

NGÔN NGỮ LẬP TRÌNH JAVA

Nguyễn Huy Cường
nh.cuong@hutech.edu.vn

04/2024

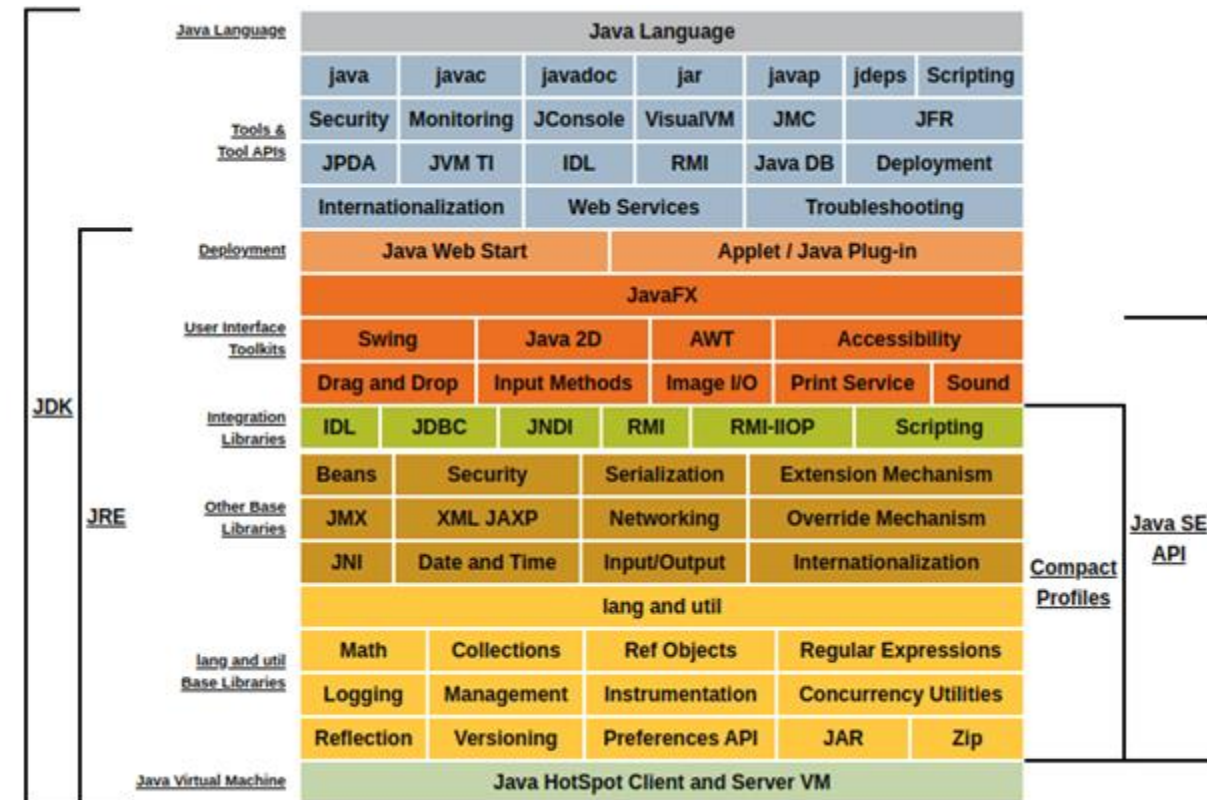
Nội dung

1. Java Platform
2. Chuẩn coding trong Java
3. Cú pháp cơ bản
4. Hướng đối tượng
5. Một số tính năng mới trên các JDK

1. Java Platform

- **JVM (Java Virtual Machine):** là máy ảo biên dịch Java code thành bytecode. Sau đó, JVM sẽ thông dịch bytecode thành mã máy để CPU thực thi.
- **JRE (Java Runtime Environment):** bao gồm JVM + các thư viện và những thành phần bổ sung
- **JDK (Java Development Kit)** là 1 bộ công cụ phát triển ứng dụng Java.

JDK = JRE + công cụ phát triển ứng dụng viết bằng java.



2. Java Coding Standards

<https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

NAMING CONVENTIONS	APPLICATION	EXAMPLES
Lower Camel Case	variables and methods	firstName timeToFirstLoad indexNumber
Upper Camel Case	classes, interfaces, annotations, enums, records	TomcatServer RestController WriteOperation
Screaming Snake Case	constants	INTEREST_RATE MINIMUM_SALARY EXTRA_SAUCE
lower dot case	packages and property files	java.net.http java.management.rmi application.properties
kebab case	not recommended	landing-page.html game-results.jsp 404-error-page.jsf

3. Cú pháp cơ bản

- Cấu trúc chương trình java

```
// Chương trình thuộc package nào
package package_name;

// Import class vào để sử dụng
import java.util.Scanner;

// Định nghĩa thêm các class khác
class Util {
    ...
}

// Class chính, là public class duy nhất trong file
public class App {
    ...
    // Class chính phải có method main() như sau
    public static void main(String[] args) {
        ...
    }
}
```

