TRƯỜNG ĐẠI HỌC

GIAO THÔNG VẬN TẢI

University of Transport and Communications

HỌC PHẦN: CÔNG NGHỆ JAVA

Người thực hiện: Đào Thị Lệ Thủy

Chương 4. XỬ LÝ LỖI & THU GOM RÁC TRONG JAVA

XỬ LÝ LỖI

Khái niệm

Lỗi có 2 loại:

- Lỗi biên dịch (compile-time error): Sai cú pháp
- Lỗi khi thực thi (run-time error)
  - ❖ Giải thuật sai
  - ❖ Lỗi vào ra
  - ❖ …

Exception = runtime-error

Khi 1 exception xẩy ra, chương trình kết thúc đột ngột và điều khiển được trả lại cho OS ➔ Cần phải quản lý được các tình huống này

Khái niệm

Khi 1 exception xẩy ra, chương trình kết thúc đột ngột và điều khiển được trả lại cho OS

Cần phải quản lý được các tình huống này

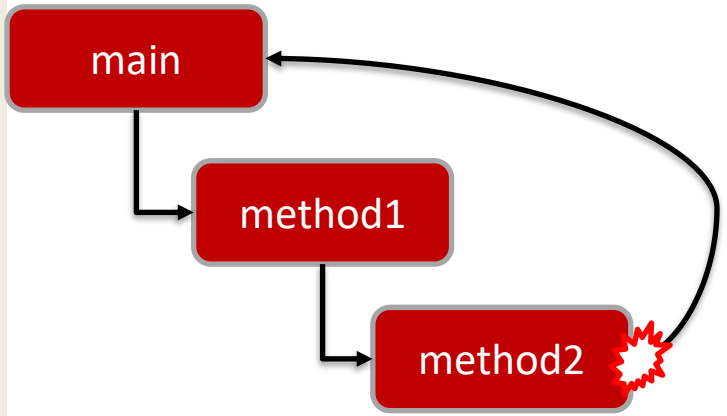
```
package exception_demo;
public class AccessErrorDemo {
    public static void main(String[] args) {
        int[] arr = new int[] { 1, 2, 3 };
        System.out.println(arr[0]);
        System.out.println(arr[2]);
        System.out.println(arr[4]);
        System.out.println(arr[1]);
    }
}
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 4
at exception_demo.AccessErrorDemo.main(AccessErrorDemo.java:9)
Stack trace: Thông tin chi tiết lỗi xảy ra ở dòng nào, file nào,
các phương thức được gọi thế nào
```

