CẤU TRÚC ĐIỀU KHIỂN VÀ MẢNG

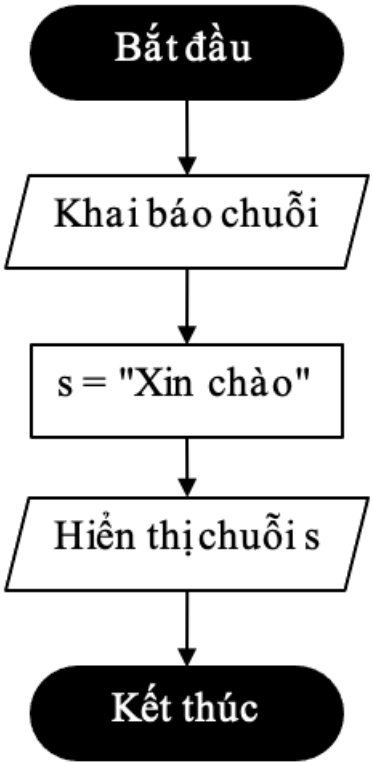
NỘI DUNG TRONG CHƯƠNG

- *Cấu trúc điều khiển*
- *Khối lệnh*
- *Cấu trúc điều kiện: if, switch*
- *Kiểu dữ liệu Enum*
- *Cấu trúc lặp: for, while, do...while*
- *Làm việc với mảng*

Chương 3 bao gồm các nội dung chính liên quan đến cấu trúc điều khiển và mảng. Các cấu trúc điều khiển được sử dụng phổ biến trong Java như if...else, switch...case, while, do...while và for được minh họa thông qua các sơ đồ luồng và các ví dụ trực quan giúp người đọc có thể hiểu được cách thức vận hành và cách thức viết mã của các cấu trúc này. Một số các kiểu dữ liệu đặc biệt như kiểu liệt kê (Enum) hay kiểu mảng (Array) cũng được đề cập tới trong chương giúp người đọc hiểu biết thêm về kiểu dữ liệu trong Java.

CẤU TRÚC ĐIỀU KHIỂN VÀ KHỐI LỆNH

Một **thuật toán** có thể được biểu diễn bởi một **sơ đồ luồng** (flowchart). Sơ đồ luồng của một chương trình đơn giản được thể hiện như **hình 3-1**. Khi đó chương trình không thực hiện rẽ nhánh, vòng lặp, các câu lệnh sẽ được thực hiện toàn bộ, từ trên xuống dưới, mỗi câu lệnh thực hiện một lần. Tuy nhiên trong thực tế các bài toán có logic phức tạp hơn rất nhiều. Khi đó cần có các cấu trúc rẽ nhánh, lặp. Các cấu trúc này cho phép thực hiện, không thực hiện hoặc thực hiện đi thực hiện lặp lại một khối lệnh. Số lần thực hiện thường được quyết định bằng các biểu thức logic.



Hình 3-1 Sơ đồ luồng thực hiện một chương trình đơn giản

Khối lệnh bao gồm một chuỗi các câu lệnh nằm giữa một cặp dấu ngoặc nhọn "{" và "}". Ví dụ:

```
1  {
2     Câu lệnh
3  }
```

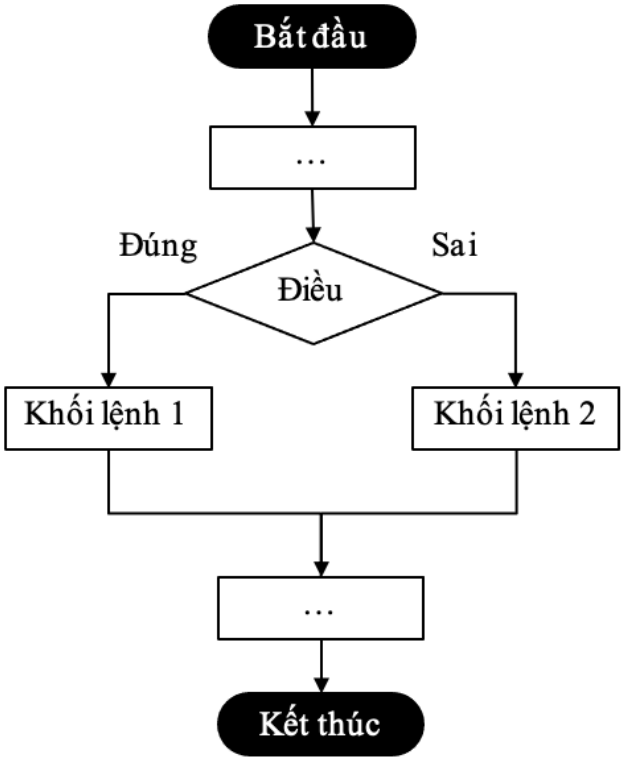
Trên thực tế, một khối có thể không chứa câu lệnh nào; một khối như vậy được gọi là **khối trống** và hữu ích trong một số trường hợp. Một khối lệnh có thể được sử dụng hợp lệ ở bất cứ nơi nào mà một câu lệnh có thể xảy ra. Có một nơi mà một khối là bắt buộc đó là trong trường hợp khai báo một **phương thức**. Thực tế, một phương thức chỉ là một **khối lệnh được đặt tên**. Đây là hai ví dụ về khối lệnh không được đặt tên:

```
 1  {
 2     System.out.print("Câu trả lời là");
 3     System.out.println(traloi);
 4  }
 5  {
 6     int bientam;   // Khai báo biến tạm
 7     temp = x;      // Lưu giá trị biến x vào biến bientam
 8     x = y;         // Lưu giá trị của biến y vào biến x
 9     y = bientam;      // Lưu giá trị biến bientam vào biến y
10  }
```

Trong ví dụ thứ hai, một **biến tạm thời** (bientam), được khai báo bên trong khối. Điều này là hoàn toàn hợp lệ và việc khai báo một biến bên trong một khối để đảm báo biến đó chỉ được **sử dụng bên trong khối**. Một biến được khai báo bên trong một khối hoàn toàn không thể truy cập được và không thể nhìn thấy được từ bên ngoài khối đó. Khi máy tính thực hiện câu lệnh khai báo biến, nó sẽ **cấp phát** bộ nhớ để giữ giá trị của biến. Khi khối kết thúc, vùng nhớ đó sẽ được **thu hồi**.

CÂU LỆNH ĐIỀU KIỆN IF...ELSE

Cấu trúc điều kiện trong Java được viết dưới dạng câu lệnh "**if** ... **else**" . Sơ đồ khối cấu lệnh điều kiện như sau:



Hình 3-2 Sơ đồ khối cấu trúc rẽ nhánh bằng câu lệnh if

```
1  if (biểu thức){
2      khối lệnh 1
3  } else{
4      khối lệnh 2
5  }
```

Câu lệnh **if** yêu cầu máy tính thực hiện một trong hai hành động, tùy thuộc vào việc giá trị của một b**iểu thức** đã cho là đúng hay sai. Khi máy tính thực hiện câu lệnh if, nó sẽ đánh giá giá trị của biểu thức. Nếu giá trị là **true**, máy tính thực hiện . Nếu giá trị của biểu thức là **false**, thì máy tính sẽ bỏ qua **khối lệnh 1khối lệnh 1** và thực hiện **khối lệnh 2**. Lưu ý rằng trong mọi trường hợp, một và chỉ một khối lệnh được thực thi. Một ví dụ về câu lệnh if ... else:

```
 1  public class Main {
 2    public static void main(String[] args) {
 3      int time = 22;
 4      if (time < 10) {
 5        System.out.println("Good morning.");
 6      } else if (time < 20) {
 7        System.out.println("Good day.");
 8      }  else {
 9        System.out.println("Good evening.");
10      }
11    }
12  }
```

Kết quả khi chạy sẽ hiển thị:

```
Good evening.
```

Trong nhiều trường hợp, nếu muốn lựa chọn giữa **thực thi hoặc không thực thi** một khối lệnh, có thể sử dụng câu lệnh **if** bỏ qua phần **else** như sau:

```
1  if (biểu thức boolean)
2      khối lệnh
```

Ví dụ **hoán đổi x và y** khi x lớn hơn y:

```
1  if ( x > y ) {
2      int temp;     //Một biến tạm thời được sử dụng trong khối này.
3      temp = x;     // Lưu một bản sao của giá trị x tại biến temp.
4      x = y;        // Sao chép giá trị của y vào x.
5      y = temp;     // Sao chép giá trị của biến temp vào y.
6  }
```

Đây là câu lệnh if hoán đổi giá trị của hai biến x và y, nhưng chỉ khi x lớn hơn y thì chương trình mới thực hiện, còn khi x nhỏ hơn y thì việc hoán đổi là không cần thiết. Sau khi câu lệnh if này được thực thi, giá trị của x sẽ luôn nhỏ hơn hoặc bằng giá trị của y.

