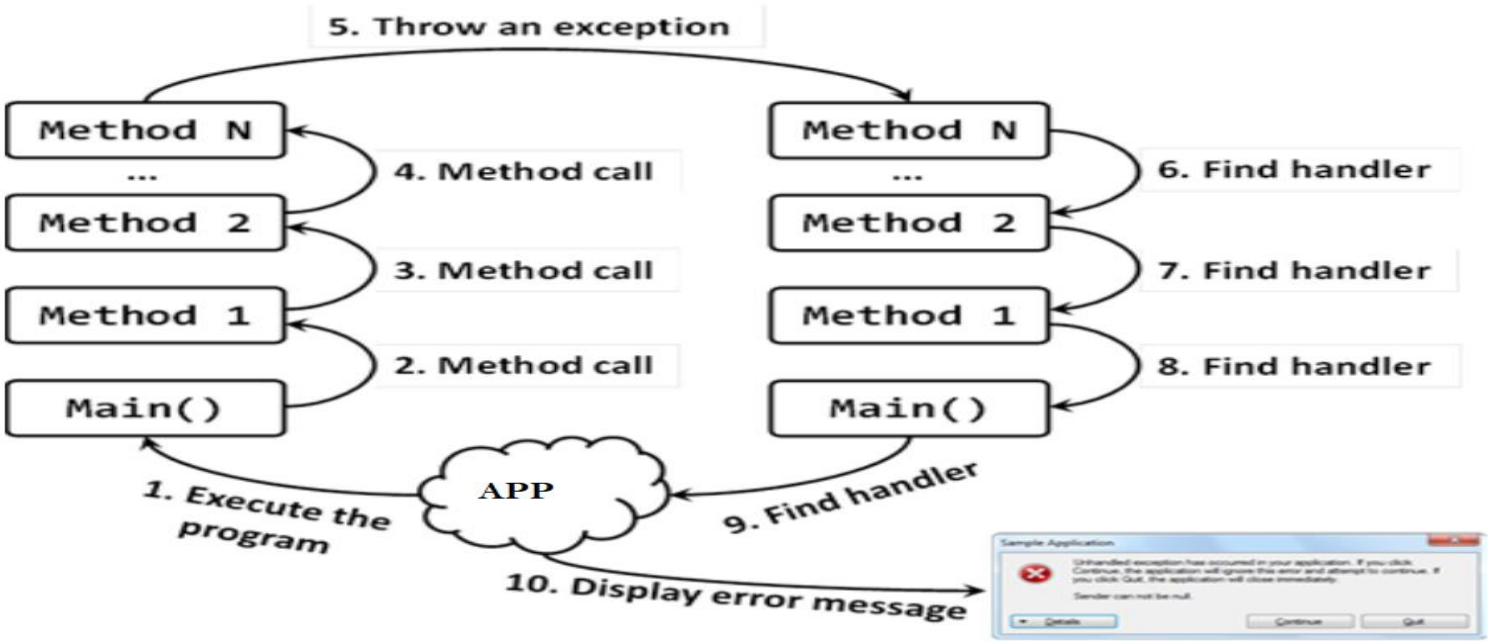



```
public static void main(String[] args) {
    int[] i_lists = { 1, 5, 8, 19 };
    int[] null_lists = null;
    try {
        // Thực hiện các câu lệnh
        int total = i_lists[0] + i_lists[4];
        System.out.println("Total: " + total);
        // Làm việc với null pointer
        System.out.println("Danh sách chưa cấp phát:" + null_lists[1]);
    } catch (ArrayIndexOutOfBoundsException e) {
        // Nếu có lỗi, sẽ xử lý lỗi
        System.out.println("Có lỗi khi truy cập sai chỉ số mảng.");
    } catch (NullPointerException e) {
        // Nếu có lỗi, sẽ xử lý lỗi
        System.out.println("Có lỗi khi truy cập sai chưa cấp phát.");
    } finally {
        // Kết thúc - giải phóng và dọn dẹp
        System.out.println("Kết thúc chương trình");
    }
}
```

```
Markers  Properties  Problems  Console  Debug
<terminated> SimpleTryCatch [Java Application] C:\Program Files\Java\jdk1.8.0_271\bin\javaw.exe (Mar 14, 2021, 9:24:34 PM)
Có lỗi khi truy cập sai chỉ số mảng.
Kết thúc chương trình
```

# Cơ chế lan truyền lỗi

- Lỗi được tự động lan truyền nếu như chúng ta không thực hiện việc kiểm soát lỗi.
  - Cho phép tập trung việc kiểm soát các lỗi tại một mức nhất định để có thể đồng bộ và thực hiện chung một hành động để xử lý lỗi.
  - Cho phép xử lý các lỗi nghiệp vụ [Java không hỗ trợ] bằng cách định nghĩa lỗi hoặc sử dụng lỗi đã có trong Java.