Khái niệm

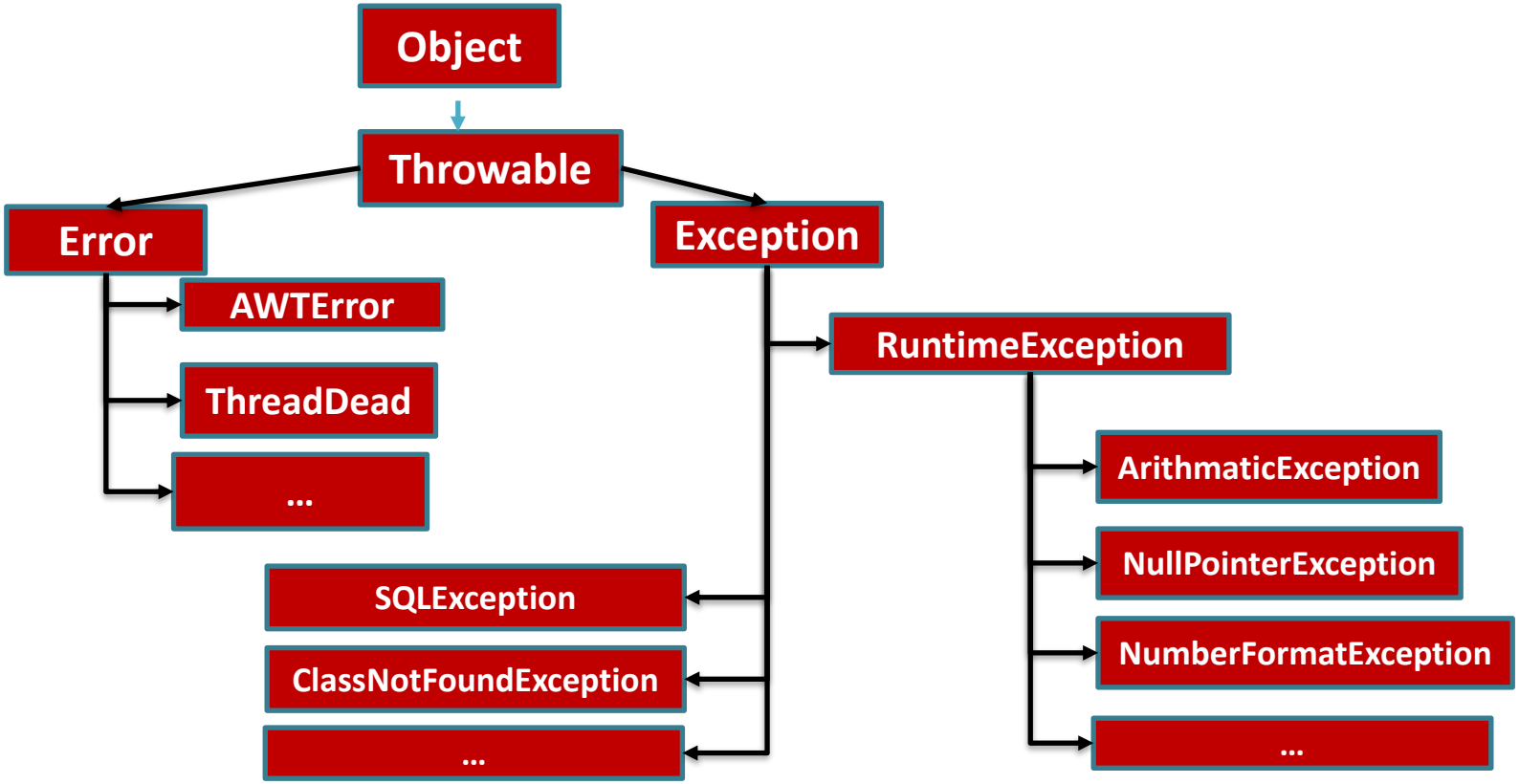
- Lỗi được lan truyền qua các lời gọi hàm

```
public class ExceptionDemo {

    public static void main(String[] args) {
        method1();
        System.out.println("Finish");
    }

    private static void method1() {
        method2();
    }

    private static void method2() {
        int[] arr = new int[] {1};
        System.out.println(arr[2]);
    }
}
```




```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 2
at exception_demo.ExceptionDemo.method2(ExceptionDemo.java:16)
at exception_demo.ExceptionDemo.method1(ExceptionDemo.java:11)
at exception_demo.ExceptionDemo.main(ExceptionDemo.java:6)
```

