Buổi 2

- **Mảng**
- **Nhập dữ liệu**
- **Hàm, Hằng**
- **Foreach**
- **Giới thiệu lớp Math, String, Number, Character**

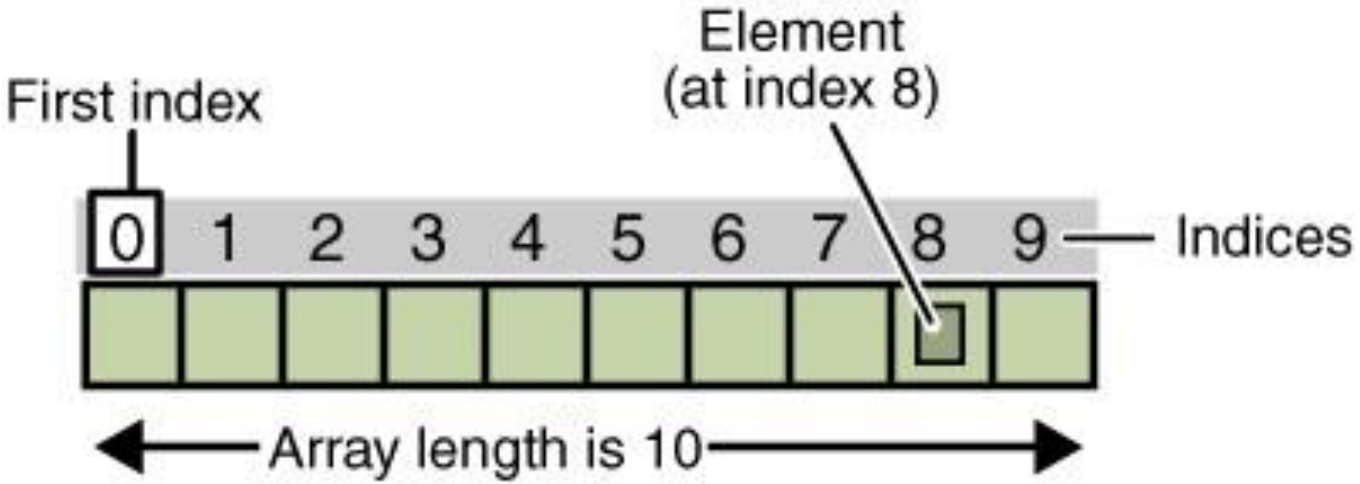
Mảng

Giới thiệu

- Chỉ một giá trị được lưu trữ trong một biến tại một thời điểm.
- Một vòng lặp đơn không thể truy cập và thao tác trên các giá trị rời rạc và không liên kết với nhau.
- Mảng là một cách lưu trữ dữ liệu đặc biệt, nó có thể lưu trữ nhiều phần tử có cùng một kiểu dữ liệu trong các ô nhớ có vị trí liên tiếp nhau.
- Mảng là cách thức tốt nhất cho việc thao tác trên nhiều phần tử dữ liệu có cùng kiểu tại cùng một thời điểm.

Cấu trúc của mảng

- Mảng có một tên riêng, và mỗi phần tử trong mảng sẽ được truy cập thông qua một số nguyên gọi là chỉ số của mảng, và chỉ số này bắt đầu từ 0.
- Kích cỡ của mảng là số lượng lớn nhất các phần tử mà mảng có thể lưu trữ.



Mảng 1 chiều

- Mảng một chiều là một dãy liên tiếp các biến có cùng kiểu dữ liệu.
- Khai báo mảng 1 chiều:

```
datatype[] arrayName = new datatype[size];
```

- Ví dụ:

```
int[] arri = new int[10];

float[] arrf = new float[15];
```

- Hoặc khai báo và khởi tạo mảng:

```
double arrd[] = {10.1, 11.3, 12.5,13.7,14.9, 3.21, -5.63};
```