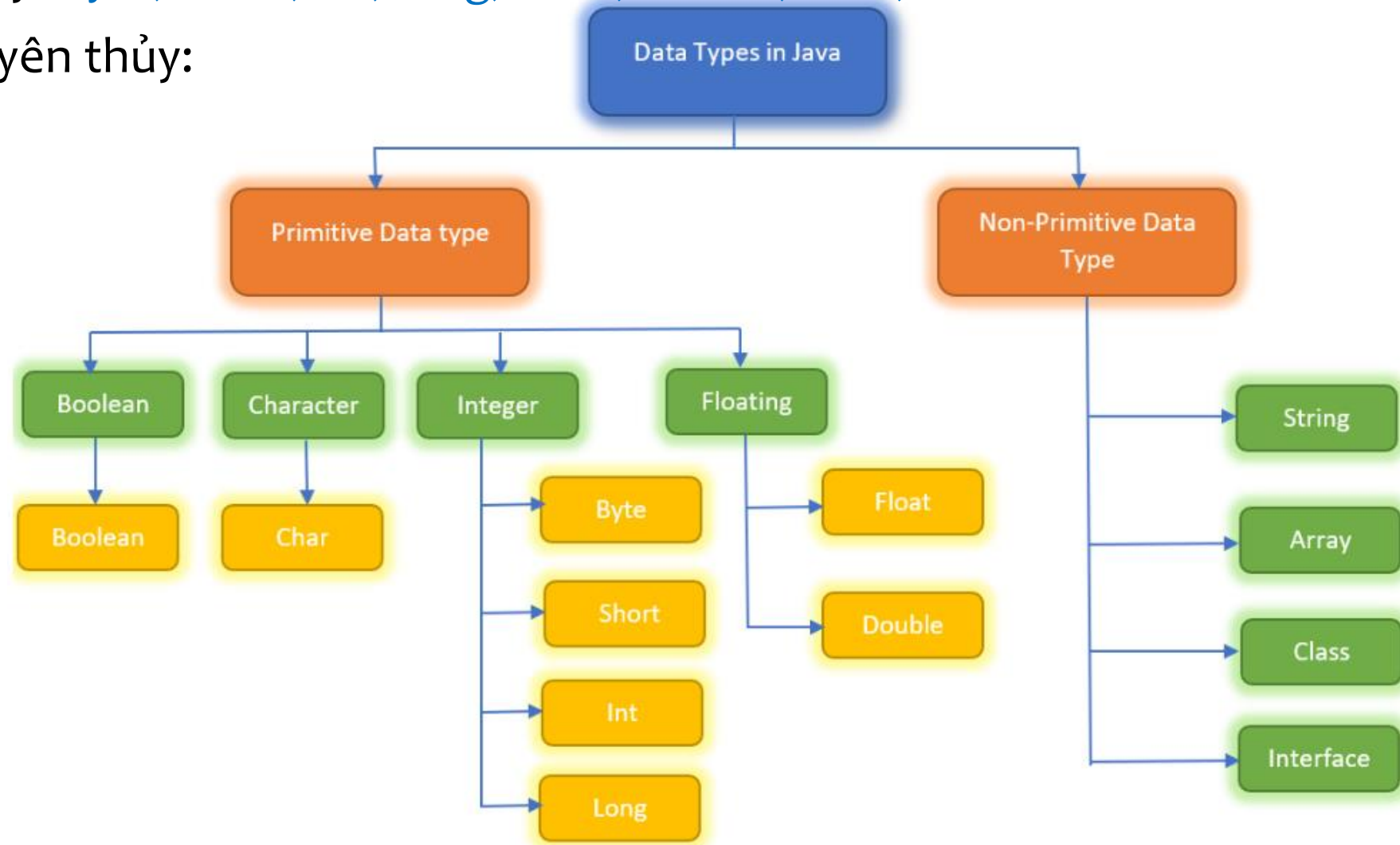
3. Cú pháp cơ bản

- Từ khóa

C# keyword	Java keyword	C# keyword	Java keyword	C# keyword	Java keyword	C# keyword	Java keyword
abstract	abstract	Explicit	N/A	object	N/A	this	This
as	N/A	Extern	native	operator	N/A	throw	Throw
base	Super	Finally	finally	out	N/A	true	True
bool	boolean	Fixed	N/A	override	N/A	try	try
break	break	Float	float	params	N/A	typeof	N/A
byte	N/A	For	for	private	private	unit	N/A
case	case	Foreach	N/A	protected	N/A	ulong	N/A
catch	catch	Get	N/A	public	public	unchecked	N/A
char	char	Goto	goto	readonly	N/A	unsafe	N/A
checked	N/A	If	if	ref	N/A	ushort	N/A
class	class	Implicit	N/A	return	return	using	import
const	const	In	N/A	sbyte	byte	value	N/A
continue	continue	Int	int	sealed	final	virtual	N/A
decimal	N/A	Interface	interface	set	N/A	void	void
default	default	Internal	protected	short	short	volatile	volatile
delegate	N/A	Is	instanceof	sizeof	N/A	while	while
do	do	Lock	synchronized	stackalloc	N/A	:	extends
double	double	Long	long	static	static	:	implements
else	else	namespace	package	string	N/A	N/A	strictfp
enum	N/A	New	new	struct	N/A	N/A	throws
event	N/A	Null	null	switch	switch	N/A	transient

Cú pháp cơ bản
Kiểu dữ liệu

- Kiểu dữ liệu nguyên thủy: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`
- Kiểu dữ liệu không nguyên thủy:



Cú pháp cơ bản

Kiểu dữ liệu nguyên thủy

- Kích thước kiểu dữ liệu nguyên thủy và giá trị mặc định

KDL	Kích thước	Khoảng giá trị	Giá trị mặc định
byte	1 byte	-128 .. 127	0
short	2 bytes	-32,768 ... 32,767	0
int	4 bytes	-2,147,483,648 ... 2,147,483,647	0
long	8 bytes	-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807	0L
float	4 bytes	$3.40282347 \times 10^{38} \dots 1.40239846 \times 10^{-45}$	0.0f
double	8 bytes	$1.7976931348623157 \times 10^{308} \dots 4.9406564584124654 \times 10^{-324}$	0.0d
char	2 bytes	Lưu ký tự	\u0000
boolean	1 bit	true/false	false

```

byte by = 124;
short s = 12345;
int i = 12345678;
long l = 12345678910L;
float f = 123456.235678f;
double d = 123456.235d;
char c = 'c';
boolean b = true;

```


1- String

String: (chuỗi) là một đối tượng biểu diễn một nối tiếp của các ký tự...

