**I.Primitive and Object data type**

1. Phân biệt kiểu dữ liệu nguyên thủy và kiểu dữ liệu object

Kiểu dữ liệu nguyên thủy (Primitive types)

-Lưu trữ trực tiếp giá trị trong bộ nhớ.

- Không có thuộc tính hoặc phương thức đính kèm.

- Khi truyền vào hàm, dữ liệu được truyền bằng giá trị (pass by value), tức là tạo ra một bản sao độc lập.

- Chiếm ít bộ nhớ hơn và có hiệu suất xử lý nhanh.

Kiểu dữ liệu đối tượng (Object types)

Kiểu đối tượng là những kiểu dữ liệu phức tạp, có thể chứa nhiều giá trị và chức năng bên trong.

-Lưu trữ tham chiếu đến địa chỉ của đối tượng trong bộ nhớ (chứ không trực tiếp lưu giá trị).

-Có thể chứa thuộc tính và phương thức.

-Khi truyền vào hàm, dữ liệu được truyền bằng tham chiếu (pass by reference), nghĩa là hàm có thể thay đổi giá trị của đối tượng ban đầu.

-Tốn nhiều bộ nhớ hơn do cần lưu thông tin về cấu trúc của đối tượng.

Như vậy điểm khác biệt giữa 2 kiểu dữ liệu đối tượng và nguyên thủy đó là 1 thằng có tính tham chiếu và 1 thằng có tính tham trị và khi sử dụng chúng nó sẽ khác nhau về việc cấp phát địa chỉ ô nhớ và các refernces vào địa chỉ ô nhớ đó.

2. Có thể chuyển đổi giữa hai kiểu dữ liệu này không ?

Việc chuyển đổi giữa 2 kiểu dữ liệu này là hoàn toàn khả thi.

• Boxing: Chuyển kiểu nguyên thủy thành đối tượng tương ứng (gói giá trị vào object).

• Unboxing: Lấy giá trị từ object và chuyển thành kiểu nguyên thủy.

Ví dụ :

// Boxing: Chuyển int thành Integer(Object)

int a = 5;

Integer obj = Integer.valueOf(a); // Boxing thủ công

// Autoboxing (Java tự động chuyển đổi) Integer obj2 = a;

// Unboxing: Lấy giá trị từ object Integer

int b = obj.intValue(); // Unboxing thủ công

// Auto-unboxing int c = obj2;

3. Có thể so sánh hai kiểu dữ liệu này với nhau không?

Có thể so sánh 2 kiểu dữ liệu này , cụ thể :

• Nguyên thủy được so sánh bằng toán tử == , != , < , >

Tại vì nguyên thuỷ thì lưu giá trị trên cùng 1 vùng bộ nhớ , tức là nếu có gía trị trước đó được lưu vào vùng bộ nhớ này thì khi có giá trị mới cần so sánh , nó sẽ tìm đến vùng bộ nhớ này để xem đã tồn tại giá trị được so sánh hay chưa .

• Object so sánh bằng tham chiếu. Nếu muốn so sánh nội dung, bạn phải dùng phương thức .equals().

Ví dụ :

// So sánh nguyên thủy

int a = 5; int b = 5;

System.out.println(a == b); // true, vì giá trị bằng nhau

// So sánh object (String)

String str1 = new String("Hello");

String str2 = new String("Hello");

System.out.println(str1 == str2); // false, vì khác tham chiếu

System.out.println(str1.equals(str2)); // true, vì cùng nội dung

4. Giá trị khi khởi tạo biến với hai loại kiểu dữ liệu này là gì?

• Kiểu nguyên thủy: Khi khai báo, biến sẽ được gán giá trị mặc định tùy thuộc vào loại dữ liệu.

• Object: Khi khai báo, biến object sẽ có giá trị mặc định là null nếu chưa được khởi tạo cụ thể.

**II.String**

***I.Tìm hiểu về các đặc điểm và tính chất của String trong java***

-Một đối tượng String là chuỗi ký tự dùng để lưu dữ các ký tự theo một dãy có thứ tự, số ký tự lưu được không hạn chế, có thể biểu diễn chuỗi trên một dòng với các ký tự trong dấu nháy kép ""

-Tính chất của String

+Tính bất biến :

Một khi một đối tượng String được tạo, nó không thể thay đổi nội dung.