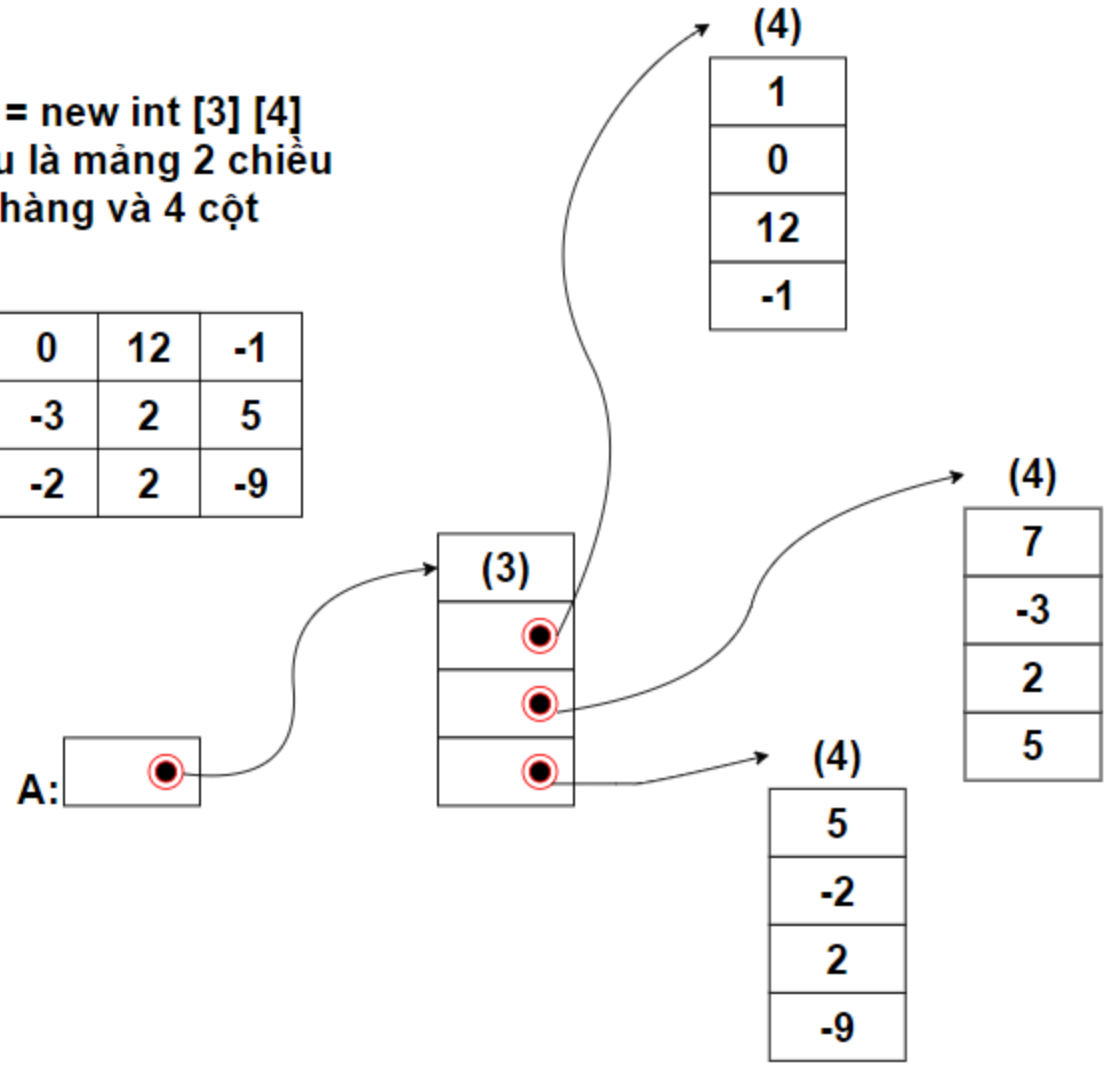
Mảng 2 chiều

- Mảng hai chiều (2D arrays) là mảng một chiều với các phần tử là một mảng một chiều khác.
- Có thể tạo ra mảng ba chiều hay nhiều hơn, nhưng mảng hai chiều được sử dụng nhiều nhất.
- Trong mảng hai chiều, phần tử đầu tiên là: arrayName[0][0].