- ❑ Khởi tạo chuỗi bằng String literal (quản lý bằng String pool) hoặc Sử dụng từ khóa **new** (tạo ô nhớ mới).

```
public static void main(String[] args) {
    String s1= "Lập trình Java";
    char[] ch = { 'l', 'a', 'p', 't', 'r', 'i', 'n' };
    String s2 = new String(ch);
}
```

- ❑ Chú ý khi so sánh chuỗi: **==** là so sánh object (so sánh địa chỉ object), **equals** (so sánh giá trị value)

- ❑ String là **immutable** (bất biến)

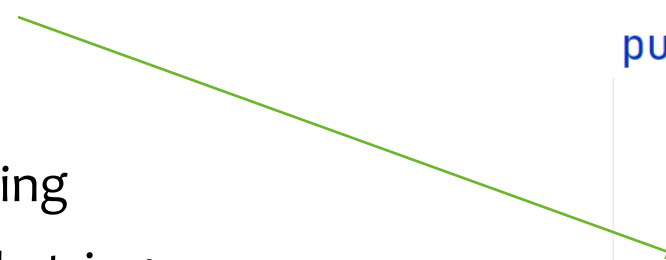
StringBuilder là Mutable

- ❑ Một số phương thức trong String

Equals, replace, split, indexOf, substring...

https://www.w3schools.com/java/java_strings.asp

```
public static void main(String[] args) {
    String s1= "Lập trình ";
    s1.concat(str: "Java");
    System.out.printf(s1);
}
```

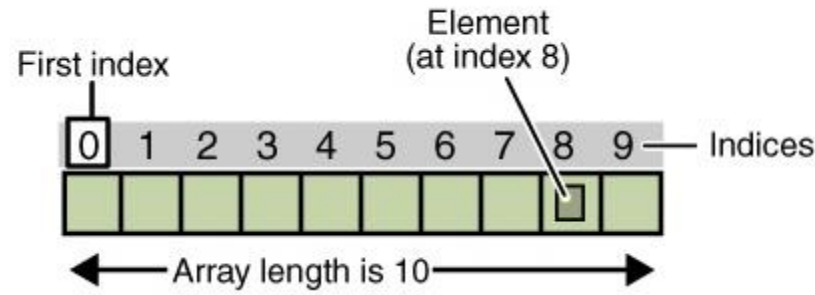


Kiểu dữ liệu không nguyên thủy

2- Array: Mảng

Mảng trong java chỉ có thể lưu trữ một tập các phần tử có số lượng phần tử cố định

- ❑ Mảng trong java lưu các phần tử theo chỉ số, chỉ số của phần tử đầu tiên là 0.



- ❑ Khai báo mảng: `KDL[] ten_mang` , hoặc `KDL ten_mang[]`

- ❑ Duyệt mảng với for hoặc foreach

- ❑ kích thước mảng: `.length`

- ❑ Mảng nhiều chiều

`KDL [][] ten_mang`

```
public static void main(String[] args) {
    String[] languages = {"C#", "Java", "Python", "JavaScript"};
    int[] primes = {2, 3, 5, 7, 11};

    for (int i = 0; i < languages.length; i++) {
        System.out.print(languages[i] + " ");
    }
    for (int element : primes)
        System.out.print(element + " ");
}
```

3- Java List

- List là một **interface** nằm trong nền tảng tập hợp của Java (Java **Collection** Framework). **List** cho phép các phần tử *trùng lặp*, và các phần tử *null*.

❑ Có đầy đủ các tính năng của một Collection.

❑ Có các phương thức: **chèn, cập nhật, xoá và tìm kiếm** một phần tử theo chỉ số.

- Khởi tạo List:

❑ **Arrays.asList(), List.of()**

List<data_type> listname = Arrays.asList(array_name);