Khi thực hiện các thao tác như nối chuỗi hoặc thay thế, Java tạo ra một đối tượng mới, thay vì thay đổi đối tượng hiện có.

ví dụ :

String str1 = "Hello";

String str2 = str1.concat(" World"); // Tạo ra chuỗi mới "Hello World"

System.out.println(str1); // Kết quả: Hello

System.out.println(str2); // Kết quả: Hello World

+String là 1 đối tượng :

String không phải là kiểu nguyên thủy, mà là một lớp trong Java (java.lang.String).

Khi khai báo String, đối tượng này được lưu trong String Pool để tối ưu bộ nhớ.

ví dụ :

String str1 = "Java"; // Lưu trong String Pool

String str2 = "Java"; // Sử dụng lại đối tượng trong String Pool

System.out.println(str1 == str2); // true, cùng tham chiếu

+String Pool trong Java:

Java sử dụng String Pool để tối ưu bộ nhớ, nơi các chuỗi có nội dung giống nhau sẽ dùng chung một tham chiếu.

Nếu bạn tạo chuỗi bằng từ khóa new, đối tượng sẽ được lưu trong heap memory, không sử dụng String Pool.

ví dụ :

String str1 = "Java"; // Lưu trong String Pool

String str2 = new String("Java"); // Tạo đối tượng mới trong heap

System.out.println(str1 == str2); // false, khác tham chiếu

+Các phương thức phổ biến trong String

length(): Trả về độ dài chuỗi.

charAt(int index): Lấy ký tự tại vị trí chỉ định.

substring(int beginIndex, int endIndex): Lấy chuỗi con.

indexOf(String s): Tìm vị trí đầu tiên của chuỗi con.

equals(Object obj): So sánh nội dung chuỗi.

equalsIgnoreCase(String anotherString): So sánh không phân biệt chữ hoa/thường.

toUpperCase() / toLowerCase(): Chuyển chuỗi thành chữ hoa/thường.

trim(): Loại bỏ khoảng trắng ở đầu và cuối chuỗi

+Nối chuỗi bằng +

+Hỗ trợ so sánh (equals, compareTo)

+Hỗ trợ Regex

+StringBuilder / StringBuffer

***II.Có bao nhiêu cách để tạo 1 biến String***

Có 2 cách để tạo đối tượng String:

Sử dụng string literal

Sử dụng từ khóa new

1. Sử dụng String Literal

String literal được tạo ra bằng cách sử dụng 2 dấu nháy kép. Ví dụ:

String s = "welcome";

2. Sử dụng từ khóa new

String s=new String("Welcome");

***III. StringPool***

***1. String pool là gì ?***

String pool là một vùng nhớ đặc biệt nằm trong vùng nhớ Heap (Heap memory), dùng để lưu trữ các biến được khai báo theo kiểu String.

String pool giúp tối ưu hoá việc lưu trữ và sử dụng vùng nhớ khi khai báo biến String, giúp hạn chế tình trạng tràn bộ nhớ Java Heap Space.

***IV .Làm sao để so sánh hai chuỗi trong java***

Sử dụng phương thức equals() hoặc toán tử ==

Vì các cách khia báo String khác nhau sẽ lưu vào StringPool theo nhiều cách khác nhau , vì vậy cũng có nhiều cách để so sánh 2 chuỗi

-Toán tử == thường sử dụng để so sánh gía trị được lưu cùng trong địa chỉ bộ nhớ

- Phương thức equals() thường so sánh gía trị của các object được lưu ở 2 vùng bộ nhớ khác nhau

ví dụ :

String s1 = "laptrinhtudau";

String s2 = "laptrinhtudau";

String s3 = new String("PHP");

String s4 = "Java";

String s5 = new String("laptrinhtudau");

String s6 = new String("laptrinhtudau");

System.out.println(s1.equals(s2));

System.out.println(s1.equals(s3));

System.out.println(s1.equals(s4));

System.out.println(s5==s6);

kết quả trả về :

true

false

false

false

**III.Static and final**

***I. Thế nào là static ? Phương thức, thuộc tính khai báo bằng từ khóa static được sử dụng khi nào ? Làm thế nào để truy cập được tới phương thức, thuộc tính static***

***1. Khái niệm static trong Java***

static là một từ khóa trong Java, được dùng để định nghĩa các thuộc tính (fields) hoặc phương thức (methods) mà thuộc về lớp (class) chứ không thuộc về đối tượng (instance).