Mảng A = new int [3] [4]
được hiểu là mảng 2 chiều
với 3 hàng và 4 cột

A:

1	0	12	-1
7	-3	2	5
5	-2	2	-9



Nhưng thực tế, mảng A tham chiếu đến mảng gồm 3 phần tử
Trong mỗi các phần tử đó lại tham chiếu đến 4 phần tử kiểu int

Khai báo mảng 2 chiều

```
// Khai báo một mảng có 5 dòng, 10 cột
        DataType[][] myArray1 = new DataType[5][10];
// Khai báo một mảng 2 chiều có 5 dòng.
        DataType[][] myArray2 = new DataType[5][];
// Khai báo một mảng 2 chiều, chỉ định giá trị các phần tử.
        DataType[][] myArray3 = new DataType[][] {
            { value00, value01, value02 , value03 },
            { value10, value11, value12 }
        };
```

Nhập dữ liệu

- Java là một ngôn ngữ lập trình thuần hướng đối tượng, nên cách **nhập dữ liệu từ bàn phím** cũng phải sử dụng theo hướng đổi tượng. Tức là bạn phải khai báo một đối tượng, sau đó gọi hàm nhập, rồi gán giá trị cho biến.

- **Scanner** là một lớp trong thư viện Java.util giúp bạn nhập và lưu data từ bàn phím. Ngoài scanner, còn có nhiều lớp khác nữa, nhưng đây là lớp được sử dụng nhiều nhất.

Các phương thức thường dùng trong lớp Scanner

Tên phương thức	Tác dụng
nextBoolean	Nhập vào kiểu Boolean (true – false) từ bàn phím
nextByte	Nhập vào kiểu dữ liệu Byte
nextShort	Nhập vào kiểu Short (số nguyên từ -32768 đến 32767)
nextInt	Nhập vào kiểu số nguyên từ bàn phím
nextFloat	Nhập vào kiểu số thực
nextDouble	Nhập vào kiểu Double (số thực lớn hơn float)
nextLine	Nhập vào kiểu String (String trong java giống char luôn nhé!)
nextLong	Nhập vào số nguyên lớn

Cú pháp nhập

Để sử dụng bạn khai báo một đối tượng Scanner có tên tùy ý. Ở đây mình đặt là ip

Scanner ip = new Scanner(System.in);

ip.nextInt(); // Gọi hàm nhập số nguyên. Các hàm khác cú pháp tương tự

Ví dụ gọi hàm nhập :

Int a = **ip.nextInt()**;

Hàm , Phương thức

Định nghĩa:

Hàm là cách gói gọn một đoạn code thực hiện một tác vụ được định nghĩa cụ thể.

- Mỗi hàm có một tên (ngắn gọn và dễ nhớ), danh sách các đối biểu diễn dữ liệu mà hàm xử lý, giá trị trả về biểu diễn kết quả xử lý.
- Hàm có thể trả về giá trị, hoặc không trả về giá trị.
- Hàm có thể có đối hoặc không có đối.
- Sử dụng hàm giúp giảm bớt những đoạn code giống nhau, chương trình rõ rang và dễ sửa lỗi hơn.

Cú pháp

```
modifier returnType nameOfMethod (Parameter List) {
    // method body
}
```

- Modifiers (tạm dịch là phạm vi sửa đổi và truy cập)
- returnType: kiểu dữ liệu trả về
- nameOfMethod: tên của hàm(method)
- Parameter là các tham số đầu vào của hàm(có thể có nhiều tham số với nhiều kiểu dữ liệu khác nhau)
- // body: là các mã code bên trong hàm

Ví dụ:

```
public static int sum(int a, int b) {
    return a + b;
}
```

Ví dụ

Hàm không có đối, không trả về giá trị

```
public void printHello(){
    System.out.println("Xin chao cac ban");
}
```

Hàm có đối, có trả về giá trị

```
public int sumTwoNumber(int a, int b){
    return a + b;
}
```

Hàm sumTwoNumber có hai đối kiểu int là a và b, hàm trả về một giá trị kiểu int.