Lưu ý về việc sử dụng dấu ngoặc

Ví dụ, giả sử viết mã:

```
1  if (x > 0)
2      if (y > 0)
3         System.out.println("Trường hợp đầu tiên");
4  else
5      System.out.println("Trường hợp thứ hai");
```

Bây giờ, hãy nhớ rằng cách **thụt lề** này không có ý nghĩa gì đối với máy tính. Bạn có thể nghĩ rằng phần **else** là nửa sau của câu lệnh "if (x> 0)", nhưng quy tắc mà máy tính tuân theo sẽ gắn phần **else** vào **"if (y> 0)"**. Hay nói cách khác, máy tính thực hiện câu lệnh theo cách sau:

```
1  if ( x > 0 )
2      if (y > 0)
3         System.out.println("Trường hợp đầu tiên");
4      else
5          System.out.println("Trường hợp thứ hai");
```

Có thể buộc máy tính sử dụng theo cách mình mong muốn bằng cách thêm dấu ngoặc nhọn như sau:

```
1  if ( x > 0 ) {
2      if (y > 0)
3         System.out.println("Trường hợp đầu tiên");
4  }
```

```
5  else
6      System.out.println("Trường hợp thứ hai");
```

Câu lệnh rỗng

Trong phần này sẽ đề cập đến một loại câu lệnh nữa trong Java: **câu lệnh rỗng**.

```
1  if (x < 0) {
2      x = -x;
3  };
```

Đây là một câu lệnh chỉ bao gồm một dấu chấm phẩy và nó yêu cầu máy tính không làm gì cả. Sự tồn tại của câu lệnh rỗng làm cho câu lệnh sau hợp lệ:

```
1  if ( done )
2     ;  // Câu lệnh trống
3  else
4     System.out.println( "Khác");
```

Không thể bỏ **dấu chấm phẩy** trong ví dụ trên, vì cú pháp Java yêu cầu một câu lệnh giữa **if** và **else**. Tuy nhiên, cũng có thể sử dụng một khối trống { } không có gì ở giữa, cho những trường hợp như vậy. Đôi khi, các câu lệnh rỗng có thể gây ra **lỗi khó tìm** trong chương trình. Ví dụ: đoạn chương trình sẽ luôn in ra chữ "Hello" cho dù giá trị của x là lớn hơn hay nhỏ hơn 0;

```
1  if (x < 0); {
2      System.out.println( "Hello");
3  }
```

Lý do vì ";" ở cuối dòng đầu tiên là một câu lệnh, và câu lệnh rỗng này được thực thi. Câu lệnh System.out.println không thực sự nằm trong câu lệnh if, vì vậy nó sẽ luôn thực hiện trong mọi trường hợp.

Gán giá trị trong if

Một lưu ý khác cũng cần để ý đó là việc **gán giá trị trong if**. Hãy xem xét hai đoạn mã sau:

```
1  int y;
2  if (x < 0) {
3      y = 1;
4  }
5  else {
6      y = 2;
7  }
8  System.out.println(y);
```

```
int y;
if (x < 0) {
     y = 1;
}
if (x >= 0) {
    y = 2;
   }
System.out.println(y);
```

Trong phiên bản bên trái, y được gán giá trị 1 nếu x < 0 và được gán giá trị 2 nếu ngược lại. Điều này cũng được thực hiện tương tự với biên bản bên phải. Tuy nhiên, trên thực tế, trình biên dịch Java sẽ **báo lỗi cho câu lệnh System.out.println** trong đoạn mã bên phải, trong khi đoạn mã bên trái hoàn toàn ổn.

Vấn đề là trong đoạn mã bên phải, máy tính không thể biết rằng **biến y chắc chắn đã được gán một giá trị**. Khi câu lệnh if không có phần else, câu lệnh bên trong if có thể có hoặc có thể không được thực thi,

tùy thuộc vào giá trị của điều kiện. Trình biên dịch không thể biết liệu nó có được thực thi hay không, vì điều kiện sẽ chỉ được đánh giá khi chương trình đang chạy. Đối với mã ở bên phải ở trên, theo như trình biên dịch, có thể không có câu lệnh nào, y = 1 hoặc y = 2, sẽ được đánh giá, vì vậy có thể câu lệnh đầu ra (output) đang **cố gắng in một giá trị không xác định** và trình biên dịch coi đây là một lỗi. Giá trị của một biến chỉ có thể được sử dụng nếu trình biên dịch có thể xác minh rằng biến đó sẽ được gán giá trị tại thời điểm đó khi chương trình đang chạy. Lưu ý rằng, trong đoạn mã bên trái ở trên, y chắc chắn được gán một giá trị, vì trong câu lệnh if...else, một trong hai lựa chọn thay thế sẽ được thực thi bất kể giá trị của điều kiện trong if là bao nhiêu.
Xét một ví dụ khác dưới đây. Sau khi đoạn mã bên trái được thực thi, x là 1; sau mã ở bên phải, x là 2.

```
1  int x;
2  x = -1;
3  if (x < 0)
4      x = 1;
5  else
6      x = 2;
```

```
int x;
x = -1;
if (x < 0)
    x = 1;
if (x >= 0)
    x = 2;
```

CÂU LỆNH KHẲNG ĐỊNH ASSERT

Khẳng định (assertions) trong Java giúp phát hiện lỗi bằng cách kiểm tra mã mà chúng ta cho là đúng. Một khẳng định được thực hiện bằng cách sử dụng từ khóa **assert**.

Cú pháp của nó là:

```
assert condition;
```

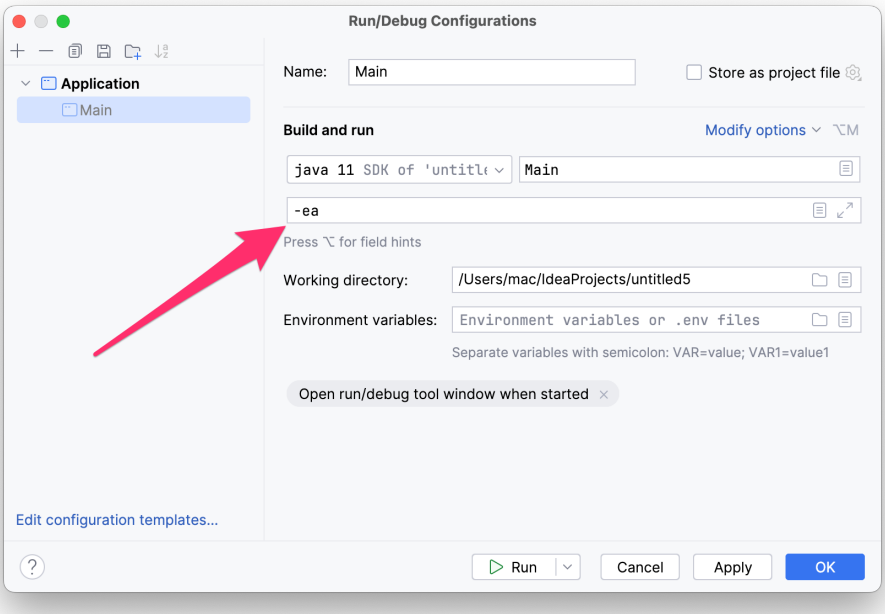
Tại đây **condition** là một biểu thức mà chúng ta khẳng định là đúng khi chương trình thực thi. Nếu điều khẳng định không đúng, một ngoại lệ sẽ xuất hiện.

Ví dụ:

```
1  String[] weekends = {"Friday", "Saturday", "Sunday"};
2  assert weekends.length == 2;
3  System.out.println("There are " + weekends.length + "  weekends in a week");
```

Trong đoạn mã trên, tại dòng lệnh số hai, một khẳng định được đưa ra đó là độ dài của mảng **weekends** là 2. Tuy nhiên vì mảng trên có độ dài bằng 3 vì vậy một lỗi sẽ xuất hiện trong chương trình.

Tuy nhiên tính năng này chỉ hoạt động khi lúc chạy chương trình cần bật tính năng này bằng tham số -ea. Trong IntelliJ IDEA, để bật tính năng này cần khai báo **VM options** ở bên trong **Run/Debug Configurations**

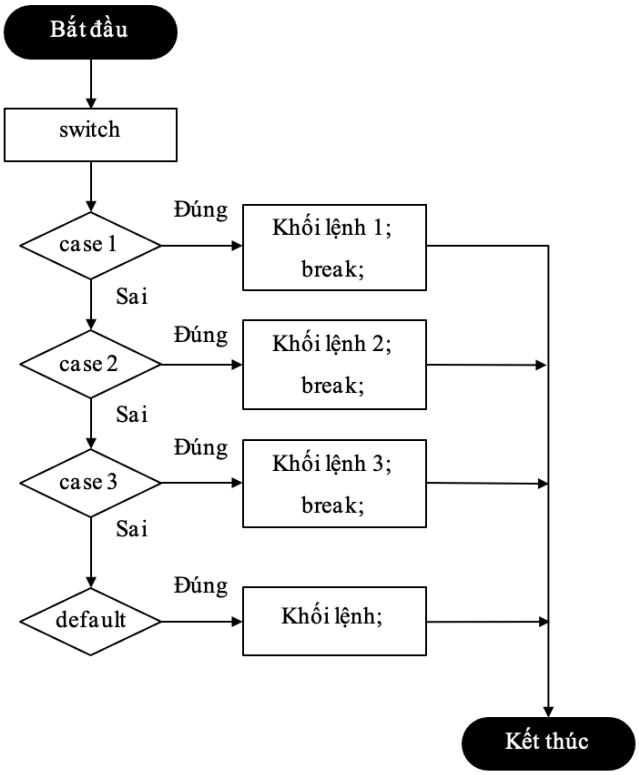


Hình 3-3 Kích hoạt tác dụng của từ khóa assert trong IntellJ IDEA

CÂU LỆNH SWITCH

Câu lệnh switch truyền thống

Câu lệnh **switch** cho phép kiểm tra giá trị của một biểu thức và tùy thuộc vào giá trị đó, để chuyển trực tiếp câu lệnh và khối lệnh tương ứng.