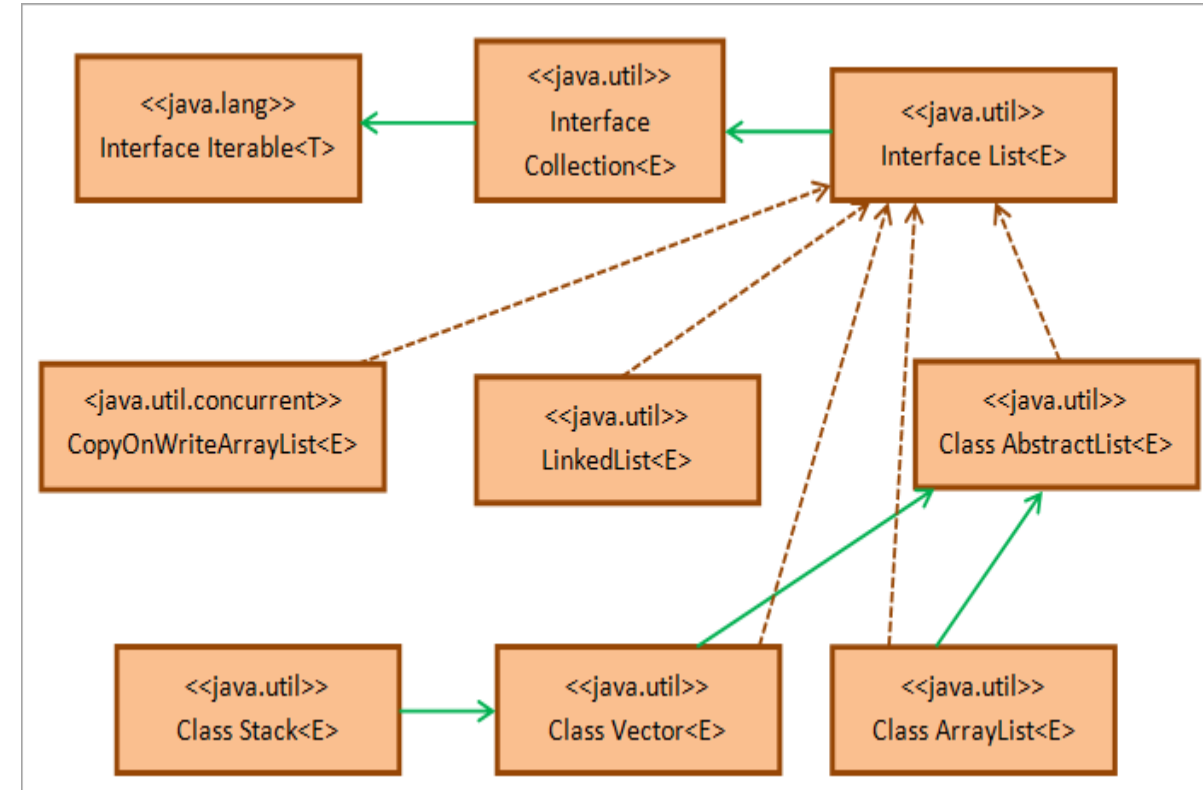
List<data_type> listname = List.of(array_name);

❑ List.add()

❑ Using Collections Class Methods: **addAll()**

❑ Java8 Streams: Stream.of().ToList()

List<Integer> nums = Stream.of(1, 2, 3).toList();



```

public static void main(String[] args) {
    List<String> languages = List.of("C#", "Java", "Python", "JavaScript");
    List<Integer> primes = Arrays.asList(0, 1, 2, 3, 4, 5);
    languages.forEach(l -> {System.out.print(l + " ");});
    for (int element : primes)
        System.out.print(element + " ");
}

```

Java Collectors

- ❑ **toList()**: Tạo danh sách
- ❑ **toSet()**: tích lũy các phần tử thành một tập hợp, loại bỏ tất cả các mục trùng lặp.
- ❑ **toCollection()**: Tạo bộ sưu tập cụ thể
- ❑ **Counting()**: Đếm phần tử
- ❑ **minBy()**: Tìm giá trị nhỏ nhất
- ❑ **maxBy()**
- ❑ **partitioningBy()**: phân chia một danh sách thành 2 danh sách. 1 danh sách thoả mãn điều kiện “true” và một danh sách không thoả mãn điều kiện
- ❑ **toUnmodifiableSet()**: Tạo tập hợp không thể thay đổi
- ❑ **joining()**: Kết hợp phần tử
- ❑ **averagingLong(), AveragegingInt(), AveragegingDouble()**
- ❑ **toMap()**
- ❑ **summingInt(), summingDouble(), summingLong ()**
- ❑ **summaryInt()**
- ❑ **groupingBy()**

Xem thêm: <https://www.javatpoint.com/java-8-collectors>

Java Iterable

- **Iterable** là một *interface* cho phép một đối tượng được lặp qua (iterated) bởi một vòng lặp.
- Duyệt các phần tử từ đầu đến cuối của một collection.

```
List<String> list = new ArrayList<String>();  
list.add("Huy Cuong");  
list.add("Thach Truc");  
list.add("Huu Trung");  
// Iterate  
Iterator<String> iterator = list.iterator();  
while (iterator.hasNext()) {  
    String element = iterator.next();  
    System.out.println(element);  
}
```

Xử lý ngoại lệ

- Một ngoại lệ (Exception) trong Java là 1 vấn đề phát sinh trong quá trình thực thi. Khi xảy ra ngoại lệ, luồng xử lý bị gián đoạn, chương trình/ứng dụng dừng bất thường. Nó là một đối tượng được ném ra tại Runtime.
- Java Exception được triển khai bằng cách sử dụng các lớp như Throwable, Exception, RuntimeException và các từ khóa như `throw`, `throws`, `try`, `catch` và `finally`.

```
try
{
    int[] myNumbers = {6, 9, 6};
    System.out.println(myNumbers[10]);
} catch (Exception e) {
    System.out.println("Something went wrong.");
} finally {
    System.out.println("The 'try catch' is finished.");
}
```

4. Java OOP

- Java là một ngôn ngữ OOP.

- ❑ Mô hình lập trình hướng đối tượng trực quan hóa mọi thứ dưới dạng đối tượng.

Một đối tượng, trong khái niệm OOP, là một thực thể trong thế giới thực có cả **trạng thái** và **hành vi**. Một lớp là một khuôn mẫu cho cùng một loại đối tượng.

Ví dụ: Lớp **Student** là một lớp đóng vai trò là khuôn mẫu cho các đối tượng sv1, sv2, sv3

- ❑ OOP dựa trên bốn khái niệm cơ bản:

Kế thừa

Trừu tượng

Đa hình

Đóng gói.

