Phương thức trong Java

Một phương thức trong Java là một tập hợp các lệnh mà được nhóm cùng với nhau để thực hiện một hành động. Ví dụ khi bạn gọi phương thức System.out.print, hệ thống thực sự thực thi một vài lệnh để hiển thị một thông báo trên bàn điều khiển console.

Bây giờ, bạn sẽ học cách tạo các phương thức cho riêng bạn với hoặc không với các giá trị trả về, gọi một phương thức với hoặc không với các tham số, tải các phương thức sử dụng cùng tên, và áp dụng phương thức trừu tượng trong thiết kế chương trình.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về Phương thức trong Java**.

Tạo phương thức trong Java

Xem xét ví dụ sau để giải thích cú pháp của một phương thức:

```
public static int funcName(int a, int b) {
  // body
}
```

Ở đây:

- **public static** : Là modifier.
- **int**: Kiểu trả về
- **funcName**: Tên hàm
- **a, b**: Các tham số chính thức
- **int a, int b**: Danh sách các tham số

Các phương thức cũng còn được biết như các Procedure (thủ tục) hoặc Function (hàm):

- **Procedure:** Chúng không trả về bất kỳ giá trị nào.
- **Function:** Chúng trả về giá trị.

Sự định nghĩa phương thức bao gồm một header và phần thân phương thức. Tương tự như sau:

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

Cú pháp trên bao gồm:

- **modifier:** Nó định nghĩa kiểu truy cập của phương thức và nó là tùy ý để sử dụng.
- **returnType:** Phương thức có thể trả về một giá trị.
- **nameOfMethod:** Đây là tên phương thức.
- **Parameter List:** Danh sách các tham số, nó là kiểu, thứ tự, và số tham số của một phương thức. Đây là tùy ý, phương thức có thể không chứa tham số nào.
- **method body:** Phần thân phương thức định nghĩa những gì phương thức đó thực hiện với các lệnh.

Ví dụ:

Đây là code nguồn của phương thức max() được định nghĩa ở trên. Phương thức này nhận hai tham số là num1 và num2 và trả về giá trị lớn nhất của hai số:

```
/** the snippet returns the minimum between two numbers */
public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Gọi phương thức trong Java

Để sử dụng một phương thức, nó nên được gọi. Có hai cách để gọi một phương thức, ví dụ: phương thức trả về một giá trị hoặc phương thức không trả về giá trị nào.

Tiến trình gọi phương thức là đơn giản. Khi một chương trình gọi phương thức, điều khiển chương trình truyền tín hiệu tới phương thức được gọi. Phương thức được gọi này sau đó trả về điều khiển tới caller trong hai điều kiện, khi:

- Lệnh return được thực thi.
- Tiến tới dấu ngoặc đóng ở cuối phương thức.

Phương thức trả về void được xem như là gọi tới một lệnh. Bạn xét ví dụ sau:

```
System.out.println("This is tutorialspoint.com!");
```

Ví dụ sau minh họa về phương thức trả về giá trị:

```
int result = sum(6, 9);
```

Ví dụ:

Ví dụ sau minh họa cách định nghĩa một phương thức và cách để gọi nó:

```
public class ExampleMinNumber{

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("Minimum Value = " + c);
   }

   /** returns the minimum of two numbers */
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;
```

```
      return min;
   }
}
```

Nó sẽ cho kết quả sau:

```
Minimum value = 6
```

Từ khóa void trong Java

Từ khóa void cho phép chúng ta tạo các phương thức mà không trả về giá trị nào. Ở đây, trong ví dụ sau, chúng ta xem xét một phương thức void là *methodRankPoints*. Phương thức này là một phương thức void mà không trả về bất kỳ giá trị nào. Gọi tới một phương thức void phải là một lệnh, ví dụ như *methodRankPoints(255.7);*. Nó là một lệnh Java mà kết thúc với dấu chấm phảy như dưới đây.

Ví dụ:

```
public class ExampleVoid {


   public static void main(String[] args) {
      methodRankPoints(255.7);
   }


   public static void methodRankPoints(double points) {
      if (points >= 202.5) {
         System.out.println("Rank:A1");
      }
      else if (points >= 122.4) {
         System.out.println("Rank:A2");
      }
      else {
         System.out.println("Rank:A3");
      }
   }
}
```

Nó sẽ cho kết quả sau:

```
Rank:A1
```

Truyền các tham số bởi giá trị trong Java

Trong khi làm việc dưới tiến trình gọi, các tham số được truyền. Điều này nên trong cùng thứ tự như các tham số tương ứng của chúng trong phương thức. Các tham số có thể được truyền bởi giá trị hoặc bởi tham chiếu.

Truyền các tham số bởi giá trị nghĩa là gọi một phương thức với một tham số. Thông qua điều này, giá trị tham số được truyền tới tham số.

Ví dụ:

Chương trình sau minh họa việc truyền tham số bởi giá trị trong Java. Các giá trị của các tham số vẫn tồn tại giống như vậy cho dù sau lời gọi phương thức đó.