# Cơ chế lan truyền lỗi

- **Ví dụ về sự lan truyền lỗi**

```
public class AutoThrowExample {

    public static void main(String[] args) {
        AutoThrowExample object = new AutoThrowExample();
        object.functionA();
    }

    public int functionA() {
        int sum = 0;
        // call B function
        int[] input = this.functionB();
        for (int i : input) {
            sum += i;
        }
        return sum;
    }

    public int[] functionB() {
        int[] input = { 1, 3, 5, 6, 9 };
        // add the exception
        input[5] = 8;
        return input;
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at week_08.exceptionsExample.AutoThrowExample.functionB(AutoThrowExample.java:23)
        at week_08.exceptionsExample.AutoThrowExample.functionA(AutoThrowExample.java:13)
        at week_08.exceptionsExample.AutoThrowExample.main(AutoThrowExample.java:7)
```

# Cơ chế tạo lỗi với throw

- Khi gặp các lỗi nghiệp vụ, chúng ta có thể tạo lỗi để đẩy lên tầng trên xử lý mà không cần xử lý ngay trong hàm.
- Có thể tự định nghĩa lỗi mới hoặc sử dụng lỗi Java cung cấp và đẩy lỗi lên trên thông qua **throw**.

```
public class ThrowExceptionExample {

    public static void main(String[] args) {
        ThrowExceptionExample object = new ThrowExceptionExample();
        try {
            object.functionA(12);
        } catch (ArithmeticException e) {
            System.out.println("Wrong data!");
        }
    }

    public void functionA(int n) {
        // The input must be under 10.
        if (n >= 10) {
            throw new ArithmeticException("The input is incorrect!");
        } else {
            System.out.println("Correct value: " + n);
        }
    }
}
```

# Cơ chế bắt lỗi với throws

- Có thể sử dụng **throws** để bắt lỗi để đẩy lỗi lên tầng trên nhằm yêu cầu tầng trên bắt buộc phải kiểm soát lỗi.
- **Throws** chỉ hỗ trợ yêu cầu tầng trên với các checked exception.

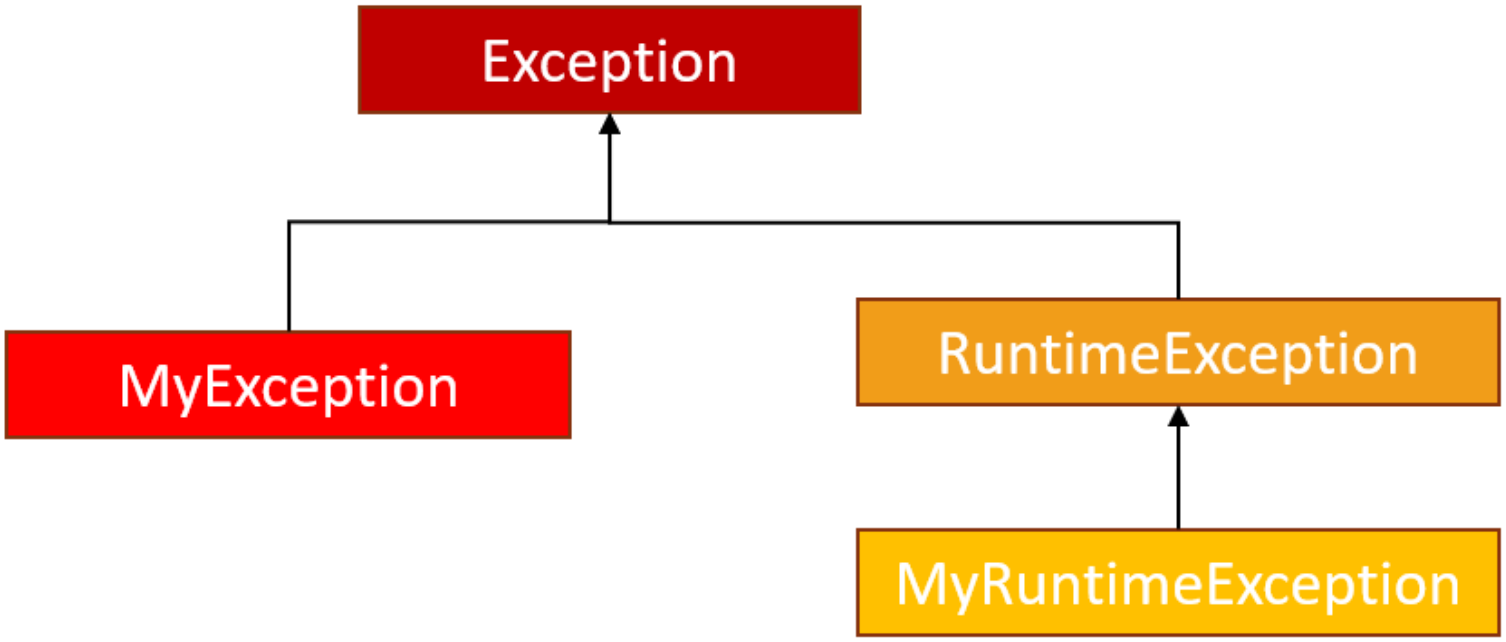
```
public class ThrowExceptionExample {

    public static void main(String[] args) {
        ThrowExceptionExample object = new ThrowExceptionExample();
        object.functionA(12);
        try {
            object.functionA(12);
        } catch (ArithmeticException e) {
            System.out.println("Wrong data!");
        } catch (FileNotFoundException e){
            System.out.println("Sorry, it's an example!");
        }
    }

    public void functionA(int n) throws FileNotFoundException{
        // The input must be under 10.
        if (n >= 10) {
            throw new ArithmeticException("The input is incorrect!");
        } else {
            System.out.println("Correct value: " + n);
        }
    }
}
```