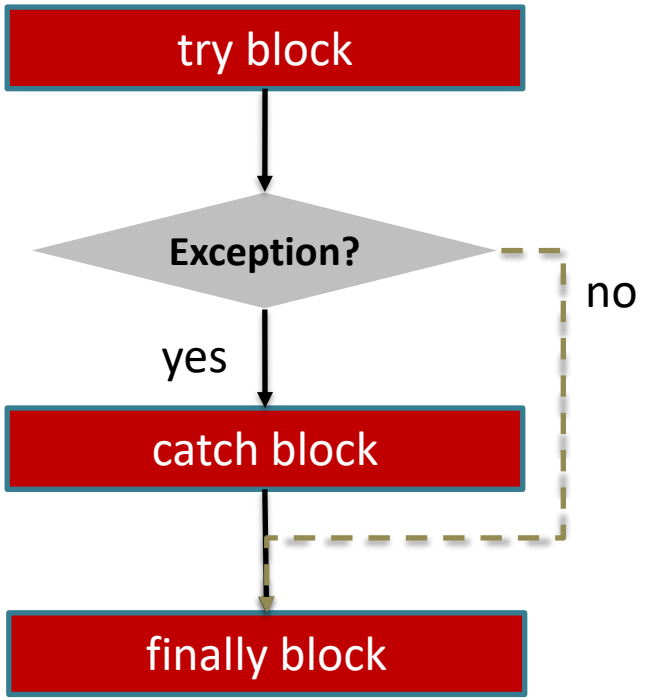
Cấu trúc các lớp quản lý lỗi trong Java



Khi 1 error/exception xẩy ra, ta nói rằng nó bị "thrown" (throw)

Mô hình try...catch...finally...

```
try {
//...
}
catch(Exception e) {
//...
}
finally {
//...
}
```



Thực thi 1 khối lệnh (try), nếu xuất 1 lỗi thì bẫy lỗi (catch) để xử lý tình huống lỗi và cuối cùng thực thi tiếp (finally).

Mô hình try...catch...finally...

- Một khối lệnh **try** phải đi kèm theo ít nhất 1 khối lệnh **catch** hoặc **finally**


```
try {
//...
}
Catch(Exception e) {
//...
}
```

```
try {
//...
}
finally {
//...
}
```

- Khối lệnh finally luôn được thực hiện trong mọi tình huống

```
public class TryFinallyDemo {

     public static void main(String[] args) {
          finallyTest();
     }
          public static void finallyTest() {
          try {
               System.out.println("inside try");
          return;
          }
          finally {
               System.out.println("inside finally");
          }
     }
}
```

Mô hình try...catch...finally...

- Một khối lệnh **try** có thể đi kèm nhiều khối lệnh **catch**
  - Khối lệnh **catch** nào được thực hiện sẽ phụ thuộc và exception type của nó
  - Khối lệnh **catch** sẽ bắt được tất cả exception là lớp con của exception type
  - Exception type khối lệnh **catch** đứng trước không được là cha của exception type khối lệnh **catch** đứng sau

```
try {
//...
}
catch(Exception e) {
//...
}
```

Exception type

```
try {
        System.out.println(10 / 0);
}
catch(ArrayIndexOutOfBoundsException e) {
        System.out.println("Array out of bound exception");
}
catch(ArithmeticException e) {
        System.out.println("arithmetic exception");
}
```

arithmetic exception

```
try {
int[] arr = new int[] {1 ,2 ,3};
        System.out.println(arr[4]);
}
catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array out of bound exception");
}
catch(ArithmeticException e) {
System.out.println("arithmetic exception");
}
```

Array out of bound exception

Mô hình try...catch...finally...

- **catch(Exception e)** không được đứng trước các khối lệnh **catch** khác

```
try {
        System.out.println(10 / 0);
}
catch(Exception e) {
        System.out.println(“Exception");
}
catch(ArrayIndexOutOfBoundsException e) {
        System.out.println("Array out of bound exception");
}
catch(ArithmeticException e) {
        System.out.println("arithmetic exception");
}
```

Báo lỗi: Unreachable catch block for `ArrayIndexOutOfBoundsException`. It is already handled by the catch block for Exception

Các từ khóa xử lý ngoại lệ trong java

Ví dụ

Vấn đề khi không có Exception Handling

```
public class Testtrycatch1{
 public static void main(String args[]){
 int data=50/0;//có thể đưa ra exception
System.out.println(“Phần code tiếp theo..."); } }
```

Chương trình sẽ cho kết quả sau:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Các từ khóa xử lý ngoại lệ trong java

Ví dụ Cũng ví dụ trên nhưng sử dụng khối try-catch để xử lý

```
public class Testtrycatch2{
public static void main(String args[]){
  try{
     int data=50/0; }
catch(ArithmeticException e){
        System.out.println(e);}
System.out.println("Phần code tiếp theo...."); } }
```

Chương trình sẽ cho kết quả sau:

java.lang.ArithmeticException: / by zero Phần code tiếp theo....