Kế thừa (Inheritance)

- Tính kế thừa: là một cơ chế trong đó một đối tượng có được tất cả các thuộc tính và hành vi của một đối tượng cha
- Trong Java, sử dụng từ khóa **extends**
- Ví dụ:

```
class SuperClass
{
    String superClassField =
"Super_Class_Field";
    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}
class SubClass extends SuperClass
{
    String subClassField = "Sub_Class_Field";
    void subClassMethod()
    {
        System.out.println("Sub_Class_Method");
    }
}
public class Main {
    public static void main(String[] args) {
        SubClass subClass = new SubClass();
        subClass.subClassMethod();

        System.out.println(subClass.subClassField);

        subClass.superClassMethod();

        System.out.println(subClass.superClassField);
    }
}
```


Trừu tượng (Abstraction)

- Tính trừu tượng trong Java là tính chất không thể hiện cụ thể mà chỉ nêu tên vấn đề. Đó là một quá trình che giấu các hoạt động bên trong và chỉ hiển thị những tính năng thiết yếu của đối tượng tới người dùng
- Trong Java được thể hiện bằng **abstract class/ interface**

Từ khóa **implements** được sử dụng bởi các lớp mà kế thừa từ Interface

- Ví dụ:

```
abstract class AbstractClass
{
    abstract void anIdea();
}

class SubClassOne extends AbstractClass
{
    @Override
    void anIdea()
    {
        System.out.println("An idea is
implemented according to SubClassOne
requirement");
    }
}

class SubClassTwo extends AbstractClass
{
    @Override
    void anIdea()
    {
        System.out.println("An idea is
implemented according to SubClassTwo
requirement");
    }
}
```

Đa hình (Polymorphism)

- Tính đa hình: Đa hình là khái niệm mà hai hoặc nhiều lớp có những **phương thức giống nhau** nhưng có thể thực thi theo những cách thức khác nhau
- Ví dụ:

```
class SuperClass
{
    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}
class SubClass extends SuperClass
{
    @Override
    void superClassMethod()
    {
        System.out.println("Super_Class_Method_Is_Overridden");
    }
}
public class Main
{
    public static void main(String[] args)
    {
        SuperClass superClass = new SuperClass();
        superClass.superClassMethod();
        superClass = new SubClass();
        superClass.superClassMethod();
    }
}
```

Đóng gói (Encapsulation)

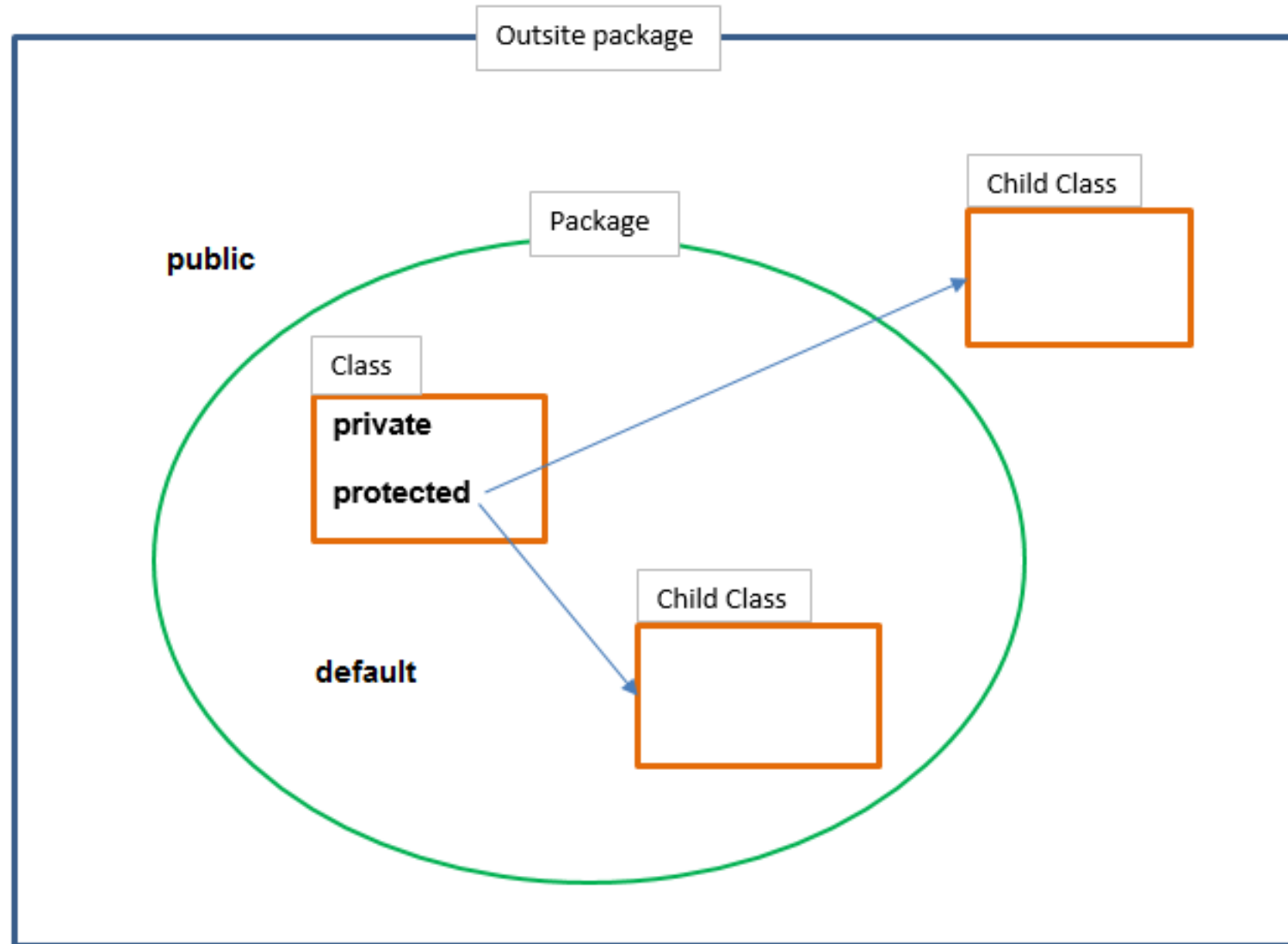
- Tính đóng gói là kỹ thuật ẩn giấu thông tin không liên quan và hiện thị ra thông liên quan. Mục đích chính của đóng gói trong java là giảm thiểu mức độ phức tạp phát triển phần mềm.
- Ví dụ: Getter/ Setter của class

```
class Customer
{
    private int custID;
    private String name;

    //Getter and setter for custID

    public int getCustID()
    {
        return custID;
    }
    public void setCustID(int custID)
    {
        this.custID = custID;
    }
    //Getter and setter for name
    public String getName() {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
}
```