Hình 3-4 Sơ đồ khối câu lệnh switch

Cú pháp của câu lệnh **switch** kèm theo các **case** là các trường hợp có thể xảy ra:

```
switch (biểu thức) {
    case hằng số 1:
        các lệnh 1
```

```
        break;
    case hằng số 2:
        các lệnh 2
        break;
        .
        .
        .
    case hằng số n:
        các lệnh n
        break;
    default:
        các lệnh n + 1
}
```

Giá trị của **biểu thức** có thể là một trong các kiểu số nguyên như **int**, **short, long** hoặc **byte**. Nó có thể là kiểu khác như **char**, **String**, hoặc kiểu **enum** (liệt kê). Đặc biệt, lưu ý rằng biểu thức không được là giá trị **double** hoặc **float**. Các vị trí trong một câu lệnh **switch** mà nó có thể nhảy đến khi giá trị của nó bằng các hằng số khai báo trong mỗi **case**. Cũng có thể sử dụng nhãn **default** trong câu lệnh switch; điều này cung cấp một điểm nhảy mặc định được sử dụng khi giá trị của biểu thức không được liệt kê trong bất kỳ hằng số nào.

Các câu lệnh **break** trong **switch** không thực sự bắt buộc theo cú pháp của câu lệnh switch. Tác dụng của break là làm cho máy tính vượt qua phần cuối của câu lệnh switch, bỏ qua tất cả các trường hợp còn lại. Nếu bỏ qua câu lệnh break, máy tính sẽ chỉ tiếp tục sau khi hoàn thành một trường hợp và sẽ thực thi các câu lệnh được liên kết với **case tiếp theo**. Lưu ý rằng có thể sử dụng nhiều nhãn liên tiếp, chứa nhiều hằng số khác nhau. Điều này chỉ có nghĩa là máy tính sẽ nhảy đến cùng một nơi và thực hiện cùng một hành động cho khi biểu thức có giá trị bằng một trong các nhãn.

Câu lệnh switch trên có thể thay thế bằng các câu lệnh if ... else như sau:

```
if (biểu thức == hằng-số-1) {
    các lệnh 1
}
else if (iểu thức == hằng-số-2) {
    các lệnh 2
}
    .
    .
    .
else if (biểu thức== hằng-số-n) {
    các lệnh n
}
else {
    các lệnh n + 1
}
```

Câu lệnh **switch** có thể hiệu quả hơn câu lệnh **if** vì máy tính có thể đánh giá một biểu thức và chuyển trực tiếp đến trường hợp đúng, trong khi trong câu lệnh if, máy tính phải đánh giá tối đa N biểu thức trước khi nó biết bộ câu lệnh nào sẽ thực thi. Đây là một ví dụ về câu lệnh **switch**:

```
public class Main {
  public static void main(String[] args) {
```

```
    int day = 4;
    switch (day) {
      case 1:
        System.out.println("Monday");
        break;
      case 2:
        System.out.println("Tuesday");
        break;
      case 3:
        System.out.println("Wednesday");
        break;
      case 4:
        System.out.println("Thursday");
        break;
      case 5:
        System.out.println("Friday");
        break;
      case 6:
        System.out.println("Saturday");
        break;
      case 7:
        System.out.println("Sunday");
        break;
    }
  }
}
```

Kết quả hiển thị:

```
Thursday
```

Kiểu liệt kê enum

Kiểu **liệt kêenum** là một kiểu dữ liệu đặc biệt cho phép một biến có thể là một tập hợp các hằng số định sẵn. Ví dụ, giả sử rằng kiểu của biểu thức là kiểu liệt kê Season được xác định bởi

```
enum Season {SPRING, SUMMER, FALL, WINTER}
```

Kiểu của biểu thức trong lệnh switch có thể là kiểu liệt kê. Trong trường hợp đó, các hằng số trong case phải là các giá trị từ kiểu liệt kê. Như trong trường hợp này, các hằng số trong **case** phải được chọn trong số các giá trị SPRING, SUMMER, FALL hoặc.WINTER. Tuy nhiên, có một điểm cần lưu ý: enum được sử dụng trong case chỉ cần khai báo giá trị chẳng hạn như "SPRING", không cần đầy đủ là "Season.SPRING". Ví dụ giả sử rằng **currentSeason** là một biến kiểu **Season**, có thể có câu lệnh **switch** sau:

```
switch ( currentSeason ) {
   case WINTER:
      System.out.println("December, January, February");
      break;
   case SPRING:
```

```
      System.out.println("March, April, May");
      break;
   case SUMMER:
      System.out.println("June, July, August");
      break;
   case FALL:
      System.out.println("September, October, November");
      break;
}
```

Câu lệnh switch mới

Một phiên bản mới của câu lệnh **switch mới** đã được thêm vào **Java 14**. Cú pháp như sau:

```
switch (biểu thức) {
   case hằng-số-1 -> Câu lệnh 1;
   case hằng-số-1 -> Câu lệnh 2;
   …
   case hằng-số-n -> Câu lệnh n;
   default -> Câu lệnh n+1;
}
```

Ví dụ:

```
switch ( N ) {
   case 1 -> System.out.println("Số bằng 1");
   case 2, 4, 8 -> {
      System.out.println("(Số là lũy thừa của 2)");
   }
   case 3, 6, 9 -> {
      System.out.println("Số chia hết cho 3");
   }
   case 5,7 -> System.out.println("Số là 5 hoặc 7");
   default ->
      System.out.println("Số không nằm trong khoảng từ 1 đến 9");
}
```

Phiên bản mới sử dụng toán tử **"mũi tên"** thay cho dấu hai chấm sau mỗi trường hợp và mã trong một trường hợp là một câu lệnh đơn hoặc là một câu lệnh khối gồm một số câu lệnh được đặt trong dấu ngoặc nhọn. Không cần câu lệnh **break**, mặc dù câu lệnh break có thể được sử dụng để kết thúc sớm một trường hợp. Điều này tránh được lỗi phổ biến do **break bị quên**. Hơn nữa, thay vì chỉ cho phép một giá trị trên mỗi case, nó có thể kiểm tra một số giá trị được phân tách bằng dấu phẩy.

Cú pháp của **switch** mới cũng có thể được viết dưới dạng biểu thức:

```
String computerMove = switch ( (int)(3*Math.random()) ) {
    case 1 -> "Rock";
    case 2 -> "Paper";
    default -> "Scissors";
};
```

Một **biểu thức switch** phải luôn tính toán một giá trị và do đó hầu như sẽ luôn có **default**. Biểu thức trong một case có thể được thay thế bằng một khối chứa một số câu lệnh; giá trị cho case đó sau đó phải được chỉ định bằng câu lệnh **yield** (chẳng hạn như "yield 42; ") thay vì câu lệnh trả về return;

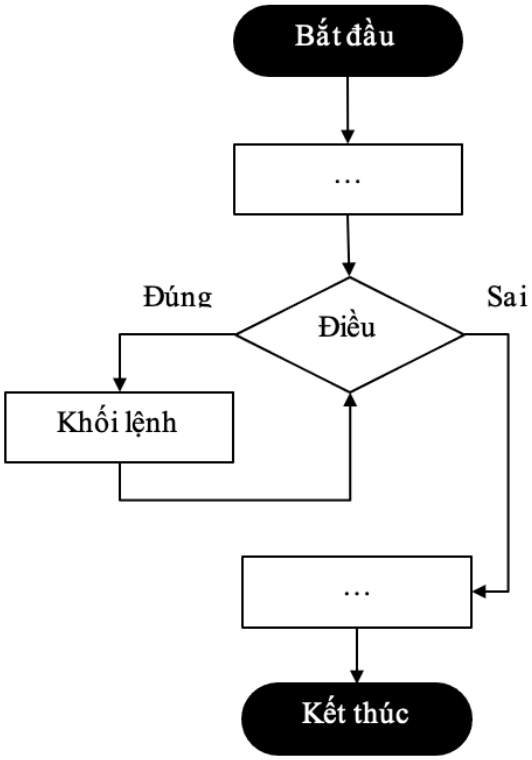
VÒNG LẶP

Trong lập trình, chúng ta cần thực hiện các một số các thao tác được **lặp đi lặp lại nhiều lần.** Lấy ví dụ về việc sắp xếp một mảng các số nguyên. Thuật toán được biết đến nhiều nhất để thực hiện điều này là "bubble sort" hay còn gọi là thuật toán nổi bọt. Thuật toán so sánh phần tử đầu tiên của một mảng với các phần tử còn lại, nếu chúng không theo đúng thứ tự, thì thực hiện đổi chỗ hai phần tử cho nhau. Thao tác này lặp đi lặp lại trong mảng cho đến khi không cần hoán đổi nữa. Trong các ngôn ngữ lập trình, các thuật toán thường được giải quyết bằng **vòng lặp**.