Lúc này, phần còn lại của code đã được thực thi.

Sử dụng throw/throws

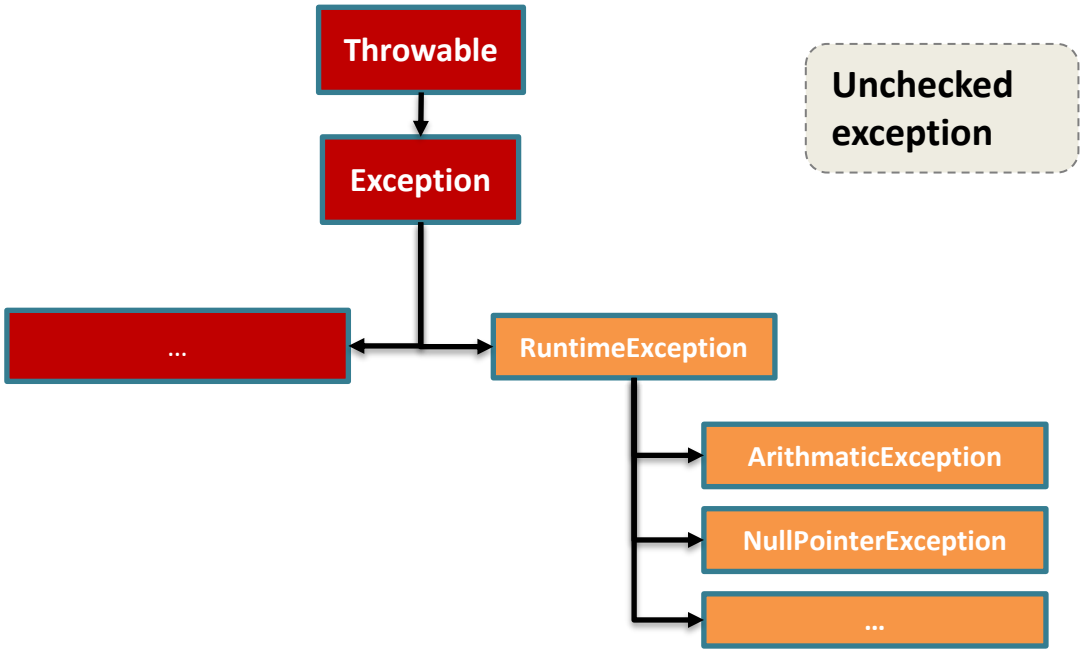
- throw được sử dụng để ném ra 1 exception. Việc này là cần thiết khi chương trình gặp phải những tình huống không mong đợi.
- Ví dụ: Kiểm tra điểm có nằm trong khoảng cho phép hay không

```
package throw_demo;

public class Student {
	private double mathMark;

	public double getMathMark() {
	return mathMark;
	}

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

	public static void main(String[] args) {
		Student s1 = new Student();
		s1.setMathMark(20);
	}
}
```

```
Exception in thread "main"
java.lang.IllegalArgumentException: 20.0 is out of
allowed mark range [0, 10].
at throw_demo.Student.setMathMark(Student.java:12)
at throw_demo.Student.main(Student.java:19)
```

Check và uncheck exception

- Checked exception là những lỗi **có thể dự đoán được** nhưng không hạn chế được.
- Ví dụ: Chương trình đọc file danhsach.txt
  - File có thể không tồn tại
  - File tồn tại nhưng không có quyền đọc
  - ….
- Uncheck exception là những lỗi logic khác
  - Uncheck exception thừa kế từ **RuntimeException**



throws

- Checked exception được kiểm tra tại compile time, nếu trong phương thức:
  - Có throw checked exception
  - Gọi tới phương thức khác throws checked exception

Cần xử lý:

⇒Thực hiện try…catch để xử lý

⇒Ném ra checked exception khác phù hợp hơn

⇒Ném ra runtime exception

⇒Bỏ qua

⇒khai báo danh sách exception qua từ khóa **throws**

throws

```
package throw_demo;
import java.io.IOException;
public class CheckedExceptionDemo {
    public static void main(String[] args) throws InterruptedException
    {
        sleepDemo();

        try {
                throwDemo();
        } catch (Exception e) {
                System.out.println(e.getMessage());
        }
    }

    public static void sleepDemo() throws InterruptedException {
        Thread.sleep(1000);
    }
    public static void throwDemo() throws IOException {
        throw new IOException("demo throw IOException");
    }
}
```

Tự định nghĩa lỗi

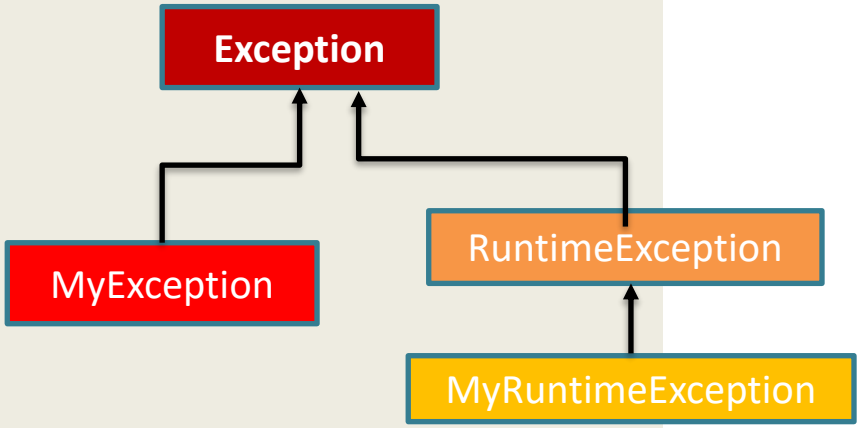
- Có thể tự định nghĩa lỗi bằng cách thừa kế Exception (checked) hoặc RuntimeException (unchecked)

MarkOutOfRangeException.java

```
package throw_demo;
public class MarkOutOfRangeException extends RuntimeException{
     public MarkOutOfRangeException(String message) {
     super(message);
     }
}
```

Student.java

```
package throw_demo;
public class Student {
     private double mathMark;
     public double getMathMark() {
          return mathMark;
     }
     public void setMathMark(double mathMark) {
          if(mathMark < 0 || mathMark > 10) {
               throw new MarkOutOfRangeException(mathMark + " is out of allowed
               mark range [0, 10].");
          }
          this.mathMark = mathMark;
     }
     public static void main(String[] args) {
          Student s1 = new Student();
          s1.setMathMark(20);
     }
}
```



Nguyên tắc khi xử lý lỗi

- Cố gắng try…catch lỗi cụ thể. catch(Exception) đôi khi cần thiết nhưng nhìn chung quá rộng và nên tránh
- Chương trình cần xử lý phù hợp hoặc dừng lại khi cần thiết, tránh lạm dụng try…catch để bỏ qua lỗi (nuốt exception)

  Ví dụ: Chương trình hiển thị danh sách "Tỉnh thành" được lưu trong file **tinhthanh.txt**.

  Khi file không tồn tại (IOException) có thể có 2 lựa chọn:

  Lựa chọn 1. Tạo ra file **tinhthanh.txt** và hiển thị danh sách rỗng.