# Throw vs Throws

| No | throw | throws |
|---|---|---|
| 1 | Từ khóa throw trong java được sử dụng để ném ra một ngoại lệ rõ ràng. | Từ khóa throws trong java được sử dụng để khai báo một ngoại lệ. |
| 2 | Ngoại lệ checked không được truyền ra nếu chỉ sử dụng từ khóa throw. | Ngoại lệ checked được truyền ra ngay cả khi chỉ sử dụng từ khóa throws. |
| 3 | Sau throw là một instance. | Sau throws là một hoặc nhiều class. |
| 4 | Throw được sử dụng trong phương thức. | Throws được khai báo ngay sau dấu đóng ngoặc đơn của phương thức. |
| 5 | Bạn không thể throw nhiều exceptions. | Bạn có thể khai báo nhiều exceptions, ví dụ:<br>public void method() throws IOException, SQLException{<br>…<br>} |

# Định nghĩa mới Exception

- Trong thực tế, có những lỗi chúng ta có thể xử lý chung một nghiệp vụ ➔ có thể định nghĩa lỗi mới để kiểm soát đơn giản.
- Các lỗi định nghĩa đều kế thừa từ lớp Exception (checked) hoặc RuntimeException (unchecked).



# Định nghĩa mới Exception

```
public class UseMyException {
    private double mathMark;

    public double getMathMark() {
        return mathMark;
    }

    public void setMathMark(double mathMark) throws InvalidNumberException {
        if (mathMark < 0 || mathMark > 10) {
            throw new InvalidNumberException(mathMark +
                    " is out of allowed mark range [0, 10].");
        }
        this.mathMark = mathMark;
    }

    public static void main(String[] args) {
        UseMyException object = new UseMyException();
        try {
            object.setMathMark(20);
        } catch (InvalidNumberException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
public class InvalidNumberException extends Exception {

    private static final long serialVersionUID = 1L;

    public InvalidNumberException(String message){
        super(message);
    }
}
```

# Bài tập

- **Các bài tập với try … catch … finally**

1. Xây dựng chương trình cho phép từ một tập số nguyên được sinh ra ngẫu nhiên (randomize), cho phép người dùng lựa chọn một giá trị n bất kỳ và kiểm tra thông qua cơ chế try … catch … finally.
   - Nếu người dùng không nhập số nguyên ➔ đưa ra Exception.
   - Nếu giá trị n vượt quá kích thước của tập số nguyên ➔ đưa ra Exception.
   - Nếu giá trị n thỏa mãn, hiển thị giá trị của phần tử thứ n.
2. Viết chương trình nhập chuỗi String là các số cách nhau bởi dấu cách và kiểm tra xem có thỏa mãn không, nếu đúng thì có bao nhiêu số.
   - S_input = “2 34.5 -12.9 0 12 1.98” ➔ Output: true, có 6 số.
   - S_input = “2 34.5 -12a.9 0 12 1b.98” ➔ Output: false.