Để sử dụng hàm:

– Gọi hàm thông qua tên hàm

– Truyền tham số cho hàm

Ví dụ:

```
printHello();
int c = sumTwoNumber(15, 18);
```

• printHello() hay sumTwoNumber(15, 18) là các lời gọi hàm.

• Hàm printHello() không có đối nên trong lời gọi không truyền tham số.

• Hàm sumTwoNumber(...) có hai đối nên khi gọi hàm ta truyền hai tham số (15 và 18), giá trị hàm trả về được gán cho biến c.

Đệ quy

- Hàm đệ quy là hàm có lời gọi đến chính nó trong phần than hàm
- Ví dụ tính giai thừa

```
static int giaiThua(int n){
        if(n == 0)
                return 1;
        else
                return n * giaiThua(n - 1);
}
```

Truyền tham số vào hàm

- Có hai loại tham số là tham trị và tham chiếu.
- Truyền tham trị:

– Tham số truyền vào là các hằng giá trị, hoặc giá trị chứa trong các biến.
– Nếu tham số là biến thì giá trị của biến sẽ được truyền vào hàm. Hàm không thể thay đổi giá trị của biến.

- Truyền tham chiếu:

– Tham số truyền vào hàm phải là các biến
– Địa chỉ của biến sẽ được truyền vào hàm.
– Hàm có thể làm thay đổi giá trị của biến truyền vào.

- Mặc định trong java tất cả các kiểu dữ liệu cơ bản đều là tham trị int, float, double, long, short, char, boolean
- Tất cả các kiểu dữ liệu cấu trúc đều là tham chiếu Object, Interface, Array, Collection.

Hằng

Hằng là một biến mà **giá trị không đổi** trong suốt chương trình, tất nhiên ta đã khởi tạo giá trị ngay từ đầu.
Lý do sử dụng hằng:
•Tạo ra những giá trị vốn thực tế không cho thay đổi, làm chương trình an toàn hơn.
•Giúp người đọc biết ý nghĩa con số vô cảm trong khoa học như có thể áp dụng giá trị số PI, gia tốc trọng trường,...
•Sẽ cảnh báo nếu người dùng cố tình thay đổi giá trị sau này. Đảm bảo tính toán vẹn của giá trị.

Cú pháp:
final <kiểu dữ liệu> <tên biến> = <giá trị hằng>;

```java
public class HelloWorld{
    public static void main(String []args){
            final double PI = 3.1415926535897;
            int r = 3;
            System.out.print(2*r*PI);
    }
}
```

```
"C:\Program Files\Java\jdk-11.0.11\bin\ja
18.849555921538197
Process finished with exit code 0
```

```java
public class HelloWorld{
    public static void main(String []args){
            final double PI = 3.1415926535897;
            PI = 3.14;
    }
}
```

```
D:\Downloads\B4_VuThienLy_2019602384\demo\src\demo\Main.java:6:9
java: cannot assign a value to final variable PI
```

ForEach

- Vòng lặp **for each** (hay được gọi là **enhanced for**) trong Java được giới thiệu trong phiên bản Java 5 (hiện tại đã ra tới phiên bản Java 17).
- Vòng lặp **for each** cung cấp một cách tiếp cận khác để duyệt mảng hoặc collection trong Java tốt hơn.
- Nó chủ yếu được sử dụng để duyệt qua các phần tử của mảng hoặc collection.
- **For each** cũng được bắt đầu với từ khóa for giống như trong vòng lặp for thông thường.
- Nhưng thay vì khai báo hay khởi tạo biến đếm trong vòng lặp thì chúng ta sẽ khai báo một biến có cùng loại với kiểu cơ sở của mảng, theo sau là dấu hai chấm " : " và cuối cùng là tên mảng.
- Vòng lặp **for each** giúp chúng ta duyệt các phần tử trong mảng hay collection mà không cần đến index của các phần tử đó.