ACCESS MODIFIER



NON-ACCESS MODIFIER

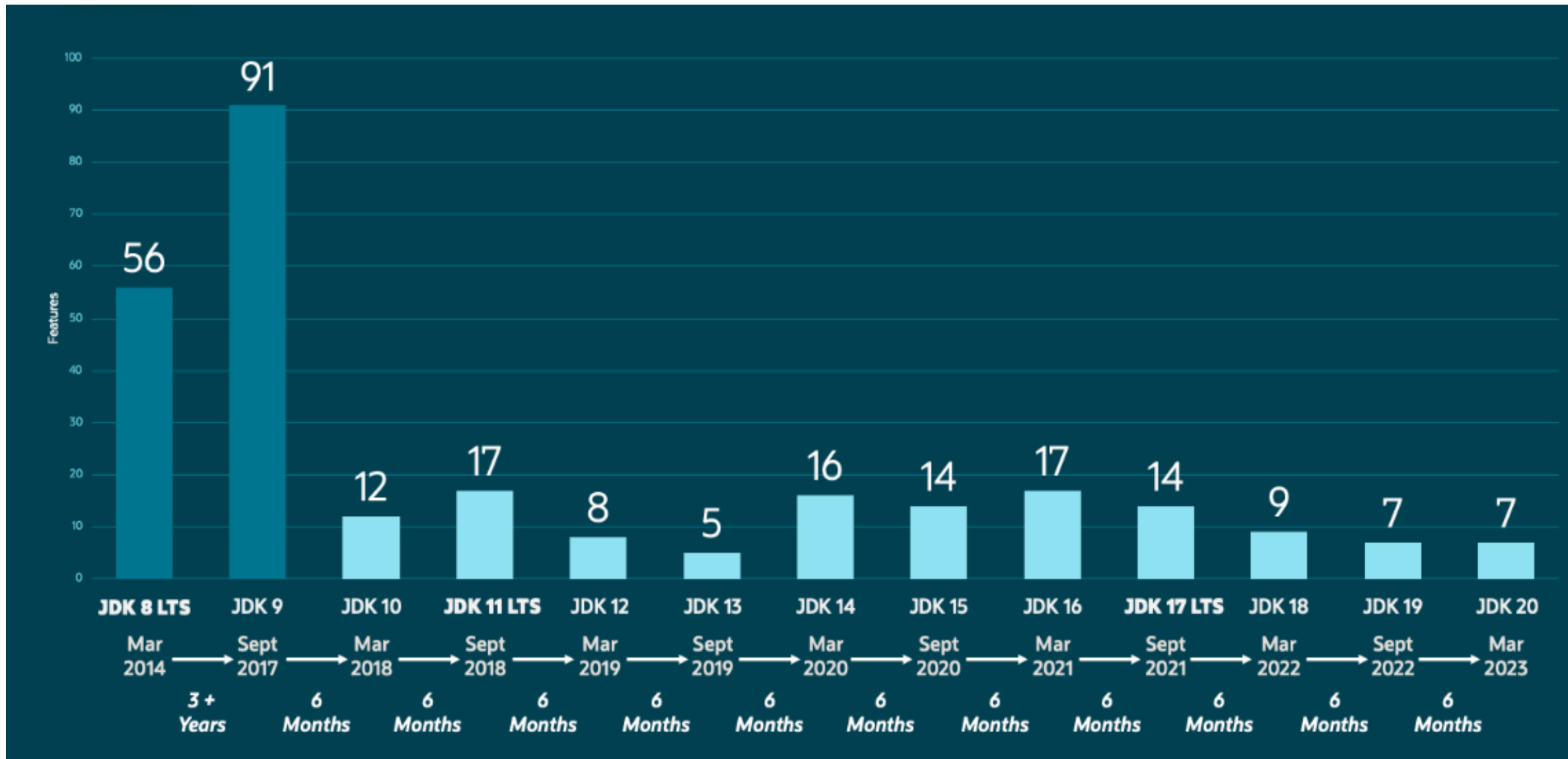
- **Static**

- ☐ Thuộc tính static của lớp: Đặc điểm của thuộc tính này là nó thuộc về **lớp** chứ không phải thuộc về từng đối tượng của lớp.
- ☐ Phương thức static của lớp:

- **Final**

- ☐ Thuộc tính của lớp có thể là một hằng số. Sử dụng từ khóa final để khai báo thuộc tính final.
- ☐ Không thể ghi đè (overriding) **phương thức final**.
Ghi đè (overriding) là TH phương thức của lớp cha được định nghĩa lại trong lớp con khi kế thừa.
- ☐ không thể kế thừa lớp final.
- ☐ hằng số liên quan đến lớp, được khai báo là **static final** để truy cập từ lớp mà không cần phải khởi tạo đối tượng.