Java cung cấp ba loại vòng lặp là **while**, **do while** và **for**, trong đó vòng lặp for dường như được sử dụng phổ biến hơn cả.

Vòng lặp while

Vòng lặp **while** cho phép thực hiện một khối lệnh nhiều lần, sơ đồ như sau:



Hình 3-5 Sơ đồ khối vòng lặp while

Vòng lặp **while** có thể viết như sau:

```
1  while (biểu thức){
2      khối lệnh
3  }
```

Khối lệnh được gọi là phần thân của vòng lặp. Phần thân của vòng lặp được lặp lại miễn là **biểu thức** còn đúng. Biểu thức này còn được gọi là **biểu thức điều kiện** hay **phép thử** của vòng lặp.

Có một số điểm có thể cần được làm rõ:

- Điều gì xảy ra nếu **điều kiện sai ngay từ đầu**, trước khi phần thân của vòng lặp được thực thi dù chỉ một lần? Trong trường hợp đó, phần thân của vòng lặp không bao giờ được thực thi.
- Điều gì xảy ra nếu **điều kiện là đúng, nhưng lại trở thành sai** ở đâu đó ở giữa thân vòng lặp? Vòng lặp có kết thúc ngay sau khi điều này xảy ra không? Không, bởi vì máy tính tiếp tục thực hiện phần thân của vòng lặp cho đến khi nó kết thúc. Chỉ sau đó, mới nhảy trở lại phần đầu của vòng lặp và kiểm tra điều kiện, khi đó vòng lặp mới có thể kết thúc.

Xét ví dụ sử dụng vòng lặp để tính tổng trung bình của các số:

```
public static void main(String[] args) {
   int inputNumber;  //Một trong các số nguyên do người dùng nhập.
   int sum;          //Tổng các số nguyên dương.
   int count;        //Số lượng các số nguyên dương.
   double average;   //Giá trị trung bình của các số nguyên dương.
   Scanner scanner = new Scanner(System.in);
   /* Khởi tạo biến tổng và biến đếm. */
   sum = 0;
   count = 0;
   /* Đọc và xử lý đầu vào của người dùng. */
   System.out.print("Nhập số nguyên dương đầu tiên: ");
   inputNumber = scanner.nextInt();
   while (inputNumber != 0) {
       sum += inputNumber;
       count++;
       System.out.print("Nhập số nguyên dương tiếp theo (nhập số 0 nếu muốn kết thú
c):");
       inputNumber = scanner.nextInt();
   }
   /* Hiển thị kết quả. */
   if (count == 0)
       System.out.println("Bạn không nhập bất kỳ số nào");
   else {
       average = ((double)sum) / count;
       System.out.println();
       System.out.println("Bạn đã nhập " + count + " số nguyên dương.");
       System.out.printf("Trung bình các số là %1.3f.\n", average);
   }
}
```

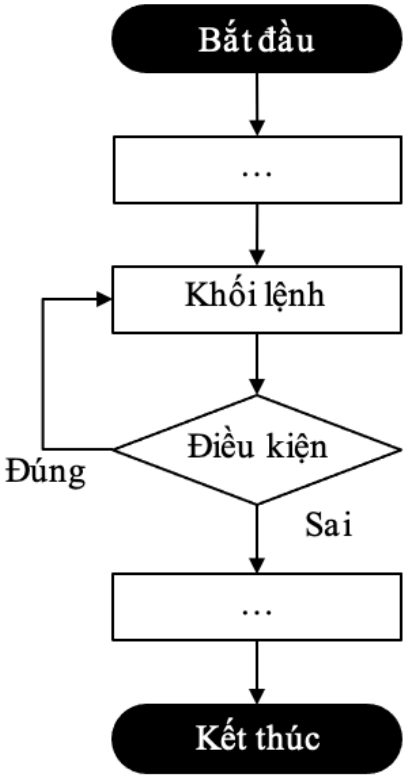
Kết quả khi chạy:

```
Nhập số nguyên dương đầu tiên: 2
Nhập số nguyên dương tiếp theo (nhập số 0 nếu muốn kết thúc):7
Nhập số nguyên dương tiếp theo (nhập số 0 nếu muốn kết thúc):4
Nhập số nguyên dương tiếp theo (nhập số 0 nếu muốn kết thúc):9
Nhập số nguyên dương tiếp theo (nhập số 0 nếu muốn kết thúc):0
Bạn đã nhập 4 số nguyên dương.
Trung bình các số là 5.500.
```

Vòng lặp **while** thường được sử không khi không biết số lần lặp, ví dụ trong trường hợp trên người lập trình sẽ không biết cần thực hiện bao nhiêu lần lặp.

Vòng lặp do ... while

Vòng lặp **do while** cho phép thực hiện một khối lệnh nhiều lần tương tự vòng lặp while tuy nhiên điều kiện sẽ được kiểm tra sau, sơ đồ như sau:



Hình 3-6 Sơ đồ khối lệnh do while

Câu lệnh do... while có cú pháp như sau

```
do
{
    Câu lệnh
}
while (biểu thức);
```

Đôi khi, việc kiểm tra điều kiện tiếp diễn ở **cuối vòng lặp,** thay vì ở đầu sẽ thuận tiện hơn như được thực hiện trong vòng lặp **while**. Câu lệnh **do.. while** rất giống với câu lệnh while, ngoại trừ từ "while", cùng với điều kiện mà nó kiểm tra, đã được **chuyển đến cuối**. Từ "do" được thêm vào để đánh dấu sự bắt đầu của vòng lặp.

Để thực hiện một vòng lặp do...while, trước tiên máy tính thực thi phần thân của vòng lặp, tức là câu lệnh hoặc các câu lệnh bên trong vòng lặp và sau đó đánh giá **biểu thức**. Nếu giá trị của biểu thức là **đúng**, máy tính sẽ quay lại phần đầu của vòng lặp do và lặp lại quá trình; nếu giá trị là **sai**, kết thúc vòng lặp và tiếp tục với phần tiếp theo của chương trình. Vì điều kiện không được kiểm tra cho đến khi kết thúc vòng lặp, phần thân của vòng lặp do **luôn được thực thi ít nhất một lần**.

Hãy lưu ý là có dấu chấm phẩy ';' ở cuối. Dấu chấm phẩy này là một phần của câu lệnh, cũng giống như dấu chấm phẩy ở cuối câu lệnh gán hoặc phần khai báo là một phần của câu lệnh. Bỏ qua “;” sẽ gây ra một lỗi cú pháp. Nói chung, mọi câu lệnh trong Java đều kết thúc bằng dấu chấm phẩy hoặc dấu ngoặc nhọn bên phải “}”

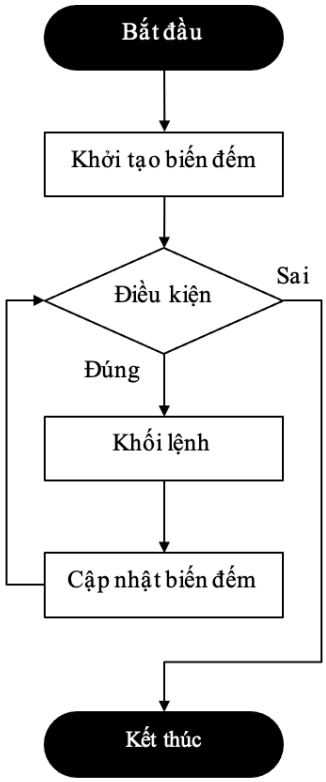
Xem xét một ví dụ khác:

```
boolean wantToContinue; // Đúng nếu người dùng muốn chơi lại.
do {
   //Play Game ;
   System.out.print("Bạn có muốn chơi lại không?");
   wantToContinue = new Scanner(System.in).nextInt();
} while (wantToContinue == true);
```

Giả sử chương trình cho phép người dùng chơi trò chơi và hỏi người chơi có muốn tiếp tục chơi hay không mỗi khi kết thúc một màn chơi (có thể thắng hoặc thua). Với việc kiểm tra **while (wantToContinue == true)** có đúng không cũng giống như việc kiểm tra xem "wantToContinue" có đúng hay không. Chú ý biểu thức logic này cũng có thể viết ngắn gọn lại thành **while (wantToContinue)**

Vòng lặp for

Vòng lặp **for** cũng cho phép thực hiện việc kiểm tra điều kiện và thực hiện khối lệnh nhiều lần. Nó thường đi kèm với các **biến đếm** (counter).