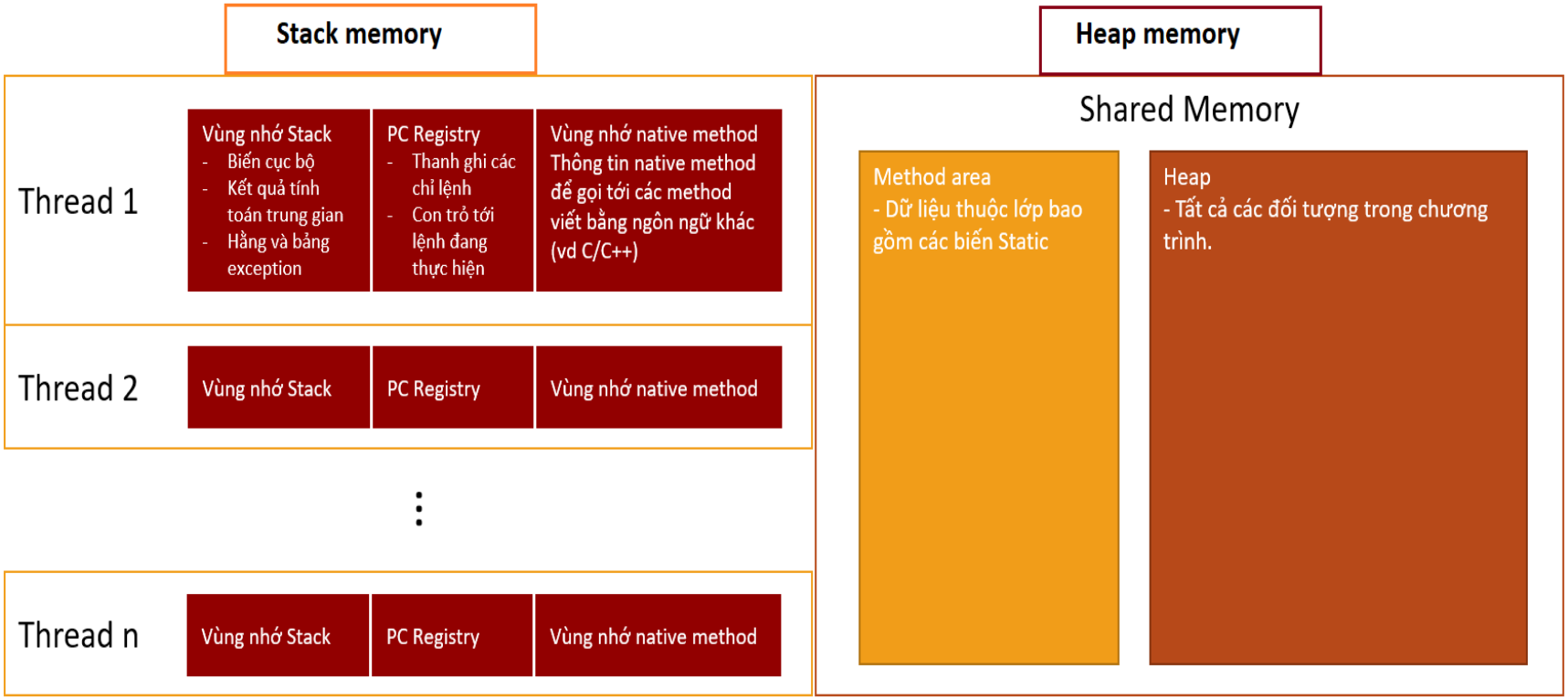
- **Chú ý**
  - Có thể xây dựng các hàm con và truyền lỗi lên hàm main để xử lý.
  - Có thể định nghĩa lỗi mới – InvalideNumberException.
  - Sử dụng finally trong trường hợp kết thúc chương trình.

Chương 4:

# 3. Quản lý bộ nhớ trong Java

# Quản lý bộ nhớ - JVM

- Khi chạy chương trình Java, JVM sẽ yêu cầu hệ điều hành cấp phát bộ nhớ trong RAM để thực thi chương trình.
- JVM chia bộ nhớ thành 2 phần: vùng nhớ chung và vùng nhớ cho các Thread.