  - Nếu tạo file không thành công: Báo lỗi và thoát chương trình

  Lựa chọn 2. Báo lỗi và thoát khỏi chương trình

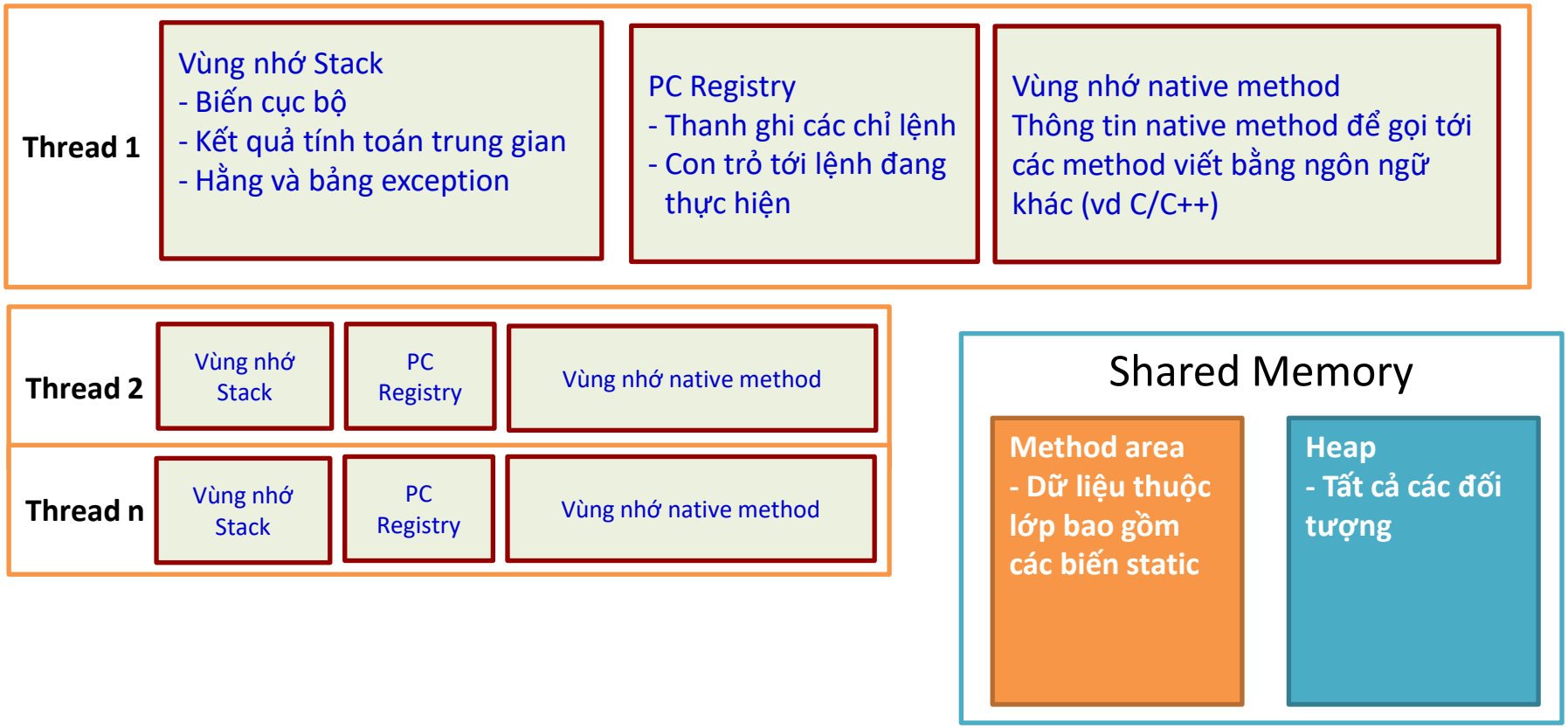
- Tự định nghĩa lỗi khi liên quan đến kiểm tra nghiệp vụ, đầu vào dữ liệu hoặc khi lỗi đi qua các tầng của chương trình

Chương 4. XỬ LÝ LỖI & THU GOM RÁC TRONG JAVA

QUẢN LÝ BỘ NHỚ & THU GOM RÁC

Giới thiệu

- Khi chạy chương trình Java, JVM sẽ yêu cầu hệ điều hành cấp cho một không gian bộ nhớ trong RAM để dùng cho việc chạy chương trình.
- JVM sẽ chia bộ nhớ được cấp phát này thành 2 phần: Vùng nhớ cho từng thread và vùng nhớ chung.