Hình 3-7 Sơ đồ khối vòng lặp for

Cú pháp của vòng lặp for như sau:

```
for(Khởi tạo; Biểu thức; Cập nhật){
      Khối lệnh;
}
```

Trong câu lệnh **for**, điều kiện phải là một biểu thức có giá trị đúng hoặc sai. Việc **khởi tạo** (initialization) thường là một khai báo hoặc một câu lệnh gán, nhưng nó có thể là bất kỳ câu lệnh nào như một câu lệnh trong một chương trình. **Cập nhật** biến đếm thường là một câu lệnh tăng, giảm biến đếm tuy nhiên vị trí này cũng có thể đặt bất kỳ câu lệnh nào. Giống như câu lệnh hay while, **Biểu thức** sẽ được kiểm tra trước khi chạy khối lệnh. Nếu biểu thức còn **đúng** thì khối lệnh còn **được thực hiện.**

Cần lưu ý rằng, bất kỳ phần nào trong ba phần đều có thể để trống. Nếu điều kiện tiếp tục trống, nó được coi như là "**true**", vì vậy vòng lặp sẽ được lặp lại mãi mãi hoặc cho đến khi nó kết thúc vì một số lý do khác, chẳng hạn như câu lệnh break. (Một số người thích bắt đầu vòng lặp vô hạn bằng "**for (;;)** " thay vì "**while (true)**".

Vòng lặp **for** được sử dụng phổ biến nhất là **vòng lặp đếm**, trong đó một biến điều khiển vòng lặp nhận tất cả các giá trị nguyên giữa một số giá trị nhỏ nhất và một số giá trị lớn nhất. Một vòng lặp đếm có dạng:

```
for (biến = min ; biến <= max ; biến ++) {
       câu lệnh
}
```

Trong đó **min** và **max** là các biểu thức có giá trị nguyên (thường là hằng số). Biến nhận các giá trị min, min + 1, min + 2, ..., max. Giá trị của biến điều khiển vòng lặp thường được sử dụng trong phần thân của vòng lặp. Ví dụ dưới đây sẽ in các số từ 0 đến 4:

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
      System.out.println(i);
    }
  }
}
```

Kết quả hiển thị:

```
0
1
2
3
4
```

Vì nhiều lý do khác nhau, các lập trình viên Java thích bắt đầu đếm ở 0 thay vì 1 và họ có xu hướng sử dụng "<" trong điều kiện, thay vì "<=". Thật dễ dàng để đếm ngược với vòng lặp for. Chỉ cần bắt đầu với 10, giảm biến đếm thay vì tăng nó và tiếp tục miễn là biến lớn hơn hoặc bằng 1. Mã lệnh sẽ viết như sau:

```
for ( N = 10 ;  N >= 1 ;  N-- )
   System.out.println( N );
```

Trên thực tế, cú pháp chính thức của câu lệnh for thực sự cho phép cả phần khởi tạo (initialization) và phần cập nhật (update) bao gồm một số biểu thức, được phân tách bằng dấu phẩy. Vì vậy, thậm chí có thể đếm lên từ 1 đến 10 và đếm ngược từ 10 đến 1 cùng một lúc. Ví dụ:

```
for ( i=1, j=10;  i <= 10;  i++, j-- ){
   System.out.printf("%5d", i);
   System.out.printf("%5d", j);
   System.out.println();
}
```

Xét ví dụ khác sử dụng vòng lặp for để in ra bảng chữ cái

```
char ch;
for (ch = 'A'; ch <= 'Z'; ch ++)
    System.out.print (ch);
System.out.println();
```

Vòng lặp lồng nhau

Các cấu trúc điều khiển có thể **chứa** các cấu trúc điều khiển bên trong nó. Ví dụ câu lệnh **if bên trong vòng lặp**, hay một **vòng lặp while bên trong vòng lặp while** khác. Nên có thể nói rằng một cấu trúc này được **lồng** vào bên trong cấu trúc khác, thậm chí có thể có nhiều cấp độ lồng nhau. Cú pháp của Java không đặt giới hạn về số lượng cấp độ lồng nhau. Hãy xem xét một ví dụ in ra một bảng cửu chương như sau:

```
 1  1   2   3   4   5   6   7   8   9  10  11  12
 2   2   4   6   8  10  12  14  16  18  20  22  24
 3   3   6   9  12  15  18  21  24  27  30  33  36
 4   4   8  12  16  20  24  28  32  36  40  44  48
 5   5  10  15  20  25  30  35  40  45  50  55  60
 6   6  12  18  24  30  36  42  48  54  60  66  72
 7   7  14  21  28  35  42  49  56  63  70  77  84
 8   8  16  24  32  40  48  56  64  72  80  88  96
 9   9  18  27  36  45  54  63  72  81  90  99 108
10  10  20  30  40  50  60  70  80  90 100 110 120
11  11  22  33  44  55  66  77  88  99 110 121 132
12  12  24  36  48  60  72  84  96 108 120 132 144
```

Dữ liệu trong bảng được sắp xếp thành 12 hàng và 12 cột. Có thể viết mã sử dụng vòng lặp for lồng nhau như sau để tạo ra bảng trên:

```
for (int rowNumber = 1;  rowNumber <= 12;  rowNumber++ ) {
   for (int N = 1;  N <= 12;  N++ ) {
      System.out.printf( "%4d", N * rowNumber );
   }
   System.out.println();
}
```

BREAK VÀ CONTINUE

Cú pháp của vòng lặp **while** và **do... while** cho phép kiểm tra điều kiện tiếp tục lặp ở đầu vòng lặp hoặc ở cuối vòng lặp. Đôi khi, việc kiểm tra ở **trong vòng lặp** lại phù hợp hơn. Java cung cấp một phương pháp để thoát ra khỏi vòng lặp từ một vị trí bất kỳ trong thân vòng lặp bằng cách sử dụng lệnh **break** và **continue**.

- Khi gặp lệnh **break** trong một vòng lặp, nó sẽ ngay lập tức nhảy ra khỏi vòng lặp.
- Khi gặp câu lệnh **continue** nó sẽ quay trở lại kiểm tra **biểu thức điều kiện** mà không cần thực hiện các câu lệnh tiếp theo trong vòng lặp đó.

Câu lệnh break

Câu lệnh **break** được sử dụng để thoát ra khỏi vòng lặp. Ví dụ:

```
1  public static void main(String[] args) {
2     for (int i = 0; i < 10; i++) {
3         if (i == 4) {
4             break;
5         }
6         System.out.println(i);
7     }
8  }
```

Ví dụ trên dừng vòng lặp khi i = 4. Kết quả khi chạy sẽ là:

```
1  0
2  1
3  2
4  3
```

Nếu sử dụng câu lệnh **break** bên trong một vòng lặp lồng nhau, nó sẽ chỉ thoát ra khỏi vòng lặp đó mà không thoát ra khỏi vòng lặp bên ngoài. Để **thoát ra khỏi vòng lặp lồng nhau** từ bên trong, cần sử dụng **break kèm theo nhãn**, cú pháp như sau:

```
break nhãn;
```

Trong đó nhãn được gán cho vòng lặp bên ngoài. Xét ví dụ một đoạn mã kiểm tra xem hai chuỗi, s1 và s2, có một ký tự chung hay không:

```
1  String s1 = "Lap trinh Java";
2  String s2 = "win";
3  boolean nothingInCommon = true; // Giả sử s1 và s2 không có ký tự chung
4  int i = 0, j; // Các biến để lặp qua các ký tự trong s1 và s2
5  bigloop: while (i < s1.length()) {
6     j = 0;
7     while (j < s2.length()) {
8          if (s1.charAt(i) == s2.charAt(j)) {// s1 và s2 có ký tự chung
9              nothingInCommon = false;
10             System.out.println(s1.charAt(i));
11             break bigloop; // thoát ra khỏi CẢ HAI vòng lặp
12         }
13         j++; // Đi tới char tiếp theo trong s2
14      }
15    i++; // Đi tới char tiếp theo trong s1
16 }
17 System.out.println(nothingInCommon);
```

Nếu một ký tự chung được tìm thấy, giá trị của biến **nothingInCommon** được đặt thành false và việc tìm kiếm ký tự chung sẽ kết thúc luôn. Câu lệnh break này cơ chế hoạt động tương tự như lệnh **goto** trong

một số ngôn ngữ lập trình như C hay C++. Cần lưu ý rằng **goto** cũng là một từ khóa trong Java tuy nhiên không được sử dụng như một lệnh.

Câu lệnh continue

Câu lệnh **continue** dùng để bỏ qua một lượt lặp. Ví dụ:

```
1  for (int i = 0; i < 10; i++) {
2     if (i == 4) {
3       continue;
4     }
5     System.out.println(i);
6  }
```

Kết quả:

```
1  0
2  1
3  2
4  3
5  5
6  6
7  7
8  8
9  9
```

Lệnh **continue** có liên quan đến **break**, tuy nhiên, thay vì nhảy ra khỏi vòng lặp hoàn toàn, nó sẽ nhảy trở lại phần đầu của vòng lặp và tiếp tục với lần lặp tiếp theo (bao gồm cả việc đánh giá điều kiện tiếp tục của vòng lặp để xem liệu có cần lặp lại thêm nữa không). Như vậy, câu lệnh **continue** yêu cầu máy tính bỏ qua phần còn lại của lần lặp hiện tại của vòng lặp. Tương tự như với **break**, khi **continue** nằm trong một vòng lặp lồng nhau, nó sẽ tiếp tục lần lặp trực tiếp chứa nó; thay vào đó, một "**continue nhãn**" có thể được sử dụng để tiếp tục lần lặp của vòng lặp bên ngoài.

MẢNG

Định nghĩa mảng

Mảng là một **cấu trúc dữ liệu** là một tập hợp các phần tử dữ liệu cùng kiểu (số, chữ, đối tượng...) được khai báo chung vào một biến.

Trong Java tất cả các phần tử sẽ được **đánh chỉ số** (bắt đầu bằng 0). Kiểu dữ liệu của một mảng có thể là bất kỳ kiểu dữ liệu hợp lệ nào trong Java, nếu đó là kiểu dữ liệu cơ sở int, thì nó được gọi là "**mảng int**". Một mảng có kiểu dữ liệu là String được gọi là "**mảng chuỗi**".

Truy cập vào vào phần tử của mảng thông qua chỉ số. Khi sử dụng một mảng trong một chương trình, có thể sử dụng một biến để tham chiếu đến toàn bộ mảng. Nhưng để tham chiếu đến các phần tử riêng lẻ của mảng cần tên mảng và số chỉ số của phần tử. Ví dụ, cú pháp để tham chiếu đến một phần tử như sau: **namelist[7].** Ở đây, namelist là biến đặt tên cho toàn bộ mảng và **namelist[7]** đề cập đến phần tử ở chỉ số 7 trong mảng đó. Nghĩa là, để tham chiếu đến một phần tử của mảng, sử dụng tên mảng, theo sau là chỉ số phần tử được đặt trong dấu ngoặc vuông.

Tạo và sử dụng mảng

Mảng được tạo bằng từ khóa **new**, ví dụ tạo một mảng String gồm 100 phần tử như sau:

```
String[] array = new String[100];
```

Trước khi có thể sử dụng một biến để tham chiếu đến một mảng, biến đó phải được khai báo và nó phải có kiểu. Ví dụ, đối với một **mảng chuỗi**, kiểu cho biến mảng sẽ là String[] và đối với một mảng int, nó sẽ là int[]. Nói chung, kiểu mảng bao gồm kiểu của mảng theo sau là một cặp dấu ngoặc vuông trống. Mảng được tạo luôn kèm theo kiểu và kích thước của mảng. Độ dài của mảng có thể được cung cấp dưới dạng số nguyên hoặc biểu thức có giá trị số nguyên. Ví dụ, sau câu lệnh gán "**A = new int [5];**", A là một mảng chứa năm phần tử nguyên A[0], A[1], A[2], A[3] và A[4]. Có thể lấy ra độ dài của danh sách mảng bằng cách truy cập đến thuộc tính **length** của mảng. Ví dụ để in ra độ dài của mảng A:

```
System.out.println(A.length);
```

Khi khởi tạo mảng, các phần tử của mảng sẽ được cấp phát không gian trong bộ nhớ. Các phần tử luôn chứa giá trị mặc định. Ví dụ, khi tạo một mảng int, mỗi phần tử của mảng sẽ tự động được khởi tạo giá trị bằng **0**. Một mảng boolean, các phần tử sẽ có giá trị mặc định là **false**. Và một mảng ký tự giá trị mặc định sẽ là **null**.

Duyệt mảng

Mảng thường được xử lý với vòng lặp **for**. Ví dụ, để in ra tất cả các phần tử trong một mảng:

```
1  int i;
2  for (i = 0; i < list.length; i++) {
3      System.out.println( list[i] );
4  }
```

Lần lặp đầu tiên, i là 0 và list[i] tham chiếu đến list[0]. Vì vậy, giá trị list[0] được in ra. Lần thứ hai qua vòng lặp, i là 1 và giá trị được lưu trong mảng là list[1] được in ra. Nếu độ dài của danh sách là 5, thì vòng lặp kết thúc sau khi in giá trị của list[4], khi i bằng 5 và điều kiện tiếp tục **i <list.length** không còn đúng nữa. Đây là một ví dụ điển hình của việc sử dụng vòng lặp **for** để xử lý một mảng.

Hãy xem thêm một vài ví dụ. Giả sử rằng A là một **mảng các số thực** và chúng ta muốn tìm giá trị trung bình của tất cả các phần tử của mảng. Chúng ta có thể sử dụng vòng lặp for để cộng các số, sau đó chia cho độ dài của mảng để lấy giá trị trung bình:

```
1  public static void main(String[] args) {
2     double arr[] = {64,25,73,64};
3     double total;
4     double average;
5     int i;
6     total = 0;
7     for ( i = 0; i < arr.length; i++ ) {
8         total = total + arr[i];
9     }
10    average = total / arr.length;
```

```
11      System.out.println(average);
12  }
```

Ngoài ra, có thể thao tác với mảng bằng vòng lặp "**foreach**", cú pháp vòng lặp **foreach** như sau:

```
1  for (double e : arr) {
2  }
```

Với mỗi lần lặp vòng lặp sẽ trả về một phần tử của mảng. Ví dụ, tìm phần tử lớn nhất trong mảng:

```
1  public static void main(String[] args) {
2      double arr[] = {64, 25, 73, 64};
3      double max = arr[0];   // At first, the largest number seen is A[0].
4      for (double e : arr) {
5          if (e > max) max = e;
6      }
7      System.out.println(max);
8  }
```

Sắp xếp mảng

Chương trình sau sắp xếp mảng:

```
1  public static void main(String[] args) {
2      int temp;
3      int[] a = {4, 5, 3, 7, -3, 9, 7};
4      for (int i = 0; i < a.length - 1; i++) {
5          for (int j = i; j <= a.length - 1; j++) {
6              if (a[i] > a[j]) {
7                  temp = a[i];
8                  a[i] = a[j];
9                  a[j] = temp;
10             }
11         }
12     }
13     for (int i = 0; i < a.length - 1; i++) {
14         System.out.print(a[i] + " ");
15     }
16 }
```

Thực hiện chương trình

```
-3 3 4 5 7 7
```

Khi làm việc với mảng trong Java có một lớp Arrays. Lớp này cung cấp rất nhiều tiện ích đối với mảng, ví dụ việc sắp xếp một mảng và in mảng đó ra được thực hiện như sau:

```
1  int[] a = {4, 5, 3, 7, -3, 9, 7};
2  Arrays.sort(a);
3  Arrays.stream(a).forEach(System.out::println);
```

Mảng hai chiều

Trong mảng **hai chiều**, hoặc "2D", các phần tử có thể được sắp xếp theo hàng/cột. Ở đây, ví dụ, là một mảng 2D các phần tử int có **năm hàng** và **bảy cột**:

	0	1	2	3	4	5	6
0	58	38	5	58	-5	56	9
1	5	8	15	2	88	-25	69
2	-39	7	-11	55	82	7	37
3	75	57	7	15	58	-8	58
4	-85	1	-9	88	8	23	38

Mảng 5 x 7 này chứa tổng cộng 35 phần tử. Khai báo mảng này trong Java viết như sau:

```
int[][] arr = new int[5][7];
```

Mảng hai chiều thường được xử lý bằng cách sử dụng các vòng lặp for lồng nhau. Ví dụ đoạn mã sau sẽ in ra các phần tử của mảng arr:

```
1  for (int row = 0; row < 5; row++ ) {
2     for ( int col = 0; col < 7; col++ ) {
3         System.out.printf( "%7d",  arr[row][col] );
4     }
5     System.out.println();
6  }
```

Mảng hai chiều có nhiều tác dụng trong thiết kế ứng dụng. Ví dụ, một mảng 2D có thể được sử dụng để lưu trữ nội dung của bàn cờ trong một trò chơi như cờ vua hoặc cờ caro. Mảng 2D cũng có thể sử dụng mảng để giữ màu của lưới các ô vuông có màu.

Hãy xem xét một công ty sở hữu 25 cửa hàng. Giả sử rằng công ty có dữ liệu về lợi nhuận thu được tại mỗi cửa hàng cho mỗi tháng trong năm 2018. Nếu các cửa hàng được đánh số từ 0 đến 24 và nếu mười hai tháng từ tháng 1 năm 2018 đến tháng 12 năm 2018 được đánh số từ 0 đến 11, thì dữ liệu lợi nhuận có thể được lưu trữ trong một mảng, lợi nhuận, được tạo như sau:

```
double[][]  profit =  new double[25][12];
```

Khi đó, profit[storeNum][monthNum] sẽ là số lợi nhuận kiếm được từ storeNum trong tháng monthNum. Ví dụ profit[3][2] sẽ là số lợi nhuận kiếm được tại cửa hàng số 3 trong tháng 3. Với cách lưu trữ này có thể dễ dàng tính ra tổng lợi nhuận của công ty - cho cả năm từ tất cả các cửa hàng như sau:

```
1  double totalProfit;
2  int store, month;
3  totalProfit = 0;
4  for ( store = 0; store < 25; store++ ) {
```

```
5     for ( month = 0; month < 12; month++ )
6        totalProfit += profit[store][month];
7  }
```

Đôi khi cần phải xử lý một hàng hoặc một cột duy nhất của mảng chứ không phải toàn bộ mảng. Ví dụ để tính tổng lợi nhuận mà công ty kiếm được trong tháng 12 (tức có chỉ số cột là 11) có thể sử dụng vòng lặp như sau:

```
1  double decemberProfit;
2  int storeNum;
3  decemberProfit = 0.0;
4  for ( storeNum = 0; storeNum < 25; storeNum++ ) {
5     decemberProfit += profit[storeNum][11];
6  }
```

THỰC HÀNH

Bài toán quy đổi điểm

Bài thực hành sau sẽ viết chương trình để thực hiện việc quy đổi điểm. Đầu tiên hãy tạo lớp **Mark**:

```
1  public class Mark{
2     public static void main(String[] args){
3         int testscore = 76;
4         char grade;
5         if (testscore >= 90) {
6             grade = 'A';
7         } else if (testscore >= 80) {
8             grade = 'B';
9         } else if (testscore >= 70) {
10            grade = 'C';
11        } else if (testscore >= 60) {
12            grade = 'D';
13        } else {
14            grade = 'F';
15        }
16        System.out.println("Grade = " + grade);
17    }
18 }
```

Thực hiện chương trình, kết quả:

```
C
```

Bài toán tháng trong năm

Hãy thực hành với câu lệnh switch sau và chuyển nó sang sử dụng cú pháp mới.

```
1  public class Month{
2     public static void main(String[] args){
3         System.out.print("Enter a month:");
```

```
        Scanner scanner=new Scanner(System.in);
        int month=scanner.nextInt();
        String monthString;
        switch (month)
        {
            case 1:  monthString = "January";      break;
            case 2:  monthString = "February";     break;
            case 3:  monthString = "March";        break;
            case 4:  monthString = "April";        break;
            case 5:  monthString = "May";          break;
            case 6:  monthString = "June";         break;
            case 7:  monthString = "July";         break;
            case 8:  monthString = "August";       break;
            case 9:  monthString = "September";    break;
            case 10: monthString = "October";      break;
            case 11: monthString = "November";     break;
            case 12: monthString = "December";     break;
            default: monthString = "Invalid month"; break;
        }
        System.out.println(monthString);
    }
}
```

Chạy chương trình, nhập vào tháng, và xem kết quả:

```
Enter a month:6
June
```

Menu lựa chọn bằng câu lệnh switch

Trong chương trình dòng lệnh, menu có thể được trình bày dưới dạng danh sách các tùy chọn được đánh số và người dùng có thể chọn một tùy chọn bằng cách nhập vào số của nó. Một ứng dụng của câu lệnh switch là trong các menu xử lý. Menu là một danh sách các tùy chọn. Người dùng chọn một trong các tùy chọn. Máy tính phải trả lời từng lựa chọn có thể theo một cách khác nhau như trong bài thực hành sau:

```
int optionNumber;
double measurement;
double inches = 0;
Scanner scanner = new Scanner(System.in);
System.out.println("Đơn vị đo muốn sử dụng?");
System.out.println("         1.  inches");
System.out.println("         2.  feet");
System.out.println("         3.  yards");
System.out.println("         4.  miles");
System.out.println("Lựa chọn 1-4: ");
optionNumber = scanner.nextInt();
switch (optionNumber) {
   case 1:
       System.out.println("Nhập độ dài theo inches: ");
       measurement = scanner.nextDouble();
       inches = measurement;
```

```
        break;
    case 2:
        System.out.println("Nhập độ dài theo feet: ");
        measurement = scanner.nextDouble();
        inches = measurement * 12;
        break;
    case 3:
        System.out.println("Nhập độ dài theo yards: ");
        measurement = scanner.nextDouble();
        inches = measurement * 36;
        break;
    case 4:
        System.out.println("Nhập độ dài theo miles: ");
        measurement = scanner.nextDouble();
        inches = measurement * 12 * 5280;
        break;
    default:
        System.out.println("Cần chọn 1-4");
}
System.out.println("Kích thước: " + inches + " inches");
```

Hãy thực hiện chương trình trên và giải thích cách thức vận hành của chương trình

Đảo ngược một chuỗi.

Tạo ra một lớp để đảo ngược lại chuỗi:

```
public class ReverseString {
    public static void main(String[] args) {
        String palindrome = "Lap trinh Java";
        int len = palindrome.length();
        char[] tempCharArray = new char[len];
        char[] charArray = new char[len];
        // put original string in an array of chars
        for (int i = 0; i < len; i++) {
            tempCharArray[i] = palindrome.charAt(i);
        }
        // reverse array of chars
        for (int j = 0; j < len; j++) {
            charArray[j] = tempCharArray[len - 1 - j];
        }
        String reversePalindrome = new String(charArray);
        System.out.println(reversePalindrome);
    }
}
```

Hãy thực hiện chương trình trên và giải thích cách thức vận hành của chương trình

Mảng chuỗi

Viết chương trình nhập vào một số và trả về thứ tương ứng trong tuần:

```java
public class DayofWeek{
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        String[] calendarDays = { "Monday", "Tuesday", "Wednesday",  "Thursday", "Friday", "Saturday", "Sunday" };
        System.out.print("Input day of week:");
        int dayofweek = scanner.nextInt();
        System.out.println("It is " + calendarDays[dayofweek - 1]);
    }
}
```

Vòng lặp for

Bài toán đếm ước số của một số.

```java
public static void main(String[] args){
    long N ;
    long testDivisor;
    long divisorCount  = 0;
    long numberTested;
    while (true) {
        System.out.print("Enter a positive integer: ");
        N = new Scanner(System.in).nextLong();
        if (N > 0)
            break;
        System.out.println("That number is not positive.  Please try again.");
    }
    divisorCount = 0;
    numberTested = 0;
    for (testDivisor = 1; testDivisor <= N; testDivisor++) {
        if ( N % testDivisor == 0 )
            divisorCount++;
        numberTested++;
        if (numberTested == 10000000) {
            System.out.print('.');
            numberTested = 0;
        }
    }
    System.out.println();
    System.out.println("The number of divisors of " + N + " is " + divisorCount);
}
```

Đếm ký tự

Viết chương trình nhập từ bàn phím một chuỗi không quá 80 ký tự và một ký tự bất kỳ. Đếm và in ra màn hình số lần xuất hiện của ký tự đó trong chuỗi vừa nhập.

```java
public static void main(String[] args) {
     String chuoi;
     char kyTu;
     int count = 0;
     Scanner scanner = new Scanner(System.in);
```

```
6
7       // nếu độ dài chuỗi nhập vào còn lớn hơn 80 thì phải nhập lại
8       do {
9           System.out.println("Nhập vào 1 chuỗi bất kỳ: ");
10          chuoi = scanner.nextLine();
11      } while (chuoi.length() > 80);
12
13      System.out.println("Nhập vào ký tự cần đếm số lần xuất hiện: ");
14      kyTu = scanner.next().charAt(0);
15
16      /*
17       * đếm và in ra số lần xuất hiện của ký tự đó trong chuỗi
18       * duyệt từ đầu đến cuối chuỗi
19       * nếu có ký tự nào tại vị trí i bằng với ký tự ch thì tăng biến count lên 1
20       */
21      for (int i = 0; i < chuoi.length(); i++) {
22          if (kyTu == chuoi.charAt(i)) {
23              count++;
24          }
25      }
26
27      System.out.println("Số lần xuất hiện của ký tự " + kyTu +
28          " trong chuỗi " + chuoi + " = " + count);
29  }
```

Kết quả sau khi biên dịch chương trình:

```
1  Nhập vào 1 chuỗi bất kỳ:
2  Welcome to Java!
3  Nhập vào ký tự cần đếm số lần xuất hiện:
4  a
5  Số lần xuất hiện của ký tự a  trong chuỗi Welcome to Java! = 2
```

Viết chương trình nhập vào một chuỗi bất kỳ bao gồm cả số, ký tự thường và ký tự hoa từ bàn phím. Sau đó đếm và in ra số ký tự thường và ký tự hoa và số có trong chuỗi đó.

