1. Kích thước của một Array là cố định khi khai báo, trong khi một List có thể thay đổi kích thước.
2. Các phần tử của một Array được lưu trữ liên tục trong bộ nhớ, trong khi các phần tử của một List có thể được lưu trữ ở bất kỳ vị trí nào trong bộ nhớ.
3. Việc truy cập các phần tử của một Array được thực hiện bằng cách sử dụng chỉ số (index), trong khi việc truy cập các phần tử của một List được thực hiện bằng cách sử dụng các phương thức như get(), add(), remove(), set().
   1. objects gọi là equal based on equals() method thì hash code giống nhau. Nhưng nếu hash code giống nhau 🡪 chưa chắc đã equal. Kiểu 2 cái kẹo giống nhau thì ở cùng 1 hộp nhưng nó ở ùng 1 hộp thì chưa chắc nó đã giống nhau.

public boolean addAll(Collection<? extends E> c)

|  |
| --- |
| Suppose the type of xarr is an array of XXX, and the type of yarr is an array of YYY. When is the assignment xarr = yarr; legal? A. Sometimes B. Always C. Never  A – khi XXX và YYY trùng nhau? |

|  |
| --- |
| When is x & y an int? (Choose one). A. Always B. Sometimes C. When neither x nor y is a float, a long, or a double  Đây là phép AND bit. Và nó chỉ cho ra int trong vài trường hợp thôi |

|  |
| --- |
| What are the legal types for whatsMyType? short s = 10; whatsMyType = !s;  A. short B. int C. There are no possible legal types.   * C vì s nó không phải boolen |

method-call conversion: chuyển kiểu dữ liệu tự động – khi cộng hoặc trừ 1 số double cho 1 số int thì int sẽ tự động được chuyển sang double

assignment conversion: mình phải viết cái ngoặc để ép kiểu. vd: (int)2.5

* Rule is the same

giá trị của x là -5, tương ứng với giá trị nhị phân 11111011 (trong kiểu dữ liệu byte). Khi áp dụng toán tử ~ lên x, Java sẽ tự động chuyển đổi kiểu dữ liệu của x thành kiểu int và đảo ngược các bit của giá trị đó, kết quả là giá trị 11111111 11111111 11111111 11111010 (trong kiểu dữ liệu int). Giá trị này tương ứng với số nguyên 4,294,967,290 trong hệ thập phân.

When you cast 128 (10000000 in binary) to an eight-bit byte type, the sign bit gets set to 1, so the number becomes interpreted as negative. Java uses [Two's Complement representation](http://en.wikipedia.org/wiki/Two%27s_complement), so 10000000 is -128 - the smallest negative number representable with 8 bits.

Under this interpretation, 129 becomes -127, 130 becomes -126, and so on, all the way to 255 (11111111 in binary), which becomes -1.

một số phương thức quan trọng của interface Collection:

1. boolean add(E e): Thêm một phần tử vào collection, trả về true nếu thành công.
2. boolean addAll(Collection<? extends E> c): Thêm tất cả các phần tử từ một collection khác vào collection hiện tại, trả về true nếu thành công.
3. boolean remove(Object o): Xóa một phần tử khỏi collection, trả về true nếu thành công.
4. boolean removeAll(Collection<?> c): Xóa tất cả các phần tử của một collection khác khỏi collection hiện tại, trả về true nếu thành công.
5. boolean contains(Object o): Kiểm tra xem collection có chứa một phần tử cụ thể không, trả về true nếu có.
6. boolean containsAll(Collection<?> c): Kiểm tra xem collection có chứa tất cả các phần tử của một collection khác không, trả về true nếu có.
7. int size(): Trả về số lượng phần tử trong collection.
8. boolean isEmpty(): Kiểm tra xem collection có phải là rỗng hay không, trả về true nếu collection rỗng.
9. void clear(): Xóa tất cả các phần tử trong collection.
10. Iterator<E> iterator(): Trả về một Iterator để lặp qua các phần tử trong collection.
11. Object[] toArray(): Trả về một mảng chứa các phần tử trong collection.

* . (Collection<? extends E> c): Tham số của phương thức, là một đối tượng Collection chứa các phần tử có kiểu dữ liệu tương thích với E. Trong đó, "? extends E" chỉ ra rằng kiểu dữ liệu của phần tử trong Collection c phải là kiểu dữ liệu con của E.

**Các kiểu tham số thông thường:**

1. **T - Type** (Kiểu dữ liệu bất kỳ thuộc Wrapper class: String, Integer, Long, Float, …)
2. **E – Element** (phần tử – được sử dụng phổ biến trong Collection Framework)
3. **K – Key** (khóa)
4. **V – Value** (giá trị)
5. **N – Number** (kiểu số: Integer, Double, Float, …)
6. **U,S,I,G, …** (tùy theo kiểu của người dùng đặt)

 transient được sử dụng trong Java để chỉ ra rằng một trường không nên được tuần tự hóa khi đối tượng chứa nó được tuần tự hóa.

Static như 1 cái hồ, ai muốn lấy cũng đc, ko can xin phep

Còn những phương thức khác như 1 cái giếng, muốn lấy phải xin phép.