Điều này có nghĩa là biến hoặc phương thức được khai báo là static sẽ được chia sẻ chung giữa tất cả các đối tượng của lớp đó và không cần tạo đối tượng để truy cập.

***2. Thuộc tính và phương thức static***

***2.1. Thuộc tính static***

-Thuộc tính được khai báo là static sẽ chỉ tồn tại duy nhất một bản sao trong bộ nhớ, bất kể có bao nhiêu đối tượng của lớp đó được tạo ra.

-Không cần tạo đối tượng để truy cập thuộc tính này, mà có thể truy cập trực tiếp thông qua tên lớp.

-Thuộc tính static được dùng khi bạn muốn một giá trị chung cho tất cả các đối tượng, ví dụ:

Đếm số lượng đối tượng đã được tạo.

Một hằng số chung cho toàn bộ lớp (như Math.PI).

***2.2. Phương thức static***

-Phương thức khai báo là static có thể được gọi mà không cần tạo đối tượng của lớp.

-Phương thức static không thể truy cập các thuộc tính hoặc phương thức non-static (vì các thuộc tính và phương thức non-static chỉ có sau khi đối tượng được tạo).

-Phương thức static được dùng khi:

Phương thức không phụ thuộc vào dữ liệu của đối tượng.

Bạn cần một phương thức tiện ích (utility), chẳng hạn như các phương thức trong lớp Math (Math.sqrt(), Math.abs()).

***3. Cách truy cập thuộc tính và phương thức static***

Có 2 cách truy cập:

Truy cập thông qua tên lớp: (Cách khuyến khích dùng)

Truy cập thông qua đối tượng: (Dù được phép, nhưng không nên dùng vì thiếu rõ ràng)

***II.Thế nào là final ? Khai báo 1 biến final khác gì với static, biến khai báo bằng final có thể chỉnh sửa được không ? Nếu được cho ví dụ minh họa.***

***1. Khái niệm final trong Java***

final là một từ khóa trong Java được dùng để đánh dấu một biến, phương thức hoặc lớp không thể bị thay đổi hoặc ghi đè. Khi một đối tượng, phương thức, hoặc lớp được khai báo với từ khóa final, nó sẽ không thể thay đổi sau khi đã được định nghĩa.

***2. Khai báo 1 biến final khác gì với static, biến khai báo bằng final có thể chỉnh sửa được không ?***

-Biến static có thể thay đổi giá trị. Sau khi gán giá trị mới cho nó, ta vẫn có thể truy cập và thấy giá trị mới.

-Biến final được khởi tạo thông qua constructor và không thể thay đổi sau khi đã được gán, cho dù ta có nhiều đối tượng khác nhau.

-Nếu biến là kiểu tham chiếu (reference type), nó không thể trỏ đến một đối tượng khác, nhưng nội dung của đối tượng vẫn có thể thay đổi (nếu đối tượng đó là mutable).

ví dụ :

public static void main(String[] args) {

final int x = 10; // Biến final

// x = 20; // Lỗi: Không thể gán lại giá trị cho x

final StringBuilder sb = new StringBuilder("Hello");

sb.append(" World"); // Có thể thay đổi nội dung của đối tượng

System.out.println(sb); // Kết quả: Hello World

// sb = new StringBuilder("New"); // Lỗi: Không thể gán lại cho sb

}

***IV.OOP***

***1.Nêu ra các tính chất quan trọng của hướng đối tượng***

***Lập trình hướng đối tượng gồm 4 tính chất gồm***

-Tính đóng gói :Tức là trạng thái của đối tượng được bảo vệ không cho các truy cập từ code bên ngoài như thay đổi trong thái hay nhìn trực tiếp ,tính đóng gói được thể hiện thông qua phạm vi truy cập (access modifier)

-Tính kế thừa :là khả năng cho phép ta xây dựng một lớp mới dựa trên các định nghĩa của một lớp đã có. Lớp đã có gọi là lớp Cha, lớp mới phát sinh gọi là lớp Con và đương nhiên kế thừa tất cả các thành phần của lớp Cha, có thể chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại.