```
1  public static void main(String[] args) {
2      String chuoi;
3      int soKyTuInHoa = 0, soKyTuInThuong = 0, soChuSo = 0;
4      Scanner scanner = new Scanner(System.in);
5
6      // nếu độ dài chuỗi nhập vào còn lớn hơn 80 thì phải nhập lại
7      do {
8          System.out.println("Nhập vào 1 chuỗi bất kỳ: ");
9          chuoi = scanner.nextLine();
10     } while (chuoi.length() > 80);
11
12     // đếm và in ra số lần xuất hiện của ký tự đó trong chuỗi
13     // duyệt từ đầu đến cuối chuỗi
14     // nếu có ký tự nào tại vị trí i bằng với ký tự ch thì tăng biến count lên 1
15     for (int i = 0; i < chuoi.length(); i++) {
16         // phương thức isUpperCase() là phương thức dùng để kiểm tra ký tự tại vị tr
i i
```

```
17          // có phải là ký tự in hoa hay không.
18          if (Character.isUpperCase(chuoi.charAt(i))) {
19              soKyTuInHoa++;
20          }
21
22          // phương thức isLowerCase() là phương thức dùng để kiểm tra ký tự tại vị tr
í i
23          // có phải là ký tự in thường hay không.
24          if (Character.isLowerCase(chuoi.charAt(i))) {
25              soKyTuInThuong++;
26          }
27
28          // phương thức isDigit() là phương thức dùng để kiểm tra ký tự tại vị trí i
29          // có phải là số hay không.
30          if (Character.isDigit(chuoi.charAt(i))) {
31              soChuSo++;
32          }
33      }
34
35      System.out.println("Trong chuỗi " + chuoi +
36          " có " + soKyTuInHoa + " ký tự in hoa," +
37          " có " + soKyTuInThuong + " ký tự in thường" +
38          " và có " + soChuSo + " số.");
39  }
```

Kết quả sau khi biên dịch chương trình:

```
1  Nhập vào 1 chuỗi bất kỳ:
2  Welcome to Java!
3  Trong chuỗi Welcome to Java! có 2 ký tự in hoa, có 11 ký tự in thường và có 0 số.
```

CÂU HỎI ÔN TẬP LÝ THUYẾT

1. Đâu là lệnh đúng để tạo một mảng

- A. int arr[] = {2, 4, 5}
- B. int arr[] = {"0","1","2"}
- C. int arr[] = [1, 3, 5]
- D. int arr[] = (1, 2, 3)

2. Đâu là cú pháp chuẩn để khai báo mảng hai chiều

- A. int[] arr; arr = new int[5];

B. int[5][7] arr; arr = new int[][];

C. int[] arr = new int[5][7];

D. int[][] arr; arr = new int[5][7];

3. Từ khóa enum giúp định nghĩa

A. Dữ liệu kiểu mảng

B. Dữ liệu kiểu chuỗi

C. Dữ liệu dạng cấu trúc

D. Dữ liệu dạng liệt kê

4. Biểu thức kiểm tra trong câu lệnh switch không thể chứa giá trị

A. String

B. enum

C. int

D. float

5. Đâu là câu lệnh đúng để khai báo một enum

A. enum Season [SPRING, SUMMER, FALL, WINTER]

B. enum Season "SPRING, SUMMER, FALL, WINTER"

C. enum Season (SPRING, SUMMER, FALL, WINTER)

D. enum Season {SPRING, SUMMER, FALL, WINTER}

6. Muốn thoát ra khỏi vòng lặp ngoài của cấu trúc lặp lồng nhau có thể sử dụng

A. Lệnh continue

B. Lệnh break kèm nhãn

C. Lệnh exit

D. Lệnh stop kèm vị trí kết thúc

7. Vòng lặp kiểm tra điều kiện rồi mới thực hiện lệnh là vòng lặp

A. while

B. do while

C. for

D. loop

8. Hãy cho biết khai báo mảng nào là đúng

```
1  double[] a1 = new double[] { "4.56", 332.267, 7.0, 0.3367, 10.0 };
2  String a2 = new String[] { "a", "b", "c" };
3  Object[] a3 = new Object[] { new Object(), "Love", 3};
```

A. mảng a1

B. mảng a2

C. mảng a3

D. Tất cả các mảng đều khai báo sai cú pháp

BÀI TẬP TỰ THỰC HÀNH

Bài toán tính lương làm thêm

Viết chương trình yêu cầu người dùng nhập vào số giờ làm việc và số tiền trả cho nhân viên mỗi giờ sau đó in ra số tiền phải trả cho nhân viên đó. Hãy hiển thị thông tin cảnh báo cho bài toán trả lương như sau: nếu mức lương được trả thấp hơn mức lương tối thiểu (25 nghìn đồng một giờ) hoặc nếu nhân viên làm việc nhiều hơn số giờ trong một tuần (40).

Ngày trong tuần

> Viết chương trình nhập vào một số (1-7) tương ứng với ngày trong tuần và hiển thị ra kết quả như sau:
>
> 1 : Thứ hai
>
> 2 : Thứ ba
>
> ...
>
> 7 : Chủ nhật

CozaLozaWoza

Viết một chương trình có tên **CozaLozaWoza** để in các số từ 1 đến 100 sao cho có 11 số trên mỗi dòng. Đồng thời chương trình sẽ in ra:
"**Coza**" thay cho các vị trí số là bội số của 3
"**Loza**" thay cho các vị trí số là bội số của 5
"**Woza**" thay cho các vị trí số là bội số của 7
"**CozaLoza**" thay cho các vị trí số là bội số của 3 và 5
Kết quả giống như mô tả sau:

```
1  1 2 Coza 4 Loza Coza Woza 8 Coza Loza 11
2  Coza 13 Woza CozaLoza 16 17 Coza 19 Loza CozaWoza 22
3  23 Coza Loza 26 Coza Woza 29 CozaLoza 31 32 Coza
4  34 LozaWoza Coza 37 38 Coza Loza 41 CozaWoza 43 44
5  CozaLoza 46 47 Coza Woza Loza Coza 52 53 Coza Loza
6  Woza Coza 58 59 CozaLoza 61 62 CozaWoza 64 Loza Coza
7  67 68 Coza LozaWoza 71 Coza 73 74 CozaLoza 76 Woza
8  Coza 79 Loza Coza 82 83 CozaWoza Loza 86 Coza 88
9  89 CozaLoza Woza 92 Coza 94 Loza Coza 97 Woza Coza
```

Gợi ý:

```
 1  public class CozaLozaWoza {
 2     public static void main(String[] args) {
 3     int lowerbound = 1;
 4     int upperbound = 110;
 5     for (int number = lowerbound; number <= upperbound; number++) {
 6             // Print "Coza" if number is divisible by 3
 7             if (......) {
 8             System.out.print("Coza");
 9             }
10             // Print "Loza" if number is divisible by 5
11             if (......) {
12               System.out.print(.....);
13             }
14             // Print "Woza" if number is divisible by 7
```

```
15            ......
16            // Print the number if it is not divisible by 3, 5 and 7
17            if (......) {
18            ......
19            }
20             // Print a space
21            ......
22            // Print a newline if number is divisible by 11
23            if (......) {
24            System.out.println();
25            }
26    }
27     }
28  }
```

Fibonaci

Trong toán học các số Fibonacci là dãy các số nguyên có dạng sau:

```
0 1 1 2 3 5 8 13 24 34 ...
```

Theo định nghĩa, hai số đầu tiên trong dãy Fibonacci là 0 và 1, và mỗi số tiếp theo là tổng của hai số trước đó. Theo thuật ngữ toán học, dãy Fn của các số Fibonacci được xác định bằng quan hệ lặp như sau:

```
Fn = Fn-1 + Fn-2
```

Với giá trị ban đầu

```
F0=0, F1=1
```

Viết chương trình có tên Fibonacci để hiển thị **20 số Fibonacci đầu tiên** bằng cách sử dụng vòng lặp (Không dùng đệ quy).

Mảng ngẫu nhiên

Viết chương trình điền ngẫu nhiên vào mảng 3 x 4 x 6, sau đó in ra giá trị lớn nhất và nhỏ nhất trong mảng

TÀI LIỆU THAM KHẢO

[1] Core Java: Fundamentals (2021) , Cay Horstmann (Oracle Press Java)

[2] Head First Java: A Brain-Friendly Guide (2022), Kathy Sierra, O'Reilly Media

[3] Java OOP Done Right: Create object oriented code you can be proud of with modern Java Paperback (2019), Mr Alan Mellor, Mellor Books

[4] Murach's Java Programming (5th Edition) (2017), Joe Murach, Mike Murach & Associates

[5]. Java for Absolute Beginners Learn to Program the Fundamentals the Java 9+ Way

[6]. Modern Java Recipes: Simple Solutions to Difficult Problems in Java 8 and 9 (2017), by Ken Kousen, O'Reilly Media

[7] Effective Java (2018), Joshua Bloch, Addison-Wesley Professional