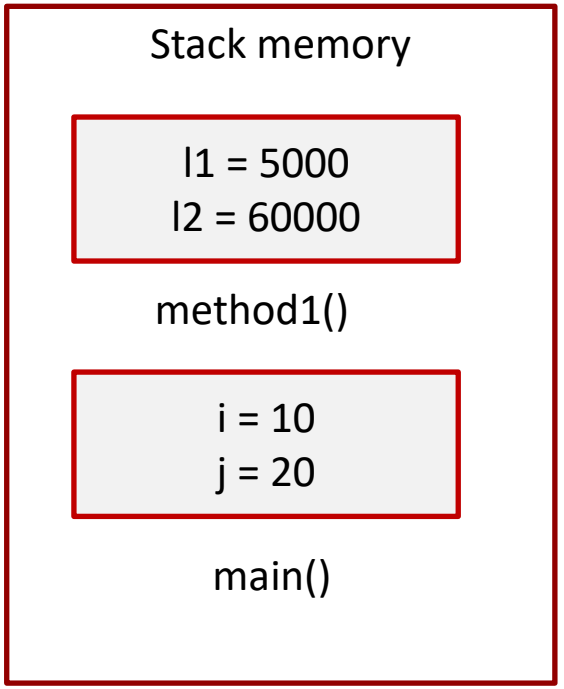


Vùng nhớ stack

- Được tạo ra theo từng thread
- Lưu các biến cục bộ và các biến kiểu dữ liệu primitive (nguyên thủy)
- Tham chiếu đối tượng
- Có cấu trúc LIFO (Last in first out)
    - Khi 1 phương thức được gọi sẽ tạo ra vùng nhớ tạm
    - Khi kết thúc phương thức vùng nhớ được giải phóng và có thể sử dụng cho phương thức tiếp theo

Vùng nhớ stack

```
public class MemoryDemo {
    public static void
    main(String[] args) {
        int i = 10;
        int j = 20;
        method1();
    }

    public static void method1()
    {
        long l1 = 5000;
        long l2 = 60000;
    }
}
```