Final dùng cho biến nghĩa là biến đấy nó sẽ kh thể thay đổi.   
Final dùng cho object nghĩa là object đấy vẫn thay đổi được nhưng reference to the object cannot be reassigned to a new object of a different type

B. The File class does not support directly changing the current directory.

The File class in Java does not provide any direct method to change the current working directory. However, you can use the `System.setProperty` method to change the user's current working directory. For example, to change the current working directory to "DirName", you can use the following code:

Sao chép mã

System.setProperty("user.dir", "/path/to/DirName");

After executing this line of code, any subsequent file operations (such as reading or writing files) will be done relative to the "DirName" directory.

A generic class can have multiple type parameters

The **`java.util.TreeSet`** class implements the **`java.util.SortedSet`** interface and extends the **`AbstractSet`** class.

A HashMap implements Map interface

The classes that implement java.util.List interface are:

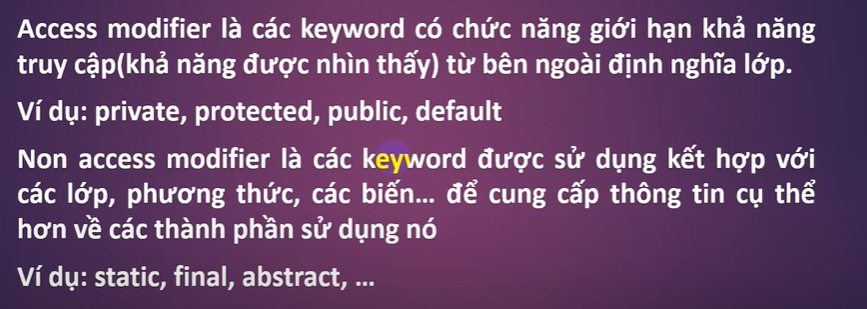
A. java.util.LinkedList  
D. java.util.ArrayList

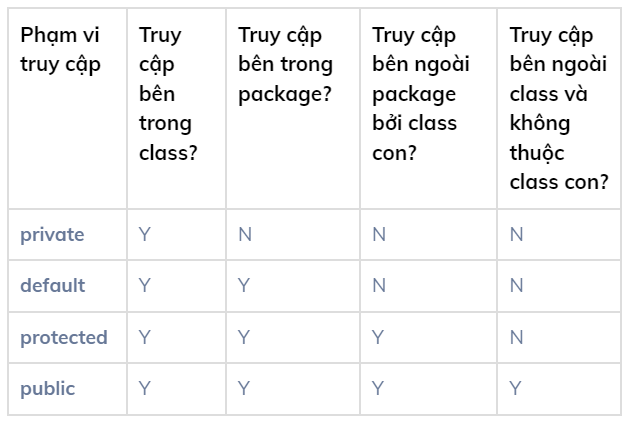
UTF characters are as big as they need to be. **Unicode characters are all 16 bits.**

|  |
| --- |
| public class Rid{  public void amethod(int i, String s){}  //Here  // public void amethod(String s, int i){}  public int amethod(int i, String s){}  public void amethod(int i, String mystring){}  public void Amethod(int i, String s){}  }  A vì đổi thứ tự  B vì thay bằng 1 chữ khác |

# Access modifier

Abstract: cho phép kế thừa chứ không cho phép tạo ra đốii tượng





Static: không cần đối tượng vẫn truy cập đc

extends is used to specify that an interface inherits from another interface

**`implements`** keyword is used to specify that a class implements one or more interfaces.

An abstract class can have both abstract and non-abstract methods.

An abstract class can contain methods with declared bodies, also known as "concrete methods". These are methods that have an implementation, rather than just a method signature. They can be used by any concrete subclasses of the abstract class, but can also be overridden by those subclasses if necessary.

// A concrete method with a declared body

public double getArea() {

return 0.0;

}

 This method returns a default value of 0.0, but can be overridden by any concrete subclass of Shape to provide a more specific implementation.

// An abstract method without a body

public abstract double getPerimeter();

Which access modifier allows you to access method calls in libraries not created in Java? 🡪 native -- http://www.java2s.com/ref/java/java-oca-ocp-practice-question-1159.html

A generic class can have only one type parameter.  
A. FALSE – một lớp thì có bao nhiêu tham số chả được. Cái ở trong ngoặc đơn là cái tham số đấy

|  |
| --- |
| Given the following method definition in a class that otherwise compiles correctly: public booleantestAns(String ans, int n){ booleanreslt; if (ans.equalsIgnoreCase("YES") && n > 5) rslt = true; return rslt; }  What will be the result of trying to compile the class and execute the testAns method with inputs of "no" and 5? A. A compiler error will prevent compilation. B. A result of true will be returned. C. A result of false will be returned. D. A runtime exception will be thrown in the testAns method.  A vì nó chưa khai báo booleanreslt = true hay false |

"Enumerating" và "iterating" là hai khái niệm khác nhau trong lập trình.

"Iterating" thường được sử dụng để chỉ quá trình lặp lại các phần tử trong một tập hợp (ví dụ: danh sách, mảng, hoặc chuỗi) bằng cách sử dụng một vòng lặp (ví dụ: for, while) để truy cập từng phần tử một và thực hiện các thao tác trên đó.

