Non Access Modifier trong Java

Java cũng cung cấp một số Non-Access Modifier để thực hiện các tính năng khác:

- *Static* Modifier để tạo các phương thức lớp và các biến.
- *Final* Modifier để kết thúc sự thi hành của các lớp, các phương thức và các biến.
- *Abstract* Modifier để tạo các lớp và các phương thức trừu tượng.
- *Synchronized* Modifier và *Volatile* Modifier được sử dụng cho các Thread.

Static Modifier trong Java

Biến static trong Java

Từ khóa *static* được sử dụng để tạo các biến mà sẽ tồn tại một cách độc lập trong bất kỳ instance được tạo cho lớp đó. Chỉ có một bản sao biến static tồn tại cho dù có nhiều instance của lớp đi chăng nữa.

Các biến static cũng được biết đến như là các biến class. Các biến local không thể được khai báo là static.

Phương thức static trong Java

Từ khóa static được sử dụng để tạo các phương thức mà sẽ tồn tại một cách độc lập trong bất kỳ instance được tạo cho lớp đó.

Các phương thức static không sử dụng bất kỳ biến instance nào của bất cứ đối tượng trong lớp mà chúng được định nghĩa. Các phương thức static nhận tất cả dữ liệu từ các tham số và tính toán mọi thứ từ các tham số của chúng, mà không tham chiếu tới các biến.

Các biến class và phương thức class có thể được truy cập bởi sử dụng tên lớp được theo sau bởi một dấu chấm và tên biến hoặc phương thức.

Ví dụ

Static Modifier trong Java được sử dụng để tạo các phương thức class và biến class như trong ví dụ sau:

```
public class InstanceCounter {
```

```
   private static int numInstances = 0;

   protected static int getCount() {
      return numInstances;
   }

   private static void addInstance() {
      numInstances++;
   }

   InstanceCounter() {
      InstanceCounter.addInstance();
   }

   public static void main(String[] arguments) {
      System.out.println("Starting with " +
      InstanceCounter.getCount() + " instances");
      for (int i = 0; i < 500; ++i){
         new InstanceCounter();
           }
      System.out.println("Created " +
      InstanceCounter.getCount() + " instances");
   }
}
```

Nó sẽ cho kết quả sau:

```
Started with 0 instances
Created 500 instances
```

Final Modifier trong Java
Biến final trong Java

Một biến final có thể được khởi tạo một cách rõ rằng chỉ một lần. Một biến tham chiếu được khai báo final có thể không bao giờ được tái gán để tham chiếu tới một đối tượng khác.

Tuy nhiên, dữ liệu bên trong đối tượng có thể bị thay đổi. Vì thế, trạng thái của đối tượng có thể thay đổi, nhưng không phải là tham chiếu.

Với các biến, *final* Modifier thường được sử dụng với *static* để tạo một hằng số cho một biến class.

Ví dụ

```
public class Test{
  final int value = 10;
  // The following are examples of declaring constants:
  public static final int BOXWIDTH = 6;
  static final String TITLE = "Manager";

  public void changeValue(){
     value = 12; //will give an error
  }
}
```

Phương thức final trong Java

Một phương thức final không thể bị ghi đè phương thức (override) bởi bất kỳ lớp phụ nào. Như đã đề cập trước đó, final modifier ngăn cản một phương thức bị sửa đổi trong một lớp phụ.

Mục đích chính của việc tạo phương thức final sẽ là: nội dung của phương thức không nên bị thay đổi bởi bên ngoài.

Ví dụ

Bạn khai báo các phương thức sử dụng final modifier trong khai báo lớp, như trong ví dụ sau:

```
public class Test{
    public final void changeName(){
       // body of method
```

```
    }
}
```

Lớp final trong Java

Mục đích chính của việc sử dụng một lớp được khai báo final là để ngăn cản lớp bị phân lớp thành lớp phụ. Nếu một lớp được đánh dấu là final, thì không có lớp nào có thể kế thừa bất kỳ đặc điểm nào từ lớp final đó.

Ví dụ

```
public final class Test {
   // body of class
}
```

Abstract Modifier trong Java
Lớp abstract trong Java

Một lớp abstract có thể không bao giờ được thuyết minh. Nếu một lớp được khai báo là abstract thì mục đích duy nhất cho lớp này là để được kế thừa.

Một lớp không thể vừa abstract và final (một lớp final không thể được kế thừa). Nếu một lớp chứa các phương thức abstract thì lớp đó nên được khai báo là abstract. Nếu không, nó sẽ ném một compile error.

Một lớp abstract có thể chứa cả các phương thức abstract cũng như các phương thức thông thường.

Ví dụ

```
abstract class Caravan{
   private double price;
   private String model;
   private String year;
   public abstract void goFast(); //an abstract method
   public abstract void changeColor();
}
```

Phương thức abstract trong Java

Một phương thức abstract là một phương thức được khai báo mà không có bất kỳ sự triển khai nào. Thân phương thức này được cung cấp bởi lớp phụ. Các phương thức abstract có thể không bao giờ là final.

Bất kỳ lớp nào mà kế thừa một lớp abstract phải triển khai tất cả phương thức abstract của lớp cha, trừ khi lớp phụ này cũng là một lớp abstract.

Nếu một lớp chứa một hoặc nhiều phương thức abstract, thì lớp đó phải được khai báo là abstract. Một lớp abstract không cần chứa các phương thức abstract.

Phương thức abstract kết thúc với một dấu chấm phảy. Ví dụ: public abstract sample();

Ví dụ

```
public abstract class SuperClass{

    abstract void m(); //abstract method

}


class SubClass extends SuperClass{

     // implements the abstract method

      void m(){

           .........

      }

}
```

Synchronized Modifier trong Java

Từ khóa *synchronized* được sử dụng để chỉ rằng một phương thức có thể được truy cập bởi chỉ một thread tại một thời điểm. Synchronized Modifier có thể được áp dụng với bất kỳ một trong 4 mức độ Access Modifier nào.

Ví dụ

```
public synchronized void showDetails(){

.......

}
```

Transient Modifier trong Java

Một biến instance được đánh dấu là *transient* để chỉ rằng JVM bỏ qua biến cụ thể khi xếp thứ tự đối tượng đang chứa nó.

Modifier này được bao trong lệnh mà tạo biến đó, đứng trước lớp hoặc kiểu dữ liệu của biến đó.

Ví dụ

```
public transient int limit = 55;   // will not persist

public int b; // will persist
```

Volatile Modifier trong Java

Volatile được sử dụng để chỉ cho JVM biết rằng một thread đang truy cập biến đó phải luôn luôn sáp nhập bản sao biến private của riêng nó với bản sao master trong bộ nhớ.

Truy cập một biến volatile đồng bộ tất cả bản sao của các biến trong bộ nhớ chính. Volatile chỉ có thể được áp dụng tới các biến instance, mà là kiểu đối tượng hoặc private. Một tham chiếu đối tượng Volatile có thể là null.

Ví dụ

```
public class MyRunnable implements Runnable
{
    private volatile boolean active;


    public void run()
    {
        active = true;
        while (active) // line 1
        {
            // some code here
        }
    }
    public void stop()
    {
        active = false; // line 2
```

```
    }
}
```

Thường thì, run() được gọi trong một thread (cái mà bạn bắt đầu sử dụng Runnable), và stop() được gọi từ thread khác. Nếu trong line 1 giá trị được lưu trữ được sử dụng, vòng lặp có thể không dừng lại khi bạn thiết lập là false trong line 2. Đó là khi bạn muốn sử dụng*volatile*.