```
public class swappingExample {

   public static void main(String[] args) {
      int a = 30;
      int b = 45;

      System.out.println("Before swapping, a = " +
                         a + " and b = " + b);

      // Invoke the swap method
      swapFunction(a, b);
      System.out.println("\n**Now, Before and After swapping values will be same here**:");
      System.out.println("After swapping, a = " +
                        a + " and b is " + b);
   }

   public static void swapFunction(int a, int b) {

      System.out.println("Before swapping(Inside), a = " + a
```

```
                            + " b = " + b);
      // Swap n1 with n2
      int c = a;
      a = b;
      b = c;

      System.out.println("After swapping(Inside), a = " + a
                            + " b = " + b);
   }
}
```

Nó sẽ cho kết quả:

```
Before swapping, a = 30 and b = 45
Before swapping(Inside), a = 30 b = 45
After swapping(Inside), a = 45 b = 30

**Now, Before and After swapping values will be same here**:
After swapping, a = 30 and b is 45
```

Method overloading trong Java

Khi một lớp có hai hoặc nhiều phương thức cùng tên nhưng khác nhau về tham số, nó được biết đến như là *method overloading*. Nó khác với **Overriding**. Trong overriding, một phương thức có một phương thức khác cùng tên, kiểu, số tham số, …

Bạn xem xét ví dụ sau trước khi tìm các số nhỏ nhất trong kiểu integer. Nếu chúng ta muốn tìm số nhỏ nhất ở kiểu double. Thì khi đó khái niệm về Overloading sẽ được giới thiệu để tạo hai hoặc nhiều phương thức cùng tên nhưng khác nhau về tham số.

Ví dụ sau giải thích điều này:

```
public class ExampleOverloading{

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
```

```
      double c = 7.3;
      double d = 9.4;
      int result1 = minFunction(a, b);
      // same function name with different parameters
      double result2 = minFunction(c, d);
      System.out.println("Minimum Value = " + result1);
      System.out.println("Minimum Value = " + result2);
   }

  // for integer
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }
   // for double
   public static double minFunction(double n1, double n2) {
     double min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }
}
```

Nó sẽ cho kết quả sau:

```
Minimum Value = 6
```

```
Minimum Value = 7.3
```

Các phương thức overloading làm chương trình có thể đọc được. Ở đây, hai phương thức được cung cấp cùng một tên nhưng khác tham số. Số nhỏ nhất từ kiểu integer và kiểu double là kết quả.

Sử dụng các tham số dòng lệnh (command-line) trong Java

Đôi khi bạn muốn truyền thông tin vào trong một chương trình khi bạn chạy nó. Điều này được thực hiện bởi việc truyền các tham số dòng lệnh tới phương thức *main()*.

Một tham số dòng lệnh là thông tin mà trực tiếp theo sau tên của chương trình trên dòng lệnh khi nó được thực thi. Truy cập các tham số dòng lệnh bên trong một chương trình Java là khá dễ dàng. Chúng được lưu giữ như là các chuỗi trong mảng String đã truyền tới main().

Ví dụ:

Chương trình sau hiển thị tất cả tham số dòng lệnh mà được gọi với:

```
public class CommandLine {


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

Thử thực thi chương trình này như đã chỉ ở đây:

```
java CommandLine this is a command line 200 -100
```

Nó sẽ cho kết quả sau:

```
args[0]: this
args[1]: is
args[2]: a
args[3]: command
```

```
args[4]: line
args[5]: 200
args[6]: -100
```

Constructor trong Java

Một constructor khởi tạo một đối tượng khi nó được tạo. Nó có cùng tên với lớp của nó và cú pháp tương tự như một phương thức. Tuy nhiên, các constructor không có kiểu trả về rõ ràng.

Một nét đặc trưng là, bạn sẽ sử dụng một constructor để cung cấp các giá trị khởi tạo tới các biến instance được định nghĩa bởi lớp, hoặc để thực thi bất kỳ thủ tục khởi đầu nào khác được yêu cầu để tạo một đối tượng theo mẫu.