"Enumerating" thường được sử dụng để đánh số các phần tử trong một tập hợp bằng cách cung cấp một chỉ mục cho từng phần tử. Điều này thường được thực hiện bằng cách sử dụng một hàm đặc biệt (ví dụ: enumerate trong Python) để lặp qua các phần tử và trả về một cặp giá trị (chỉ mục, giá trị) cho mỗi phần tử.

Vì vậy, trong khi "iterating" là quá trình lặp lại các phần tử trong một tập hợp, "enumerating" là quá trình đánh số các phần tử trong tập hợp đó để có thể truy cập chúng bằng chỉ mục.

The keyword used to prevent an object from being serialized in Java is D) transient.

When an object is marked as transient, its state will not be saved during the serialization process. This is useful when there are certain fields in an object that should not be serialized, such as fields that contain sensitive data or fields that are not relevant to the object's state.

Có lẽ việc serialize này nó sẽ tạo ra những cái thông tin thừa lưu trong bộ nhớ, giống như mình lưu các phiên bản của 1 file ở trong máy ấy

Calling notify() on an object wakes up one of the threads that is waiting on that object, but the thread that is woken up will only start running when it is given a CPU time slice by the scheduler.

1. package mail; 2.

interface Box {

protected void open();

void close();

public void empty();

}

* Nó lỗi ở dòng 4 vì cái dòng 4 là default. Và thứ tự là: public > protected > default > private

Trong java, rác (garbage) có nghĩa là các đối tượng không còn được tham chiếu nữa.

 System.gc() là để thu gom rác, vậy đó

System.gc() method can be called to suggest to the JVM that it should run the garbage collector, it does not actually force the garbage collector to run immediately or at all.

Float = 32 bit

The rules of binary numeric promotion are as follows:

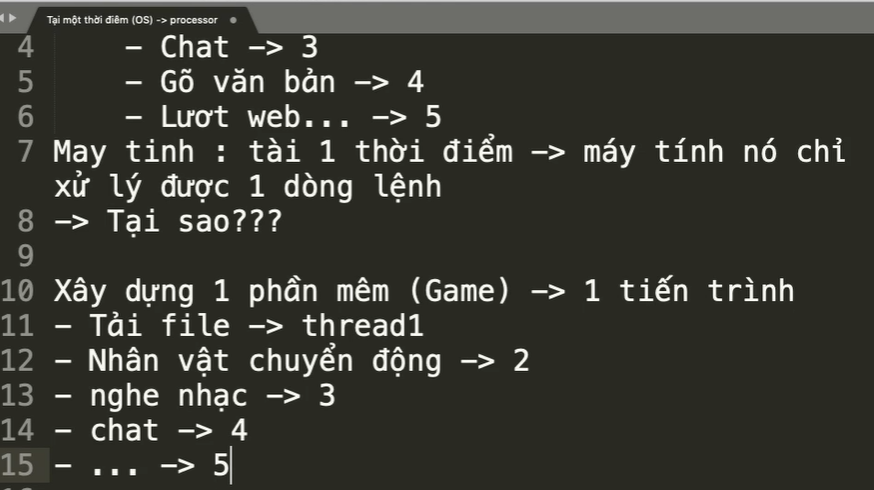
1. If either operand is of type double, the other operand is converted to double.
2. Otherwise, if either operand is of type float, the other operand is converted to float.
3. Otherwise, if either operand is of type long, the other operand is converted to long.
4. Otherwise, both operands are converted to int

StringBuffer và StringBuilder

| **STT** | **StringBuffer** | **StringBuilder** |
| --- | --- | --- |
| 1 | StringBuffer là đồng bộ (synchronized) tức là luồng an toàn. Điều này có nghĩa là không thể có 2 luồng cùng truy cập phương thức của lớp StringBuffer đồng thời. | StringBuilder là không đồng bộ (non-synchronized) tức là luồng không an toàn. Điều này có nghĩa là có 2 luồng cùng truy cập phương thức của lớp StringBuilder đồng thời. |
| 2 | StringBuffer không hiệu quả bằng StringBuilder. | StringBuilder hiệu quả hơn StringBuffer. |

Buffer thi cham hon nhung an toan hon

# Phần 16 | Tìm hiểu Thread - Phân 1 | Khóa học lập trình Java nâng cao



Nó là về đa luồng

1 byte = 8 bit

* 1. int = 4 byte
* In Java, == and .equals() are two different ways of comparing objects.
* The == operator checks whether two objects are the same object in memory, i.e., they have the same reference. This means that if two variables are pointing to the same object in memory, the == operator will return true.
* On the other hand, the .equals() method checks whether two objects have the same value. This means that if two objects have the same value, regardless of whether they are the same object in memory or not, the .equals() method will return true.