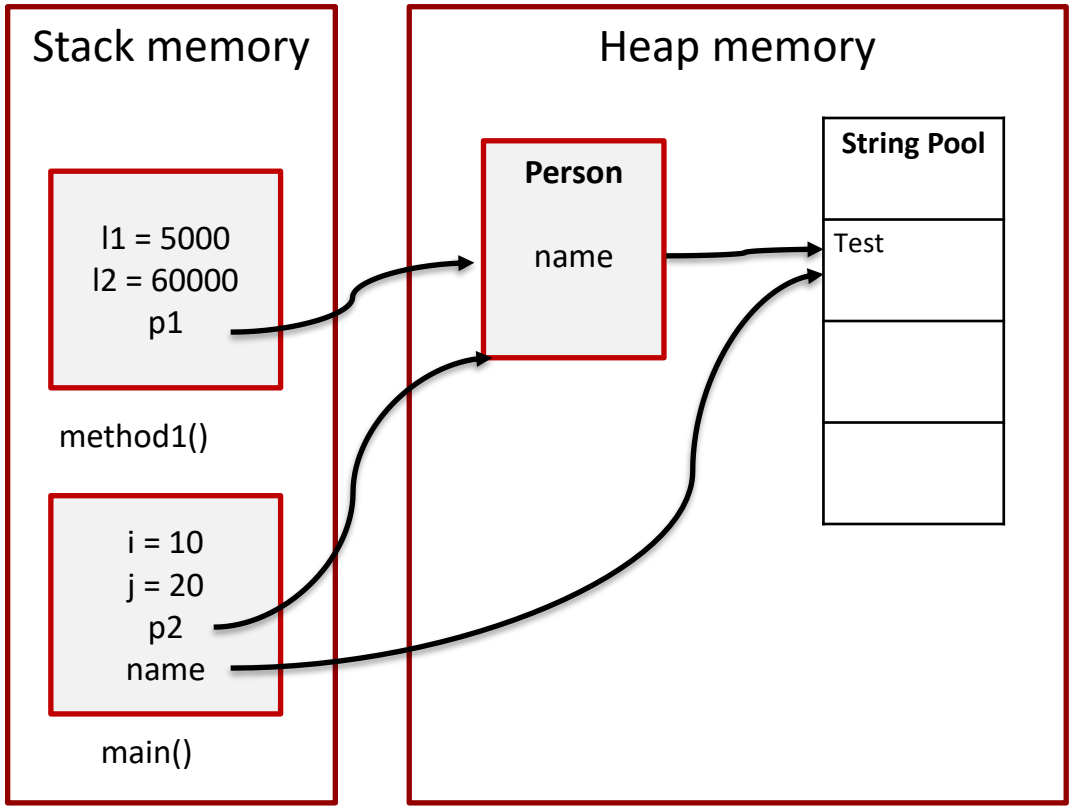


Vùng nhớ heap

- Chia sẻ chung giữa các thread
- Các đối tượng nằm trong vùng nhớ heap
- Có kích thước lớn hơn nhiều so với vùng nhớ stack
- StringPool nằm ở vùng nhớ này (***StringPool*** *là một vùng nhớ đặc biệt dùng để lưu trữ các biến được khai báo theo kiểu* ***String)***

Vùng nhớ heap

```
public class MemoryDemo {
    public static void
    main(String[] args) {
        int i = 10;
        int j = 20;
        Person p2 = method1();
        String name = p2.name;
    }

    public static Person
    method1() {
        long l1 = 5000;
        long l2 = 60000;
        Person p1 = new
        Person();
        p1.name = "Test";
        return p1;
    }
}
```



Thu gom rác

- Sử dụng hàm free() trong ngôn ngữ C và delete() trong C ++. Nhưng, trong java nó được thực hiện tự động. Vì vậy, java cung cấp việc quản lý bộ nhớ tốt hơn.
- Trong java, rác (garbage) có nghĩa là các đối tượng không còn được tham chiếu từ bộ nhớ heap nữa.
- Bộ thu gom rác (**Garbage Collection -** một phần của JVM) trong java được sử dụng để thực hiện quá trình tự động khôi phục lại bộ nhớ không được sử dụng tại runtime một cách tự động (đó là một cách để hủy các đối tượng không sử dụng nữa)

Thu gom rác

Các đối tượng không được tham chiếu:

- Khi gán giá trị null:

```
Person p=new Person();
p=null;
```

- Khi gán đối tượng đến một tham chiếu khác

```
Person p1=new Person();
Person p2=new Person();
 p1=p2;
```

- Khi một đối tượng anonymous (vô danh)

```
new Person();
```

Thu gom rác

- Phương thức gc() được sử dụng để gọi bộ thu gom rác để thực hiện quá trình dọn dẹp. Phương thức gc() được cài đặt trong các lớp System và Runtime.
- Có thể kích hoạt bộ gom rác thông qua **System.gc();**
- Phương thức finalize() sẽ được gọi trước khi đối tượng bị thu gom. Phương thức này có thể được sử dụng để thực hiện xử lý dọn dẹp.

Thu gom rác

```
public class Person {
public String name;

public void finalize() {
  System.out.println("finalized");
}

public static void main(String[] args) {
    Person p1 = new Person();
    Person p2 = new Person();
    p1 = null;
    p2 = null;
    System.gc();
  }
}
```

Kết quả in ra:

finalized

finalized

Tài liệu tham khảo

- https://gpcoder.com/2160-quan-ly-bo-nho-trong-java-voi-heap-space-vs-stack/
- https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/
- https://brucehenry.github.io/blog/public/2018/02/07/JVM-Memory-Structure/