Cú pháp của vòng lặp for each trong Java

```
for (kieuDuLieu bienDaiDien : tenMang) {
    // Khối lệnh được lặp lại
}
```

Ví dụ:

```
// Khai báo một mảng
int[] arr = {1, 2, 5, 8, 9};
// Sử dụng vòng lặp for each để lặp qua mảng
for(int n : arr) {
    System.out.println(n);
}
```

```
1
2
5
8
9
```

Ưu và nhược điểm

-Ưu điểm của vòng lặp for each trong Java

- Sử dụng vòng lặp for each làm cho code dễ đọc hơn
- Giảm bớt khả năng lỗi khi lập trình

-Nhược điểm của vòng lặp for each trong Java

- Vòng lặp for each không thể chỉnh sửa mảng, chỉ được dùng để duyệt qua tất cả các phần tử trong mảng.
- Không thể tìm được vị trí phần tử trong mảng.
- Vòng lặp for each chỉ duyệt được xuôi, không thể duyệt ngược mảng.

Vậy khi nào thì nên sử dụng for each trong lập trình Java?

- Có thể sử dụng for each khi bạn chỉ muốn duyệt qua tất cả các phần tử trong mảng hay collection, duyệt từ đầu đến cuối, không bỏ sót một phần tử nào, không cần chỉnh sửa giá trị mảng, không cần tìm index của phần tử trong mảng hay collection.

Lớp Math

- JDK định nghĩa săn một số lớp tiện dụng, một trong số là là lớp **Math** cung cấp các hàm về toán học. Bạn không cần phải tạo đối tượng lớp Math vì các hàm trong lớp đó là static, để gọi hàm chỉ đơn giản viết tên lớp Math và tên phương thức cần gọi.

- Trước khi gọi các hàm Math, bạn có thể import package để khỏi phải viết đầy đủ tên pack, như:

```
import java.lang.Math;
```

Math.PI hằng số PI

```
double g45 = Math.PI/4;
```

Math.abs() trả về giá trị tuyệt đối của tham số

```
int a = Math.abs(10); // 10 int b = Math.abs(-20); // 20
```

Math.ceil() trả về giá trị double là số làm tròn tăng bằng giá trị số nguyên gần nhất

```
double c = Math.ceil(7.342); // 8.0
```

Math.floor() trả về double là số làm tròn giảm

```
double f = Math.floor(7.343); // 7.0
```

Math.max() lấy số lớn trong hai số

```
int m = Math.max(10, 20); // 20
```

Math.min lấy số nhỏ

```
int m = Math.min(10, 20); // 10
```

Math.pow lấy lũy thừa (cơ-số, số mũ)

```
double p = Math.pow(2, 3); // 8.0
```

Math.Math.sqrt() khai căn

```
double a = Math.sqrt(9);//3
```

Math.sin(), Math.cos() sin và cos của góc đơn vị radian

```
double s = Math.sin(Math.PI/2);//1
```

Math.random() sinh số double ngẫu nhiên từ 0 đến 1

```
double r = Math.random();
```

Math.toDegrees() đổi góc radian thành độ

```
double goc = Math.toDegrees(Math.PI/2); //90
```

Math.toRadians() đổi góc đơn vị độ ra radian

```
double goc = Math.toRadians(45);//0.7853981633974483
```

String

Một đối tượng chuỗi có thể được tạo ra như sau:

```
String str = new String();
```

• Để gán dữ liệu cho một biến chuỗi:

```
str = "Hello World";
```

Lớp **String trong java** cung cấp rất nhiều các phương thức để thực hiện các thao tác với chuỗi như: compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring(), ...

Một số phương thức quan trọng trong lớp **String**:

– length(): viết là str.length() - trả về độ dài của chuỗi str

– charAt(i): str.charAt(i) - trả về ký tự str[i]

– indexOf(): str.indexOf(c | s) trả về vị trí (chỉ số) xuất hiện đầu tiên của ký tự c hay chuỗi s nếu c hay s có trong chuỗi str, ngược lại trả về -1.

– concat(s): str.concat(s) – trả về chuỗi mới bằng cách cộng chuỗi s vào cuối str.

Một số phương thức quan trọng trong lớp **Strin**g:

– compareTo(s): str.compareTo(s) - so sánh hai chuỗi str và chuỗi s (theo thứ tự bảng chữ cái – thứ tự từ điển – phân biệt chữ hoa và chữ thường) và trả về một số nguyên.