== là địa chỉ

Equalsto là giá trị

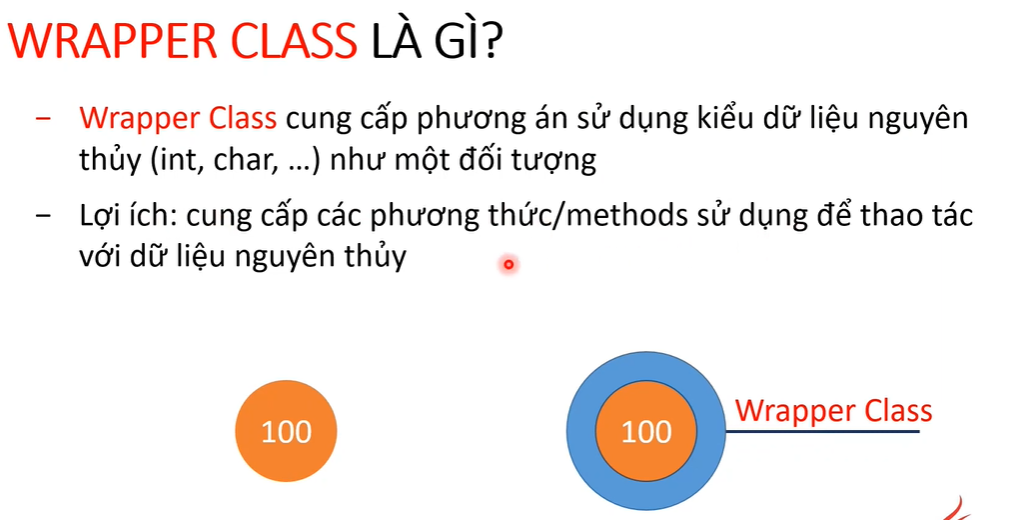
Các đối số (arguments) có kiểu dữ liệu nguyên thủy (primitive type) trong Java được truyền theo giá trị (pass by value).

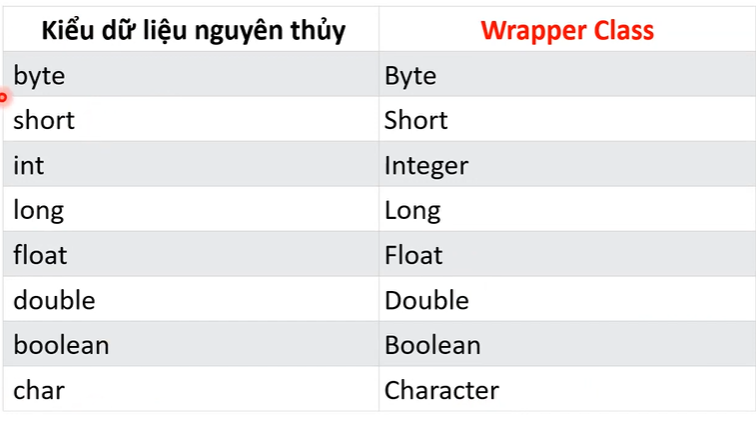
Trong truyền tham số theo giá trị, giá trị của biến được sao chép và truyền cho phương thức. Phương thức sử dụng bản sao của giá trị này để thực hiện các tính toán và thay đổi giá trị bản sao này sẽ không ảnh hưởng đến giá trị của biến gốc.

Ví dụ, nếu chúng ta truyền một biến kiểu int có giá trị là 5 cho một phương thức và phương thức thay đổi giá trị của biến này thành 10, giá trị của biến gốc sẽ không thay đổi và vẫn là 5.

Tuy nhiên, nếu đối số được truyền vào là một đối tượng (object), thì việc truyền đối số sẽ là theo tham chiếu (pass by reference), nghĩa là đối tượng được truyền vào phương thức là chính đối tượng gốc, không phải là một bản sao của đối tượng.

# [Lập trình Java] Hướng dẫn học cho người mới bắt đầu - WRAPPER CLASS





change the value that is encapsulated by a wrapper class after you have instan- tiated it thì dùng set() method thôi chứ không phải là setXXX

# Java Cơ Bản - Serializable Object trong Java - JMaster.io

Serialization và deserialization là quá trình chuyển đổi một đối tượng Java thành một chuỗi byte và chuyển đổi chuỗi byte đó thành một đối tượng Java tương ứng. Đây là một phương pháp để lưu trữ đối tượng Java trên đĩa hoặc truyền qua mạng.

Trong Java, để cho một đối tượng có thể được serialize và deserialize, lớp đó phải implements interface Serializable. Interface Serializable không có bất kỳ phương thức nào, nó chỉ là một đánh dấu cho trình biên dịch biết rằng đối tượng của lớp đó có thể được serialize.

Để serialize một đối tượng, ta sử dụng một đối tượng ObjectOutputStream. Đối tượng này sẽ ghi đối tượng vào một OutputStream, chẳng hạn như FileOutputStream. Ví dụ:

```

try {

FileOutputStream fileOut = new FileOutputStream("employee.ser");

ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(employee);

out.close();

fileOut.close();

} catch (IOException i) {

i.printStackTrace();

}

```

Để deserialize đối tượng, ta sử dụng một đối tượng ObjectInputStream. Đối tượng này sẽ đọc chuỗi byte từ InputStream, chẳng hạn như FileInputStream, và chuyển đổi nó thành đối tượng tương ứng. Ví dụ:

```

try {

FileInputStream fileIn = new FileInputStream("employee.ser");

ObjectInputStream in = new ObjectInputStream(fileIn);

Employee employee = (Employee) in.readObject();

in.close();

fileIn.close();

} catch (IOException i) {

i.printStackTrace();

} catch (ClassNotFoundException c) {

c.printStackTrace();

}

```

Lưu ý rằng các lớp được serialize và deserialize phải có cùng serialVersionUID để đảm bảo tính nhất quán giữa các phiên bản của lớp. Nếu serialVersionUID khác nhau, việc deserialize đối tượng có thể gây ra lỗi.