-Tính đa hình :Khi một tác vụ được thực hiện theo nhiều cách khác nhau được gọi là tính đa hình.

Đối với tính chất này, nó được thể hiện rõ nhất qua việc gọi phương thức của đối tượng. Các phương thức hoàn toàn có thể giống nhau, nhưng việc xử lý luồng có thể khác nhau

-Tính trừu tượng :là một tiến trình ẩn các chi tiết trình triển khai và chỉ hiển thị tính năng tới người dùng. Tính trừu tượng cho phép bạn loại bỏ tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.

***2. Access modifier trong java có những loại nào ? Nêu đặc điểm của từng loại***

Access modifier có 4 loại : public , private , protected , default

-Private Access Modifier chỉ được truy cập trong phạm vi lớp.

-Nếu bạn không khai báo modifier nào, thì nó chính là trường hợp mặc định. Default Access Modifier là chỉ được phép truy cập trong cùng package.

-Protected access modifier được truy cập bên trong package và bên ngoài package nhưng phải kế thừa.

Protected access modifier có thể được áp dụng cho biến, phương thức, constructor. Nó không thể áp dụng cho lớp.

-Public access modifier được truy cập ở mọi nơi.

***3.Phân biệt class và instance***

-Class (lớp) là một bảng mô tả hoặc kế hoạch cho một loại đối tượng. Nó định nghĩa các thuộc tính (fields) và phương thức (methods) mà các đối tượng thuộc lớp đó sẽ có.

+Lớp mô tả cấu trúc và hành vi của đối tượng.

+Không chiếm bộ nhớ khi chỉ được định nghĩa.

+Được định nghĩa bởi lập trình viên.

-Instance (thể hiện, hay đối tượng) là một bản sao cụ thể của một class. Mỗi instance có thể có giá trị khác nhau cho các thuộc tính đã định nghĩa trong class.

+Bản sao cụ thể của lớp, có thuộc tính riêng.

+Chiếm bộ nhớ cho các thuộc tính non-static.

+Được tạo ra từ class thông qua từ khóa new.

***4. Phân biệt Abstract và Interface , Nêu trường hợp sử dụng cụ thể. Nếu 2 interface hoặc 1 abstract và 1 interface có 1 function cùng tên, có thể cùng hoặc khác kiểu trả về cùng được kế thừa bởi một class, chuyện gì sẽ xảy ra?***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  | | --- | | **Abstract Class** |  |  | | --- | |  | | | **Interface** | | --- |  |  | | --- | |  | |
| |  | | --- | | **Khái niệm** |  |  | | --- | |  | | Là lớp không thể được khởi tạo và có thể chứa các phương thức có hoặc không có thân (body). | |  | | --- | | Là một tập hợp các phương thức trừu tượng mà lớp thực hiện phải cài đặt. |  |  | | --- | |  | |
| |  | | --- | | **Phương thức** |  |  | | --- | |  | | |  | | --- | | Có thể có cả phương thức trừu tượng và phương thức có thân. |  |  | | --- | |  | | |  | | --- | | Chỉ có phương thức trừu tượng (trong Java 8 và trước đó) và từ Java 8 có thể có phương thức mặc định (default method). |  |  | | --- | |  | |
| |  | | --- | | **Thuộc tính** |  |  | | --- | |  | | |  | | --- | | Có thể có thuộc tính (fields) và thuộc tính có thể là static hoặc non-static. |  |  | | --- | |  | | |  | | --- | | Không có thuộc tính, nhưng có thể định nghĩa các biến static (mặc định là public static final). |  |  | | --- | |  | |
| |  | | --- | | **Kế thừa** |  |  | | --- | |  | | |  | | --- | | Một lớp chỉ có thể kế thừa một lớp trừu tượng (không đa kế thừa). |  |  | | --- | |  | | |  | | --- | | Một lớp có thể triển khai nhiều interface (đa kế thừa). |  |  | | --- | |  | |
| |  | | --- | | **Truyền đạt hành vi** |  |  | | --- | |  | | |  | | --- | | Sử dụng khi bạn muốn chia sẻ mã giữa các lớp. |  |  | | --- | |  | | |  | | --- | | Sử dụng để định nghĩa hành vi mà không cần quan tâm đến cách cài đặt. |  |  | | --- | |  | |
| |  | | --- | | **Trường hợp sử dụng** |  |  | | --- | |  | | |  | | --- | | Khi có sự liên quan giữa các lớp (như một mối quan hệ IS-A) và có mã chung để chia sẻ. |  |  | | --- | |  | | Khi bạn cần một tập hợp các hành vi mà các lớp khác nhau có thể thực hiện mà không cần có mối quan hệ cha-con. |