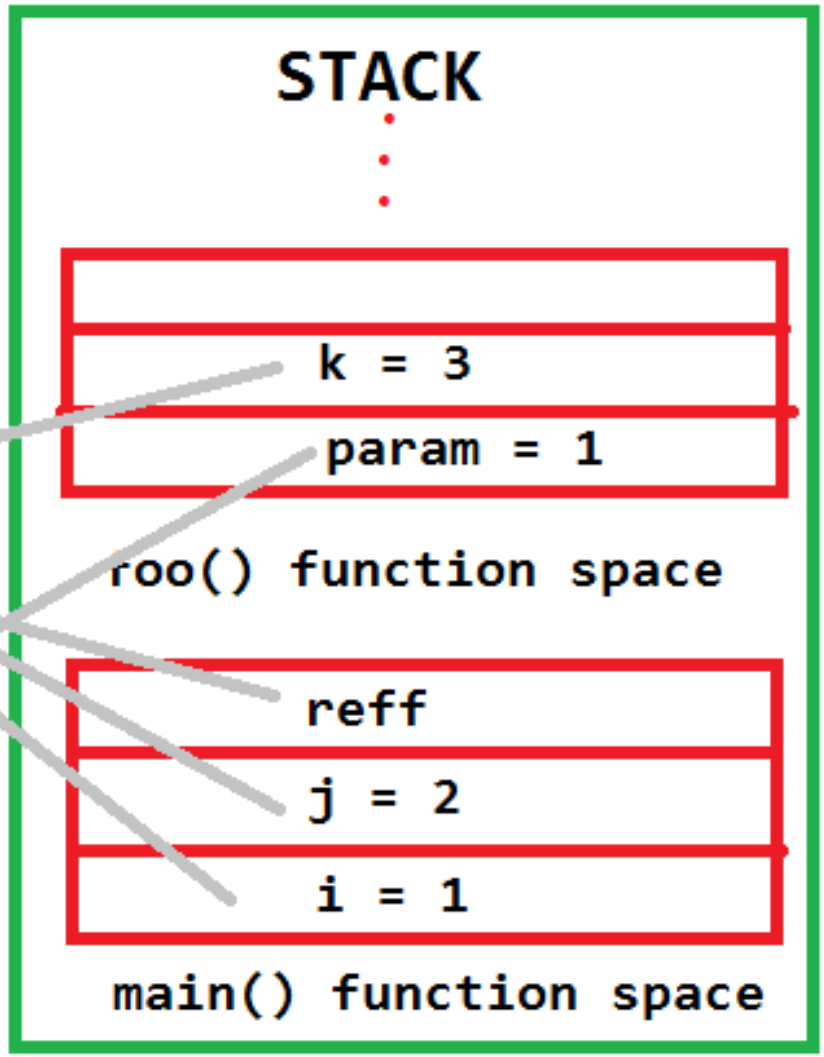


# Quản lý bộ nhớ - Stack

- Vùng nhớ Stack:
  - Được tạo ra cho từng Thread nhằm lưu các biến cục bộ, các biến dữ liệu kiểu primitive và các tham chiếu đến các đối tượng trong Heap.
  - Mỗi stack chỉ được sử dụng cho một Thread duy nhất. Thread ngoài không thể truy cập vào được.
  - Khi có 1 hàm được gọi, một khối bộ nhớ mới sẽ được tạo trong Stack cho hàm đó để lưu các biến local. Khi hàm thực hiện xong, khối bộ nhớ cho hàm sẽ bị xoá, và giải phóng bộ nhớ trong stack.
  - Sử dụng cấu trúc LIFO (Last In First Out) cho mỗi hàm được gọi.
  - Stack memory có kích thước rất nhỏ so với Heap memory, khi đầy bộ nhớ, JVM sẽ thông báo lỗi java.lang.StackOverFlowError và chúng ta có thể tăng kích thước vùng nhớ Stack bằng tham số -Xss

# Quản lý bộ nhớ - Stack

```
public class Stack_Test {

 public static void main(String[] args) {

        int i=1;
        int j=2;

        Stack_Test reff = new Stack_Test();
        reff.foo(i);
    }

     void foo(int param) {
        int k = 3;
        System.out.println(param);
    }

}
```