Trong Java, private readObject là một phương thức được sử dụng trong quá trình deserialize một đối tượng. Phương thức này cho phép lớp định nghĩa một cách chi tiết hơn việc deserialize các trường dữ liệu riêng tư (private fields) của đối tượng.

Khi một đối tượng được deserialize, phương thức readObject sẽ được gọi tự động để đọc các trường dữ liệu của đối tượng từ chuỗi byte được đọc từ file hoặc stream. Tuy nhiên, nếu một trường dữ liệu là private, phương thức readObject mặc định sẽ không có quyền truy cập vào trường này. Việc sử dụng phương thức private readObject cho phép lớp định nghĩa một cách chi tiết hơn quyền truy cập vào các trường dữ liệu riêng tư, bằng cách xác định cách thức đọc và ghi các trường dữ liệu riêng tư đó.

* + Nó là private

Phương thức private readObject phải được khai báo là private và có tham số đầu vào là ObjectInputStream. Nội dung của phương thức này phải được định nghĩa bởi lập trình viên để đọc các trường dữ liệu riêng tư của đối tượng được serialize. Tương tự, lớp cũng có thể sử dụng phương thức private writeObject để định nghĩa cách thức ghi các trường dữ liệu riêng tư của đối tượng vào chuỗi byte để serialize.

Lưu ý rằng phương thức private readObject và private writeObject chỉ được sử dụng trong trường hợp đặc biệt khi cần phải định nghĩa một cách chi tiết hơn cách thức serialize và deserialize một đối tượng. Việc sử dụng phương thức này có thể gây khó khăn trong việc bảo trì và mở rộng mã nguồn trong tương lai, vì vậy cần phải cân nhắc kỹ trước khi sử dụng.

Identifiers in Java are the names given to classes, methods, variables, and labels. They must follow certain rules, including:

* They must start with a letter, underscore (\_), or dollar sign ($).
* They cannot start with a number.
* They can contain letters, numbers, underscores, and dollar signs.
* They cannot contain spaces or special characters, except for underscores and dollar signs.
* They cannot be a reserved keyword.

Which of the following signatures are valid for the main() method entry point of an application?

A. public static void main()  
  
B. public static void main(String arg[])  
  
C. public void main(String [] arg)  
  
D. public static void main(String[] args)  
  
E. public static int main(String [] arg)

**Answer:**B and D are both acceptable – giờ mới biết <https://www.java-samples.com/showtutorial.php?tutorialid=699>

What is the range of values that can be assigned to a variable of type short?

Vì short là 16 bit. Nên mỗi cái là 2^15 và 2^15-1

1 short = 2 byte

Suppose a source file contains a large number of import statements.

How do the imports affect the time required to compile the source file?

* A. Compilation takes no additional time.
* [**B. Compilation takes slightly more time.**](http://www.java2s.com/ref/java/java-oca-ocp-practice-question-1001.html#:~:text=Suppose%20a%20source%20file%20contains,Compilation%20takes%20no%20additional%20time.)
* C. Compilation takes significantly more time.

Tại sao import nhiều mà lại take slightly more?

When a Java source file contains a large number of import statements, the compiler needs to search for and load the classes or packages specified in these statements. This can take some additional time, especially if the imported classes or packages are located in different directories or JAR files.

However, the impact of import statements on compilation time is generally not significant unless the number of imports is extremely large. In most cases, the additional time required to compile a source file with a large number of import statements is negligible compared to the total compilation time.

Therefore, the correct answer is B. "Compilation takes slightly more time."

A. import java.util.Vector;  
B. static import java.util.Vector.;C. import static java.util.Vector.;  
D. import java.util.Vector static;

* Option A is a valid import statement that imports the java.util.Vector class.
* Option B is an invalid import statement because the \* wildcard cannot be used with static imports.
* Option C is a valid static import statement that imports all the static members in the java.util.Vector class.
* Option D is an invalid import statement because the static keyword cannot be used as an identifier in an import statement.

The following options may be statically imported:

B. Static method names  
C. Static field names

Explanation:

In Java, static members, including static fields and methods, can be imported statically into another class or interface using the import static statement. This allows the static members to be used without having to qualify them with the name of the class or interface.

In contrast, package names and method-local variable names cannot be statically imported. Package names are used to organize classes and interfaces into hierarchical namespaces, but they are not themselves members that can be statically imported. Method-local variables are local to a method and are not accessible outside the method's scope.

Hexadecimal literals are prefixed with 0x or 0X, and octal literals are prefixed with 0. Decimal literals have no prefix.

The suffix d or D can be used to indicate that a floating-point literal is of type double.

 exponential notation uses the letter E or e to represent the exponent

a char variable can be assigned a Unicode character literal using the \u escape sequence followed by four hexadecimal digits that represent the Unicode code point of the character.

The decision to run the garbage collector is ultimately up to the JVM.

Trong Java, có hai loại kiểu dữ liệu chính là kiểu dữ liệu nguyên thủy (primitive types) và kiểu tham chiếu (reference types).

Các kiểu dữ liệu nguyên thủy (primitive types) bao gồm: byte, short, int, long, float, double, char, và boolean. Kiểu dữ liệu nguyên thủy lưu trữ giá trị trực tiếp trong bộ nhớ và có kích thước cố định, được xác định bởi Java.