5- Các phiên bản JDK & một số tính năng cần lưu ý



Java 8

Default And Static Methods In Interfaces

- Java 8 cung cấp thêm default method biến một **abstract method** thành **non-abstract method** với 2 mục đích:

- ❑ Cung cấp method mặc định cho interface.
- ❑ Support backward compatibility

Ví dụ:

```
public interface IStudent {
    default void AttendCourse() {
        System.out.println("Tham gia khóa học");
    }
    default void Truant() {
        System.out.println("Cúp học");
    }
}
```

```
public interface IStudent {
    no usages
    default void attendCourse() {
        System.out.println("Tham gia khóa học");
    }
    no usages
    default void truant() {
        System.out.println("Cúp học");
    }
    no usages
    private void relax() {
        System.out.println("Chơi game");
    }
}
```

- Java 9: bổ sung thêm private method cho interface

Java 8

Method References (phương thức tham chiếu)

- Method References là tính năng liên quan đến lambda expression, cung cấp cú pháp để truy cập đến các hàm khởi tạo hoặc method của class mà không cần phải khởi tạo chúng, cũng có thể sử dụng để gọi method từ các đối tượng object đã khởi tạo.
- Ví dụ:

```
public static void main(String[] args) {  
    String[] strNames = { "Huy Cuong", "Huu Trung", "Thach Truc", "Dinh Anh"};  
    Arrays.sort(strNames, String::compareToIgnoreCase);  
    Arrays.stream(strNames).forEach(System.out::println);  
}
```


Stream

- Streams là cách thức mới để xử lý tập hợp dữ liệu - Collections data bên cạnh các cách thức thông thường là vòng lặp - for, bộ lặp - iterator.

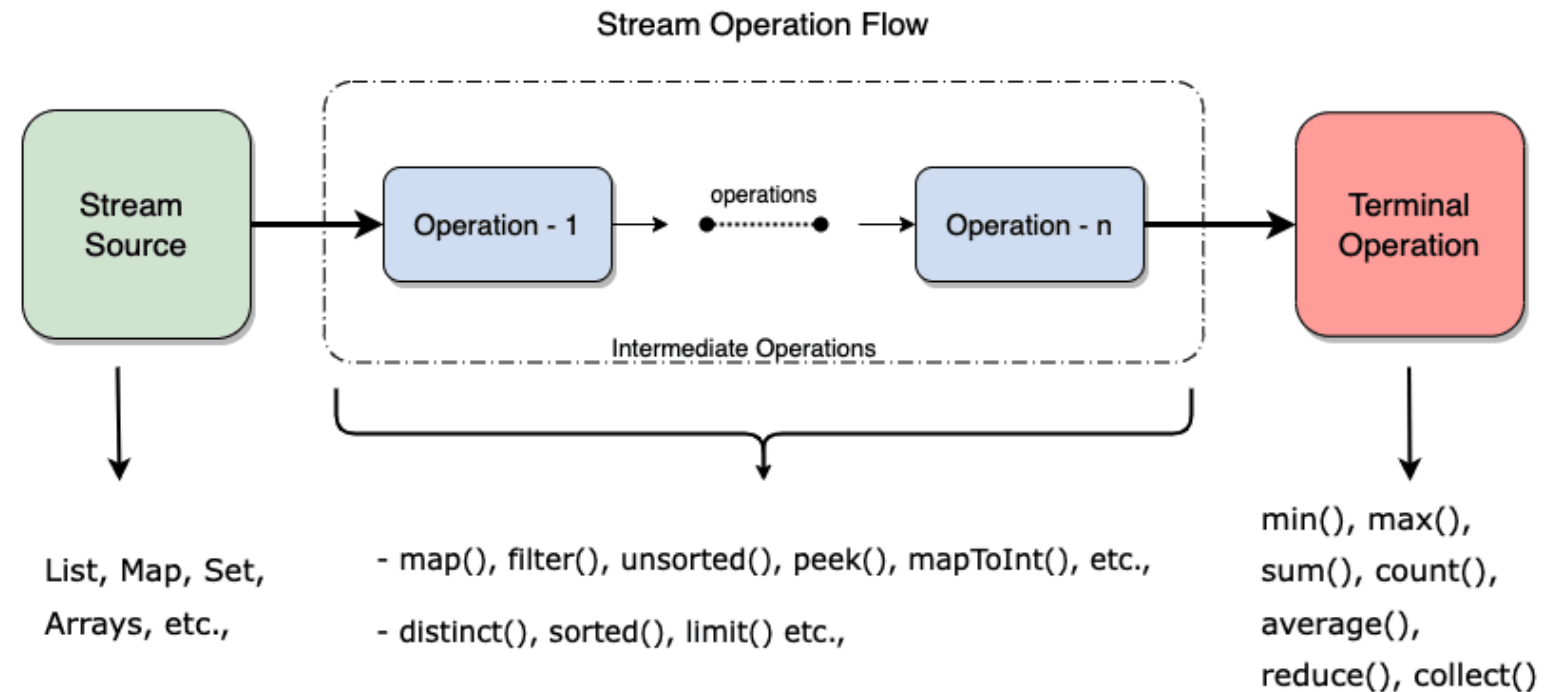
❑ Stream không làm thay

đổi dữ liệu gốc mà chỉ trả

kết quả thông qua các methods. Method của stream 2 loại:

trung gian

và hoạt động đầu cuối.