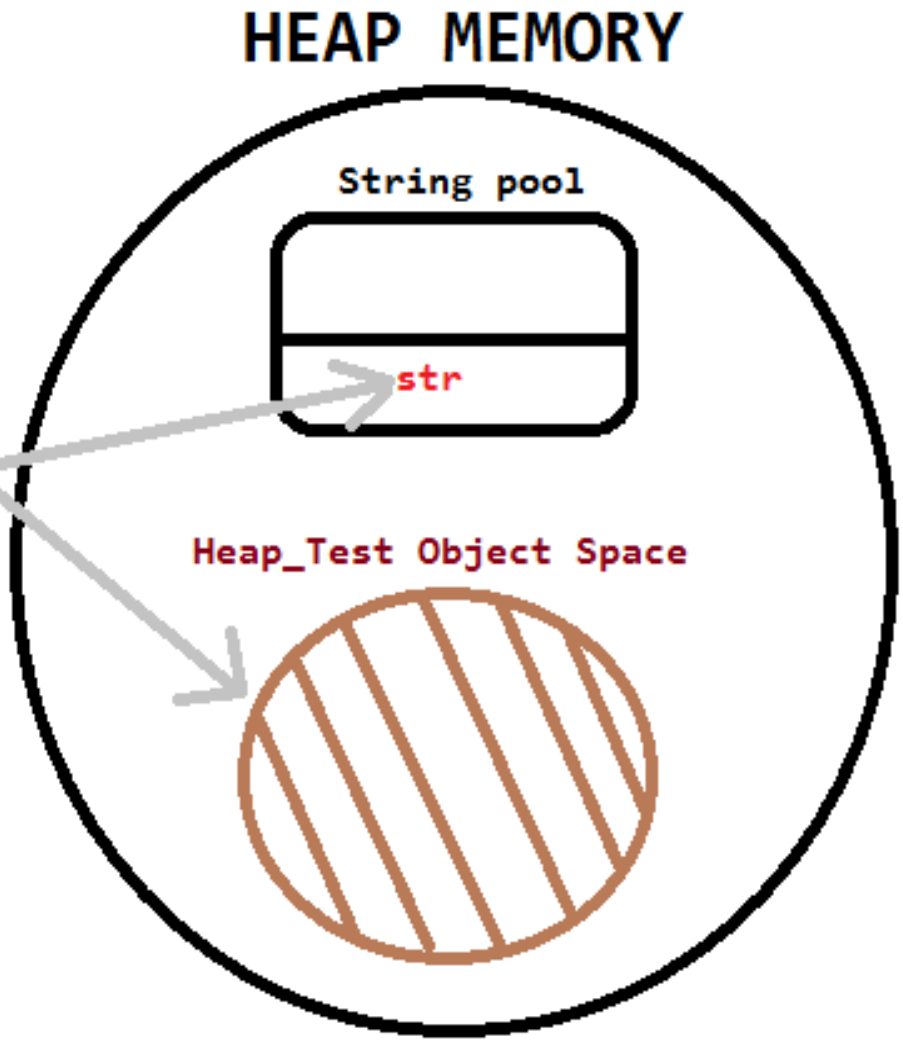


# Quản lý bộ nhớ - Heap

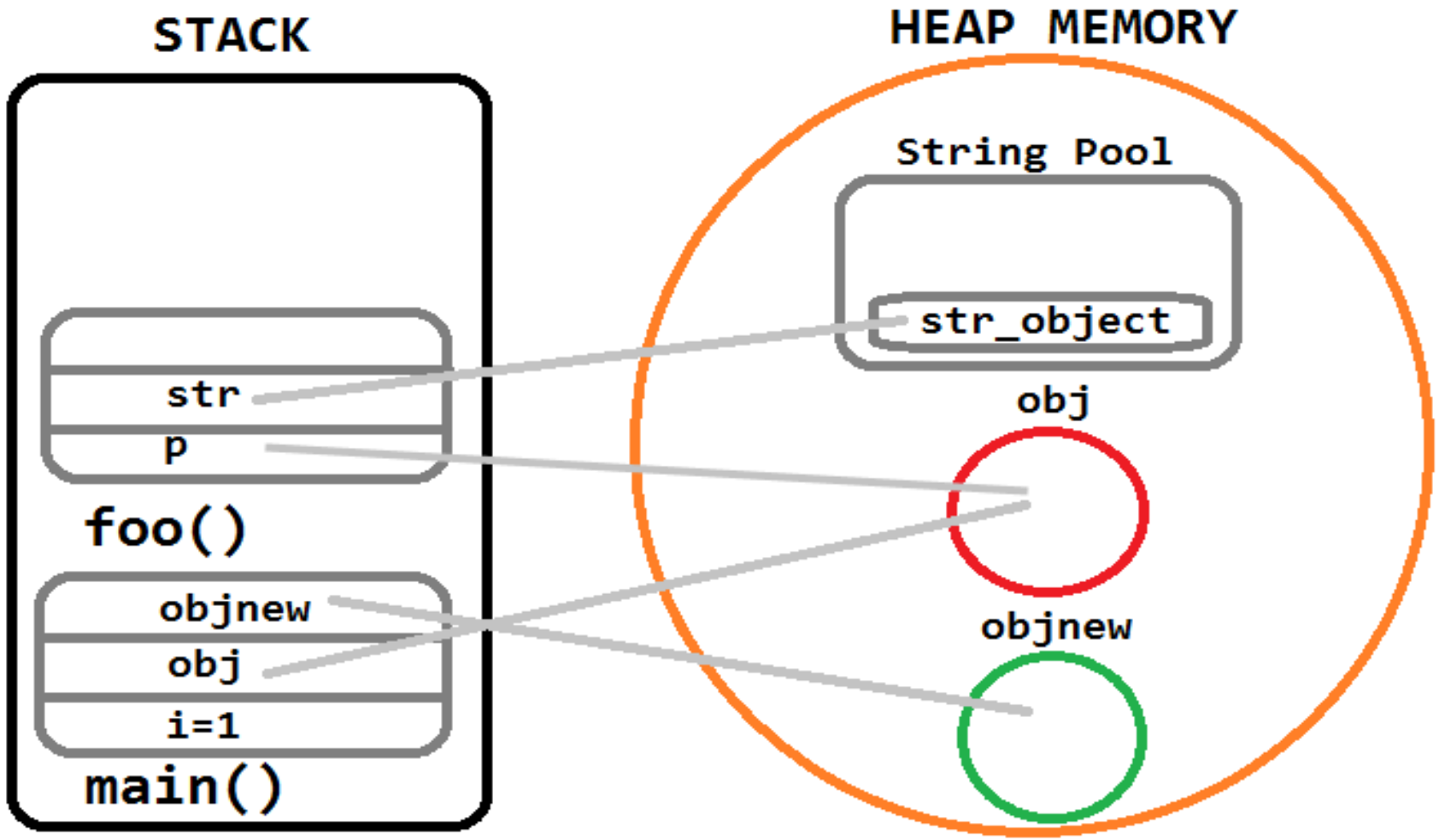
- Vùng nhớ Heap:
  - Java Heap Memory là bộ nhớ được sử dụng ở runtime để lưu các Objects. Bất cứ khi nào ở đâu trong chương trình của bạn khi bạn tạo Object thì nó sẽ được lưu trong Heap (thực thi toán tử new).
  - Các objects trong Heap đều được truy cập bởi tất cả các các nơi trong ứng dụng, bởi các Threads khác nhau.
  - Dung lượng sử dụng của Heap sẽ tăng giảm phụ thuộc vào Objects sử dụng. Garbage Collection sẽ chạy trên bộ nhớ Heap để xoá các Object không được sử dụng nữa.
  - Heap memory có kích thước lớn hơn rất nhiều so với Stack memory, khi đầy bộ nhớ, JVM sẽ thông báo lỗi java.lang.OutOfMemoryError: Java Heap Space.
  - Sử dụng -Xms và -Xmx để định nghĩa dung lượng bắt đầu và dung lượng tối đa của bộ nhớ Heap.

# Quản lý bộ nhớ - Heap

```
public class Heap_Test {

    public static void main(String[] args)
    {

        Heap_Test reff = new Heap_Test();

        reff.foo();
    }

     void foo() {

       String str = "Heap memory space";

        System.out.println(param);
    }

}
```