Các kiểu dữ liệu tham chiếu (reference types) bao gồm: các lớp (class), các giao diện (interface), các mảng (array), và các kiểu dữ liệu được định nghĩa bởi người dùng. Kiểu dữ liệu tham chiếu lưu trữ tham chiếu đến đối tượng trong bộ nhớ, thay vì lưu trữ giá trị trực tiếp. Tham chiếu sẽ trỏ đến vị trí của đối tượng trong bộ nhớ, cho phép ta truy cập và thay đổi dữ liệu của đối tượng đó.

Các kiểu dữ liệu nguyên thủy được lưu trữ trực tiếp trong bộ nhớ, trong khi các kiểu dữ liệu tham chiếu chỉ lưu trữ tham chiếu đến đối tượng trong bộ nhớ. Do đó, việc thao tác trên kiểu dữ liệu nguyên thủy và kiểu dữ liệu tham chiếu khác nhau. Khi truyền kiểu dữ liệu nguyên thủy vào một phương thức, chỉ có giá trị của biến được truyền vào, trong khi khi truyền kiểu dữ liệu tham chiếu, tham chiếu của đối tượng được truyền vào.

Trong Java, không có kiểu dữ liệu nào được truyền theo tham chiếu (pass by reference) trực tiếp. Tuy nhiên, có thể giả lập hành vi truyền theo tham chiếu bằng cách truyền một mảng vào phương thức và thay đổi giá trị của mảng đó bên trong phương thức. -- Trong java luôn luôn là **pass-by-value**.

<https://shareprogramming.net/co-che-pass-by-value-va-pass-by-reference-trong-java/>

Trong Java tất cả các [object](https://shareprogramming.net/object-va-class-trong-java/)sau khi khởi tạo đều được lưu trữ trong vùng nhớ [heap](https://shareprogramming.net/stack-va-heap-trong-java/).

Còn các biến nguyên thủy thì lưu trong stack

Which of the following expressions results in a positive value in x?  
A. int x = -1; x = x >>> 5;  
B. int x = -1; x = x >>> 32;  
C. byte x = -1; x = x >>> 5;  
D. int x = -1; x = x >> 5;

Toán tử >>> là toán tử dịch phải không dấu (unsigned right shift operator), nó sẽ dịch tất cả các bit sang phải và thêm các bit 0 vào bên trái. Toán tử >> là toán tử dịch phải có dấu (signed right shift operator), nó sẽ dịch tất cả các bit sang phải và thêm các bit giữ dấu (bit của dấu) vào bên trái.

Vì vậy, trong các lựa chọn trên, chỉ có lựa chọn A có thể cho kết quả là một giá trị dương. Với lựa chọn A, biến x được khởi tạo với giá trị -1, tương đương với biểu diễn nhị phân 11111111 11111111 11111111 11111111. Khi áp dụng toán tử >>> với tham số là 5, tất cả các bit sẽ được dịch sang phải 5 vị trí, và các bit 0 sẽ được thêm vào bên trái. Kết quả sẽ là 00000111 11111111 11111111 11111111, tương đương với giá trị 2147483647 trong hệ thập phân, là một giá trị dương.

Lựa chọn B không hợp lệ, vì tham số cho toán tử >>> phải nằm trong khoảng từ 0 đến 31 (tương ứng với 32-bit integer). Lựa chọn C không hợp lệ, vì toán tử >>> chỉ hoạt động với các kiểu dữ liệu nguyên (integer types), không phải là kiểu dữ liệu byte. Lựa chọn D sử dụng toán tử >>, do đó sẽ thực hiện phép dịch phải có dấu, và kết quả sẽ là một giá trị âm.

Which of the following expressions are legal? (Choose 2)  
A. String x = "Hello"; int y = 9; x += y;  
B. String x = "Hello"; int y = 9; if (x == y) {}  
C. String x = "Hello"; int y = 9; x = x + y;  
D. String x = "Hello"; int y = 9; y = y + x; giải thích bằng tiếng việt

Lựa chọn A và C là hợp lệ.

Lựa chọn A khởi tạo biến x với giá trị là chuỗi "Hello" và biến y với giá trị là số nguyên 9. Toán tử "+=" được sử dụng để thêm giá trị của biến y vào cuối chuỗi x. Kết quả, biến x sẽ có giá trị là "Hello9".

Lựa chọn C cũng khởi tạo biến x với giá trị là chuỗi "Hello" và biến y với giá trị là số nguyên 9. Toán tử "+" được sử dụng để nối giá trị của biến y vào cuối chuỗi x, kết quả là một chuỗi mới được gán cho biến x.

Lựa chọn B không hợp lệ, vì so sánh chuỗi với số nguyên không có ý nghĩa.

Lựa chọn D cũng không hợp lệ, vì không thể thực hiện phép cộng giữa một chuỗi và một số nguyên.

 Trong Java, theo quy ước của phép chia lấy dư, kết quả của phép tính -8 % 5 sẽ là -3.

Cách tính:

* Tìm số nguyên q sao cho q \* 5 <= -8 (q = -2)
* Tính phần dư r bằng -8 - q \* 5 = -8 - (-10) = 2
* Kết quả của phép tính là phần dư r, do đó kết quả là -3.