- Ví dụ: Tính tổng của các phần tử trong mảng?


```

List<Integer> integers = Arrays.asList(1,2,3,4);
int sum = integers.stream()
    .mapToInt(Integer::intValue)
    .sum();
System.out.printf("Tong:" + sum);
      
```

Java 8

Lambda Expressions & for each method

- Lambda Expression (biểu thức Lambda): là một **hàm** ẩn danh, cho phép người dùng chuyển các phương thức làm đối số.

- **Cú pháp:**

(argument-list) -> {body}.

- **Ví dụ:**

```
List<String> names = List.of("Trung", "Truc", "Anh", "Cuong");  
// List<String> sortNames = names.stream().sorted().toList();
```

```
List<String> sortNames = names.stream()  
    .sorted((o1,o2)-> o1.compareTo(o2))  
    .collect(Collectors.toList());  
sortNames.forEach(n -> System.out.println(n));
```

- **Ứng dụng:**

- ☐ Kết hợp với vòng lặp **forEach**
- ☐ Function interface

Optional class

- Optional là 1 đối tượng **generic**, bản chất của Optional là một container (bao chứa) chứa đối tượng mà nó reference tới. Nó có thể rỗng hoặc chứa giá trị NULL
 - ❑ OptionalInt
 - ❑ OptionalDouble
 - ❑ OptionalLong
- Tạo 1 Option rỗng:
`Optional<Book> optionalBook = Optional.empty();`

Java 10

var

- **var** có tác dụng rút gọn việc chỉ định kiểu dữ liệu khi khai báo biến trong Java.
 - ❑ Trước java 10: sử dụng **var** thông qua thư viện **Lombok**.
Lombok còn cung cấp **val**: tương đương với **final var**
 - ❑ Từ java 10 sử dụng biến ngắn gọn với **var**
 - ❑ Java 11: **var** có thể được sử dụng với **lambda** parameter.
- Lưu ý: khi dùng var để khai báo biến thuộc kiểu long và kiểu float => phải có hậu tố

```
var num = 6996L;
```

```
var pi = 3.14159265358f;
```

Java 11

String API improvement

● Java 11 cung cấp 1 số hàm trên string

- ❑ **isBlank()**: check một string có blank
- ❑ **repeat()**: thực hiện nối chuỗi với chính nó theo số lần chỉ định
- ❑ **strip()**: Bỏ tất cả khoảng trắng ở đầu và cuối string, giống trim()
- ❑ **lines()**: Chia string thành stream of lines phân cách nhau bằng \n

● Java 12

- ❑ **indent()**: chuỗi ban đầu được split thành nhiều line giống method lines(), sau đó mỗi line được indent (thêm khoảng trắng) dựa trên giá trị truyền vào
- ❑ **transform()**: apply function lên string hiện tại để tạo ra một kết quả mới

```
var result = "Hello".transform(input -> input + " world");
```

```
System.out.println(result); // Hello world
```

Java 14

Switch expression

- Từ Java 14: Cú pháp mới sử dụng toán tử `->` thay vì dấu hai chấm.
Ngoài ra, không có break: Biểu thức chuyển đổi không rơi vào trường hợp.
Các case được phân cách bằng dấu phẩy.
- Ví dụ: Xét tháng hiện tại là mùa nào trong năm ?

```
int month = LocalDate.now().getMonth().getValue();
switch (month) {
    case 1,2,3 -> System.out.println("Mùa Xuân\n");
    case 4,5,6 -> System.out.println("Mùa Hạ \n");
    case 7,8,9 -> System.out.println("Mùa Thu\n");
    case 10,11,12 -> System.out.println("Mùa Đông\n");
    default -> System.out.println("Tháng không hợp lệ\n");
}
```

Text Block

- Để sử dụng text Block dùng 3 dấu nháy “.
 - ❑ Sử dụng phương thức **formatted()** để thay thế tham số format
- Ví dụ:

```
String myMessage = """  
    Dòng 1  
    Dòng 2.  
    Dòng 3""";  
System.out.printf(myMessage);
```

```
public static void main(String[] args) {  
    String msg = """  
        Sáng thứ %s  
        Chiều thứ %s  
        Đêm thứ %s  
        """.formatted(...args: "Hai", "Ba", "Tư");  
    System.out.printf(msg);  
}
```

Instanceof

- Trước Java 16: sau khi kiểm tra `instanceof` thì còn ép kiểu.

```
if(s instanceof Student) {  
    Student p = (Student)s;  
    System.out.printf(p.getHoten());  
}
```

- Java 16: cung cấp pattern matching cho `instanceof` với cách sử dụng mới như sau

```
if(s instanceof Student p) {  
    System.out.printf(p.getHoten());  
}
```


Java 16

Record type

- Java 16: **record** ngang hàng với **class** hoặc **interface**

- Cách tạo record: Giống tạo hàm

```
public record Student(String id, String name, float dtb, String
faculty){
}
```

- Để tạo 1 đối tượng record

```
public static void main(String[] args) {
    Student s = new Student("20112023", "ABC", 3.2f, "CNTT");
    System.out.println(s.id());
    System.out.println(s.faculty());
    System.out.println(s.name);
}
```

- **Các phương thức getter:** giống với tên trường không có tiền tố POJO/JavaBean thông thường, tức là id(), name(), faculty(), dtb().

Sealed Class

- Java 17: Sử dụng keyword **sealed** để thông báo đây là một class/interface giới hạn, chỉ được phép extend bởi 1 class.
 - ❑ Nếu muốn nhiều hơn 1 class, bằng cách sử dụng từ khóa **sealed** và **permits** trong class

```
public sealed class Animal permits Dog, Duck {  
}  
public final class Dog extends Animal {  
}  
public final class Duck extends Animal {  
}
```

Q&A

Thank you!