-Trường hợp sử dụng cụ thể :

**Abstract Class:**

* Giả sử bạn có một ứng dụng quản lý động vật, bạn có thể tạo một lớp trừu tượng Animal với các thuộc tính và phương thức chung như eat(), sleep(), và các thuộc tính như name và age. Các lớp cụ thể như Dog, Cat có thể kế thừa từ lớp Animal và cài đặt phương thức riêng của chúng.

**Interface:**

* Nếu bạn muốn định nghĩa hành vi mà không cần liên quan đến các thuộc tính cụ thể, chẳng hạn như Drivable, bạn có thể định nghĩa một interface với phương thức drive(). Các lớp như Car, Bicycle có thể thực hiện interface này.
* ***Nếu 2 interface hoặc 1 abstract và 1 interface có 1 function cùng tên, có thể cùng hoặc khác kiểu trả về cùng được kế thừa bởi một class, chuyện gì sẽ xảy ra?***

**1. Hai Interface có phương thức trùng tên**

Nếu **hai interface** có cùng một phương thức (tên và tham số giống nhau), lớp thực hiện (implementing class) **phải cài đặt phương thức đó một lần**.

* **Nếu kiểu trả về giống nhau:** Không có vấn đề gì, lớp cài đặt sẽ định nghĩa phương thức đó.
* **Nếu kiểu trả về khác nhau:** Java **sẽ không biên dịch được**, vì nó không biết chọn kiểu trả về nào.

## ****2. Abstract Class và Interface có phương thức trùng tên****

Nếu một **abstract class** và một **interface** có cùng một phương thức:

* **Nếu phương thức có cùng kiểu trả về:** Lớp kế thừa sẽ kế thừa phương thức từ abstract class và phải cài đặt phương thức từ interface nếu nó chưa có thân.
* **Nếu phương thức có kiểu trả về khác nhau:** Lỗi biên dịch tương tự như khi kế thừa từ hai interface.

5.Thế nào là **Overriding**và **Overloading?**

-**Overloading** trong Java hay nạp chồng phương thức là kỹ thuật cho phép một lớp có nhiều phương thức cùng tên nhưng khác nhau về số lượng tham số, kiểu dữ liệu của tham số hoặc thứ tự của các tham số. Nghĩa là, khi bạn gọi một phương thức được định nghĩa, Java sẽ xác định phương thức nào sẽ được gọi dựa trên tham số được truyền vào.

- **Overriding** là một tính năng trong lập trình hướng đối tượng Java, cho phép một lớp con cung cấp một định nghĩa mới cho một phương thức đã được định nghĩa trong lớp cha của nó. Quá trình này giúp lớp con mở rộng hoặc thay đổi hành vi của lớp cha mà không làm thay đổi mã nguồn của lớp cha.

6. Một function có access modifier là private or static có thể overriding được không?

-1 function có access modifier là private thì không thể ghi đè được , vì private chỉ được truy cập trong nội bộ 1 lớp , vì vậy khi lớp con kế thừa từ lớp cha thì ko dùng được function ấy

- **Phương thức static** không thể được ghi đè. Thay vào đó, nếu định nghĩa lại trong lớp con, **Java sẽ tạo ra một phiên bản mới (shadowing)**. Điều này có nghĩa là nó không tham gia vào quá trình **đa hình** như các phương thức thông thường.

7. -Một phương thức final có thể kế thừa được không ?

-một phương thức final không thể ghi đè , nhưng nó vẫn có thể được kế thừa

8. Phân biệt hai từ khóa **This**và **Super**

-this là một tham chiếu đến đối tượng hiện tại (instance) của lớp. Nó được sử dụng để phân biệt giữa các biến instance và các tham số của phương thức hoặc constructor có cùng tên.

- super là một tham chiếu đến lớp cha của lớp hiện tại. Nó được sử dụng để truy cập các biến instance, phương thức, và constructor của lớp cha.