7 % -4 = 3

1. 12 % 5 sẽ cho kết quả là 2.
2. -12 % 5 sẽ cho kết quả là -2.
3. 12 % -5 sẽ cho kết quả là 2.
4. -12 % -5 sẽ cho kết quả là -2.
5. 0 % 5 sẽ cho kết quả là 0.
6. 5 % 0 sẽ gây ra một lỗi ArithmeticException ("divide by zero").
7. 10 % 3 sẽ cho kết quả là 1.
8. 10 % -3 sẽ cho kết quả là 1.
9. -10 % 3 sẽ cho kết quả là -1.
10. -10 % -3 sẽ cho kết quả là -1.

cứ số bị chia âm thì kết quả âm

^ là XOR

khi sử dụng toán tử ba ngôi, nếu các giá trị trả về có kiểu dữ liệu khác nhau, chương trình sẽ tự động chuyển đổi kiểu dữ liệu để phù hợp. Ở đây, do giá trị trả về của biểu thức là 9 và 99.99 có kiểu dữ liệu khác nhau (int và double), nên kết quả sẽ được chuyển đổi thành kiểu dữ liệu double để in ra màn hình.

What does the following code do?  
Integer i = null;  
if (i != null & i.intValue() == 5) System.out.println("Value is 5");  
  
A. Prints "Value is 5".  
B. Throws an exception.

Khi chạy đoạn mã trên, nó sẽ ném ra một ngoại lệ NullPointerException.

Giải thích:

* Dòng đầu tiên khai báo và khởi tạo biến i với giá trị null.
* Dòng thứ hai sử dụg toán tử AND (với toán tử đầu tiên là &) để kiểm tra xem biến i có khác null không. Vì biến i đang có giá trị là null, nên kết quả của biểu thức i != null sẽ là false.
* Trong trường hợp toán tử AND (với toán tử đầu tiên là &) được sử dụng, biểu thức bên phải của toán tử sẽ được đánh giá ngay cả khi biểu thức bên trái là false. Do đó, đoạn mã i.intValue() sẽ được thực thi mặc dù biến i đang có giá trị là null, gây ra ngoại lệ NullPointerException.
* Vì vậy, chương trình sẽ ném ra một ngoại lệ NullPointerException thay vì in ra "Value is 5".

toán tử AND (&) được sử dụng, cả hai biểu thức đều được đánh giá bất kể biểu thức nào trước đó đã trả về false.

Điều này là khác biệt so với toán tử AND ngắn (&&) trong Java. Khi sử dụng toán tử AND ngắn, nếu biểu thức bên trái trả về false, biểu thức bên phải sẽ không được đánh giá. Điều này được gọi là short-circuit evaluation.

Vì vậy, khi sử dụng toán tử AND (&), biểu thức bên phải sẽ được đánh giá ngay cả khi biểu thức bên trái là false, dẫn đến việc thực thi của đoạn mã i.intValue() trong trường hợp này, và gây ra ngoại lệ NullPointerException.

When a byte is added to a char in Java, the type of the result is an int.

This is because the Java language specification defines that when two values of different primitive types are combined using binary operators such as +, -, \*, or /, the values are first promoted to a common type, according to a set of rules known as binary numeric promotion.

In the case of a byte and a char, both types are first promoted to int before the addition takes place. The resulting value will therefore be an int.

The return type of the instanceof operator in Java is boolean.

The instanceof operator is used to check if an object belongs to a certain class or implements a certain interface. It takes an object on the left-hand side and a class or interface on the right-hand side, and return

Object obj = new String("Hello, world!");

if (obj instanceof String) {

System.out.println("obj is a String");

} else {

System.out.println("obj is not a String");

}

only variables, expressions, or values of reference types can appear on the left-hand side of the instanceof operator. This includes objects of classes, arrays, and interface types, as well as null references.

Primitive types such as int, double, and boolean cannot appear on the left-hand side of the instanceof operator, since they are not objects in Java.

Object obj = new String("Hello, world!");

if (obj instanceof String) {

System.out.println("obj is a String");

}

String[] arr = new String[10];

if (arr instanceof Object) {

System.out.println("arr is an Object");

}

Runnable r = new Runnable() {

public void run() {

System.out.println("Hello, world!");

}

};

if (r instanceof Runnable) {

System.out.println("r is a Runnable");

}

 reference (tham chiếu) là một địa chỉ trỏ đến một đối tượng cụ thể trong bộ nhớ heap, interface (giao diện) là một kiểu dữ liệu đặc biệt để định nghĩa các phương thức mà các lớp khác có thể triển khai, và object (đối tượng) là một thể hiện cụ thể của một lớp, được tạo ra và lưu trữ trong bộ nhớ heap.