- Số nguyên âm nếu chuỗi str đứng trước (nhỏ hơn) chuỗi s.
- Số nguyên dương nếu chuỗi str đứng sau (lớn hơn) chuỗi s.
- Số không nếu chuỗi str giống hệt chuỗi s.

– compareToInogeCase(): so sánh 2 chuỗi không phân biệt chữ hoa, chữ thường.

Một số phương thức quan trọng trong lớp **String**:

– str.lastIndexOf(c|s): trả về vị trí (chỉ số) xuất hiện cuối cùng của ký tự c hay chuỗi s trong chuỗi str.

– str.replace(c|s, cn|sn): trả về chuỗi mới sau khi thay thế tất

cả các ký tự c hay chuỗi s bằng ký tự cn hay chuỗi cn trong chuỗi str.

– str.substring(m[, n]): trả về một chuỗi con lấy từ vị trí m [đến vị trí n] trong chuỗi str (không có n thì lấy đến cuối str).

– str.toString(): trả về đối tượng chuỗi.

– str.trim(): cắt các khoảng trắng ở đầu và cuối chuỗi str.

– str.split(): cắt xâu ra thành một mảng các phần tử dựa vào xâu đầu vào

Mảng String

- Là một mảng mà các phần tử mảng là các chuỗi.
- Khai báo và khởi tạo một mảng các đối tượng chuỗi:

  String[] numbers = new String[10];
- Cần phải cấp phát bộ nhớ cho từng đối tượng của mảng:

  numbers[0]= new String(“twenty");
- Hoặc: str = {"one two three four five six seven eight nine ten"};

  numbers = str.split(" ");
- Hoặc: String[] numbers = {"one", "two", "three", "four", "five","six", "seven", "eight", "nine", "ten"};

- Ta có:numbers[0] = "one", numbers[1] = "two", ..., numbers[9] = "ten"

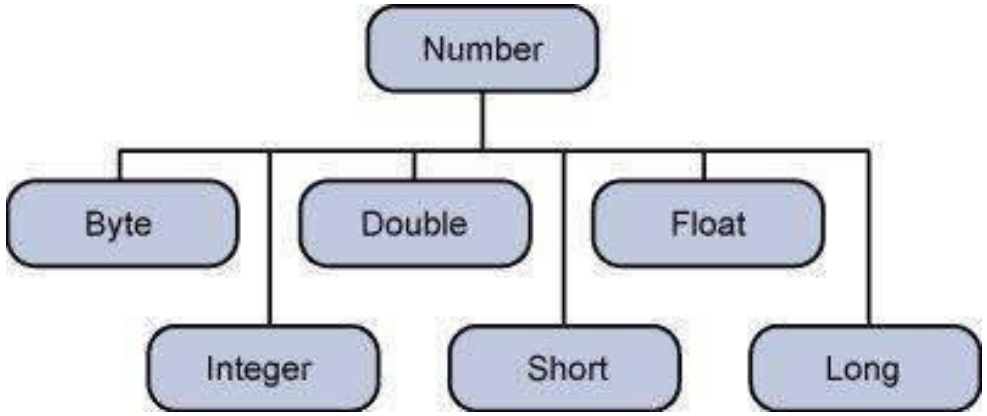
Một số lớp mở rộng của String:

-StringBuilder

-StringBuffer

(tìm hiểu thêm)

Lớp NumBer



Bình thường chúng ta hay dung kiểu dữ liệu nguyên thủy: int , float , ……

```
2   public class Test {
3      public static void main(String[] args){
4         Integer a = 5; // khai báo một đối tượng Interger
5         Float b = 6.5f;// khai báo một đối tượng Float
6         Double c = 6.5;// khai báo một đối tượng Double
7      }
8   }
```