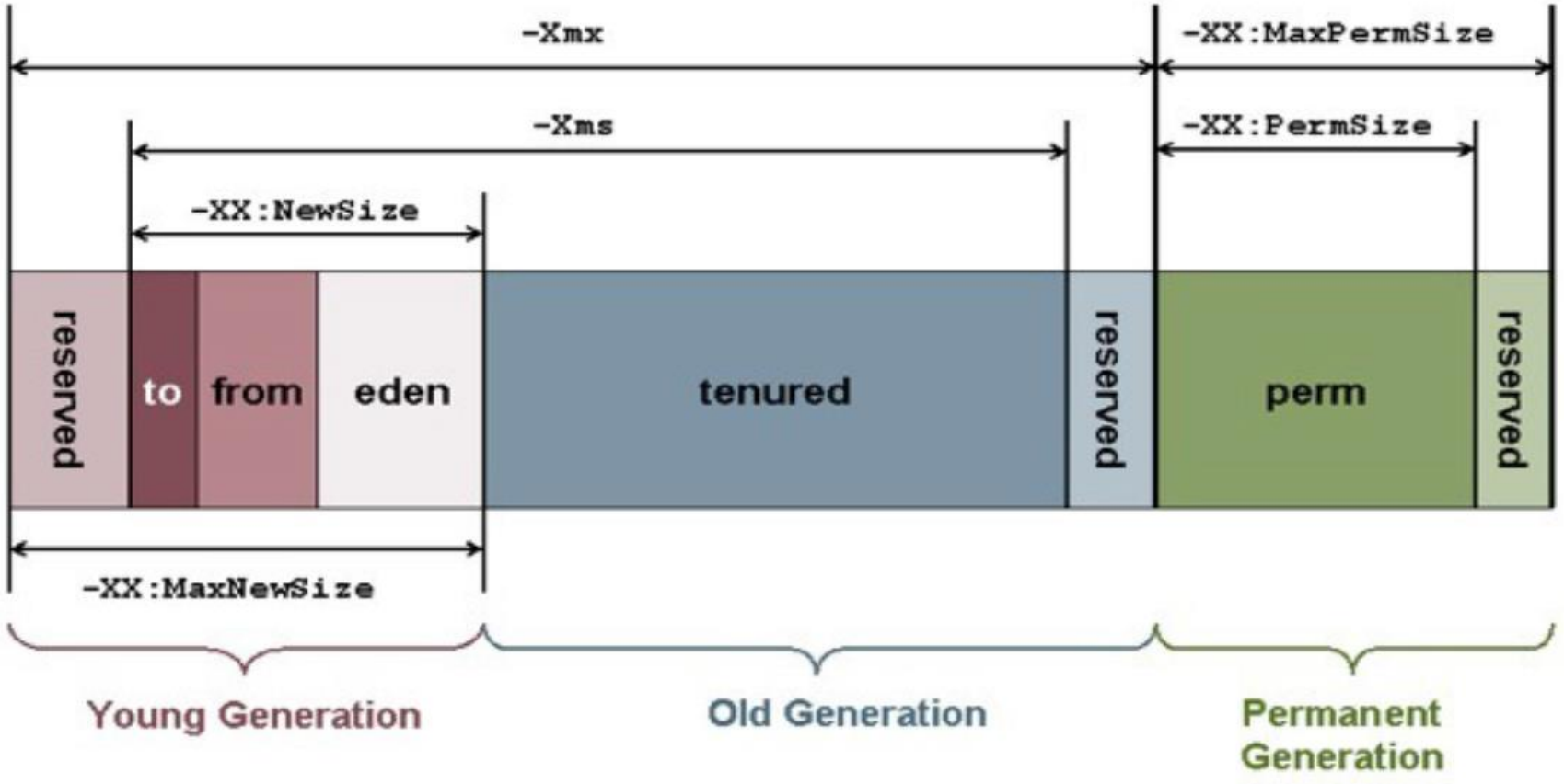


# Heap vs Stack



# Giải phóng bộ nhớ

- Java cung cấp công cụ giải phóng bộ nhớ tự động Garbage collectors – GC ➔ lập trình viên không cần phải giải phóng bộ nhớ.



# Bài tập

- **Các bài tập với Sodoku**

1. Từ chương trình Sodoku đã được xây dựng, hãy thiết kế lại chương trình với việc sử dụng các cơ chế kiểm soát lỗi try ... catch … finally
   - Có thể kiểm soát ngay lỗi nhập dữ liệu ➔ nếu giá trị đầu vào sai thì thông báo và yêu cầu nhập lại giá trị.
   - Nếu các kiểm tra không phù hợp ➔ có thể sinh lỗi và yêu cầu nhập lại giá trị.
   - Có thể định nghĩa lỗi mới để đồng bộ cách xử lý cho các nghiệp vụ.
   - Có thể xử dụng finally để đóng kết nối của scanner khi dừng chương trình.

- **Chú ý**
   - Trong bài tập này, có thể cơ chế kiểm soát lỗi bằng try … catch … finally không phải là lựa chọn tốt hơn so với việc sử dụng câu lệnh điều kiện.
   - Có thể đưa ra các trường hợp lỗi khác để xử lý.
   - Tầng xử lý lỗi nên để ở tầng Controller nhằm đồng bộ hóa các lỗi cho chương trình.

# CÔNG NGHỆ JAVA