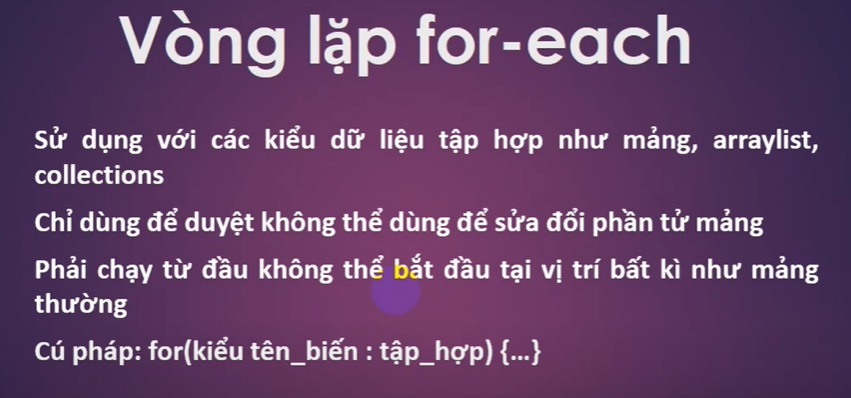
Classes, interfaces, arrays of classes, and arrays of interfaces may be the right-hand operands of instanceof.

|  |
| --- |
| Cả hai phương thức a.sort(x) và Collections.sort(a, x) đều được sử dụng để sắp xếp một mảng hoặc một danh sách theo thứ tự được chỉ định bởi một đối tượng Comparator.  Tuy nhiên, sự khác biệt giữa hai phương thức này là:  a.sort(x) là một phương thức của lớp Arrays trong Java, trong khi Collections.sort(a, x) là một phương thức của lớp Collections.  a.sort(x) chỉ hoạt động với mảng, trong khi Collections.sort(a, x) hoạt động với bất kỳ loại danh sách nào mà thực hiện giao diện List, chẳng hạn như ArrayList, LinkedList,...  a.sort(x) là một phương thức tĩnh (static method), nghĩa là nó được gọi trực tiếp từ lớp Arrays mà không cần tạo đối tượng Arrays trước. Trong khi đó, Collections.sort(a, x) là một phương thức không tĩnh (non-static method), nghĩa là nó phải được gọi trên một đối tượng Collections đã được tạo trước. |

XEM YOUTUBE CÁC THỨ

# [Học Lập trình Java với Netbeans IDE](https://www.youtube.com/playlist?list=PLE1qPKuGSJaA6-6So-knCgNNq3vNbCRD6)

# Java - Bài 20: Cấu trúc lặp for-each



# Bài tập số 3: Phương thức và vào ra File

<https://github.com/thantrieu/JavaMiniProject01/blob/master/Java%20Basic%20Tutorial%20V2/Exercises%20V2/B%C3%A0i%20t%E1%BA%ADp%20Java%20s%E1%BB%91%203.pdf>

Một class không thể là interface và ngược lại. Object là một lớp cha của tất cả các lớp trong Java, bao gồm cả các interface

# Java - Bài 26: Đọc dữ liệu từ file với Scanner

|  |
| --- |
| import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  public class Main {  public static void main(String[] args) throws FileNotFoundException{  // TODO code application logic here  File file = new File("file.txt");  Scanner sc = new Scanner(file);    //đọc từng dòng  if (sc.hasNextLine()){  String line = sc.nextLine();  }  int i=0;  int [] numbers = new int[100];  while (sc.hasNextInt()){  numbers[i] = sc.nextInt();  i++;  if (i>=100) break;  }  showNumber(numbers);  }  //// đọc từng từ  // String[] words = new String[100];  // int i=0;  // while (sc.hasNext()){  // String w = sc.next();  //// System.out.println(w);  // words[i] = w; i++;  // }  // showinfo(words);  // }  //  private static void showinfo(String[] words) {  for (String word : words) {  if (word == null) break;  System.out.println(word);  }  }  private static void showNumber(int[] numbers) {  for (int number : numbers) {  if (number == 0) continue;  System.out.println(number);  }  }  } |

List là cái interface, array là 1 cái thực thi từ cái interface đấy. Mình chuyển sang đấy để hoạt động trên 1 cái mảng thay vì dùng collection. Có thể truy cập vào 1 mảng thông qua chỉ số của mảng.

Biến tĩnh là tất cả các đối tượng đều dùng chung cái biến đấy. Static thì k cần khởi tạo đối tượng vẫn dùng được.

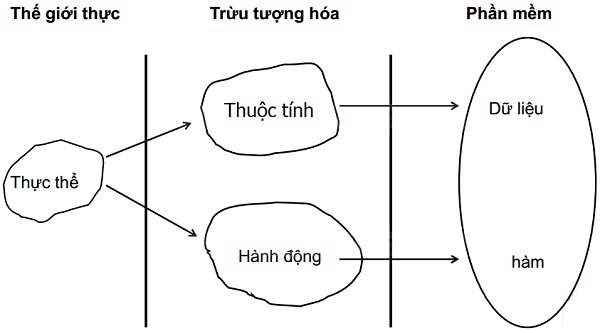
Mỗi lần tạo ra 1 đối tượng mới thì constructor sẽ đc gọi 1 lần

1 phương thức tĩnh không được gọi các phương thức không phải là phương thức tĩnh, và nó chỉ truy cập dc vào biến static

Trong sơ đồ lớp thì các phương thức và thuộc tính được gạch dưới chính là các phương thức static và thuộc tính static.

---

Trừu tượng hóa: giữ lại các đặc trưng quan trọng nhất của đối tượng



Không phải final class nên vẫn extend được

Lang thì luôn luôn có sẵn rồi

Trong interface nó luôn hiểu là: **Public final static**

**Interface và abstract class đều *cannot be instantiated***

***be subclass là được kế thừa. Final thì kh đc kế thừa***