Tất cả các lớp đều có các constructor, dù bạn có hay không định nghĩa nó, bởi vì Java tự động cung cấp một constructor mặc định mà khởi tạo tất cả biến thành viên về zero. Tuy nhiên, một khi bạn định nghĩa constructor cho riêng bạn, thì constructor mặc định sẽ không còn được sử dụng nữa.

Ví dụ:

Đây là một ví dụ đơn giản mà sử dụng một constructor trong Java:

```
// A simple constructor.
class MyClass {
   int x;

   // Following is the constructor
   MyClass() {
      x = 10;
   }
}
```

Bạn sẽ gọi constructor để khởi tạo các đối tượng như sau:

```
public class ConsDemo {

   public static void main(String args[]) {
      MyClass t1 = new MyClass();
```

```
      MyClass t2 = new MyClass();
      System.out.println(t1.x + " " + t2.x);
   }
}
```

Thường thì, bạn sẽ cần một constructor mà chấp nhận một hoặc nhiều tham số. Các tham số được thêm tới một constructor theo cách tương tự như chúng được thêm tới một phương thức, vừa khai báo chúng bên trong dấu ngoặc đơn ở sau tên của constructor.

Ví dụ:

Đây là một ví dụ đơn giản mà sử dụng một constructor trong Java:

```
// A simple constructor.
class MyClass {
   int x;

   // Following is the constructor
   MyClass(int i ) {
      x = i;
   }
}
```

Bạn sẽ gọi constructor để khởi tạo các đối tượng như sau:

```
public class ConsDemo {

   public static void main(String args[]) {
      MyClass t1 = new MyClass( 10 );
      MyClass t2 = new MyClass( 20 );
      System.out.println(t1.x + " " + t2.x);
   }
}
```

Nó sẽ cho kết quả sau:

```
10 20
```

Các tham số biến (var-args) trong Java

JDK 1.5 cho bạn khả năng truyền một số các tham số biến cùng kiểu tới một phương thức. tham số trong phương thức được khai báo như sau:

```
typeName... parameterName
```

Trong khai báo phương thức này, bạn xác định kiểu được theo sau bởi một ellipsis (…). Chỉ một tham số độ dài biến có thể được xác định trong một phương thức, và tham số này phải là tham số cuối cùng. Bất kỳ tham số thông thường nào phải đặt trước nó.

Ví dụ:

```
public class VarargsDemo {

   public static void main(String args[]) {
      // Call method with variable args
           printMax(34, 3, 3, 2, 56.5);
      printMax(new double[]{1, 2, 3});
   }

   public static void printMax( double... numbers) {
   if (numbers.length == 0) {
      System.out.println("No argument passed");
      return;
   }

   double result = numbers[0];

   for (int i = 1; i <  numbers.length; i++)
      if (numbers[i] >  result)
      result = numbers[i];
      System.out.println("The max value is " + result);
   }
}
```

Nó sẽ cho kết quả sau:

```
The max value is 56.5
The max value is 3.0
```

Phương thức finalize() trong Java

Nó là có thể để định nghĩa một phương thức mà sẽ được gọi ngay trước khi hủy một đối tượng bởi Garbage Collector. Phương thức này được gọi là *finalize()*, và nó có thể được sử dụng để chắc chắn rằng một đối tượng hoàn toàn kết thúc.

Ví dụ, bạn có thể sử dụng finalize() để đảm bảo rằng một open file mà sở hữu bởi một đối tượng nào đó đã được đóng.

Để thêm một finalizer tới một lớp, đơn giản bạn định nghĩa phương thức finalize(). Java runtime gọi phương thức đó bất cứ khi nào nó chuẩn bị để tái chế một đối tượng của lớp đó.

Bên trong phương thức finalize(), bạn sẽ xác định những hành động nào phải được thực hiện trước khi một đối tượng bị phá hủy.

Phương thức finalize() có form chung là:

```
protected void finalize( )
{
   // finalization code here
}
```

Ở đây, từ khóa protected là một specifier mà ngăn cản việc truy cập tới finalize() bởi code được định nghĩa bên ngoài lớp của nó.

Nghĩa là, bạn không thể biết khi nào hoặc lúc nào finalize() sẽ được thực thi. Ví dụ, nếu chương trình của bạn kết thúc trước khi Garbage Collection xuất hiện, finalize() sẽ không thực thi.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài ví dụ về Phương thức trong Java: **Ví dụ về Phương thức trong Java**.