Các phương thức thuộc lớp Number trong java

STT	Tên phương thức và miêu tả	Ví dụ
1	xxxValue() như: intValue() , floatValue() , doubleValue()... Biến đổi giá trị của đối tượng Number này thành kiểu dữ liệu xxx.	Integer a = 5; int x = a.intValue();
2	compareTo() so sánh đối tượng này với tham số. Trả về một kiểu dữ liệu của đối tượng mang giá trị 0 nếu bằng tham số. Trả về giá trị 1 nếu tham số bé hơn giá trị đối tượng, trả về -1 nếu lớn hơn đối tượng.	Integer a = 5; int x = a.compareTo(5);// Trả về 0
3	boolean equals() Kiểm tra đối tượng có bằng với tham số không.	Float a = 6.5f; boolean x = a.equals(6.5f) ;// Trả về true
4	toString() Trả về đối tượng String	Integer a = 567; String s = a.toString(); // s="567"
5	int parseInt(String s) Trả về giá trị int của chuỗi s	int a = Integer.parseInt("456"); // a = 456;

Lớp Character

Thông thường khi làm việc với lớp Character, chúng ta sử dụng kiểu dữ liệu char gốc.

```
char ch = 'a'; // Unicode for uppercase Greek omega character
char uniChar = '\u039A'; // an array of chars
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };
```

- Tuy nhiên trong quá trình phát triển, chúng ta sẽ gặp phải các tình huống cần phải sử dụng các đối tượng thay vì sử dụng các kiểu dữ liệu gốc. Để làm được điều này, Java cung cấp lớp wrapper là Character với kiểu dữ liệu char gốc.
- Lớp Character trong Java cung cấp một số phương thức, lớp hữu ích (ví dụ lớp static, ...) để thao tác với các ký tự. Bạn có thể tạo một đối tượng Character bằng Character constructor.

```
Character ch = new Character('a');
```

Các Ký Tự Xử Lý Văn Bản Trong Java

- Ký tự đặc biệt được đặt trước dấu gạch chéo ngược (\) là ký tự xử lý văn bản trong Java, ký tự này có ý nghĩa đặc biệt với trình biến dịch.
- Ký tự dòng mới (\n) thường được sử dụng trong hướng dẫn trong lệnh System.out.println() để lấy dòng tiếp theo sau khi chuỗi được in.

Ký tự	Mô tả
\t	Chèn một tab vào văn bản tại điểm này.
\b	Chèn một backspace vào văn bản tại điểm này.
\n	Chèn một dòng mới vào văn bản tại điểm này.
\r	Chèn về đầu dòng vào văn bản tại điểm này.
\f	Chèn cuộn trang giấy vào máy in tại điểm này.
\'	Chèn dấu nháy đơn vào văn bản tại điểm này
\"	Chèn dấu nháy kép vào văn bản tại điểm này
\\	Chèn một ký tự dấu chéo ngược vào văn bản tại điểm này.

Khi gặp một ký tự ngắt trong một lệnh print, trình biên dịch sẽ biên dịch nó cho phù hợp.

Ví dụ

Nếu muốn đặt các trích dẫn trong dấu ngoặc kép, bạn phải sử dụng ký tự \", trích dẫn đặt bên trong.

```
public class Test {
      public static void main(String args[]) {
        System.out.println("She said \"Hello!\" to me.");
      }
}
```

```
She said "Hello!" to me.
```

Các phương thức của lớp Character trong Java

STT	Phương thức và mô tả
1	**isLetter()**: Xác định xem giá trị char được chỉ định có phải là chữ cái hay không.
2	**isDigit()**: Xác định xem giá trị char được chỉ định có phải là chữ số hay không.
3	**isWhitespace()**: Xác định xem giá trị char chỉ định có phải là khoảng trắng hay không.
4	**isUpperCase()**: Xác định giá trị char chỉ định có phải chữ hoa hay không.
5	**isLowerCase()**: Xác định giá trị char chỉ định có phải chữ thường hay không.
6	**toUpperCase()**: Trả về cấu trúc dạng chữ hoa của giá trị char đã cho.
7	**toLowerCase()**: Trả về cấu trúc dạng chữ thường của giá trị char đã cho.
8	**toString()**: Trả về đối tượng String biểu diễn giá trị ký tự đã cho, là một chuỗi gồm một ký tự.

-Hết-