**alias và từ khóa static trong Java**

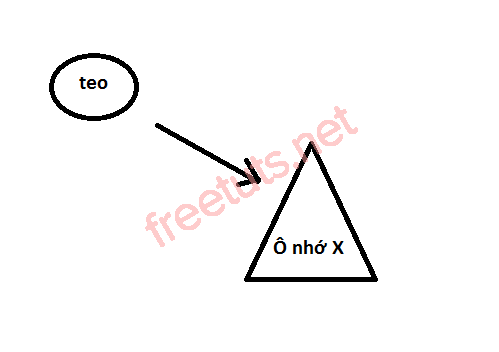
**1. Alias và cơ chế gom rác tự động**

Alias là khả năng mà tại 1 ô nhớ có nhiều đối tượng cùng trỏ tới (>= 2 đối tượng).

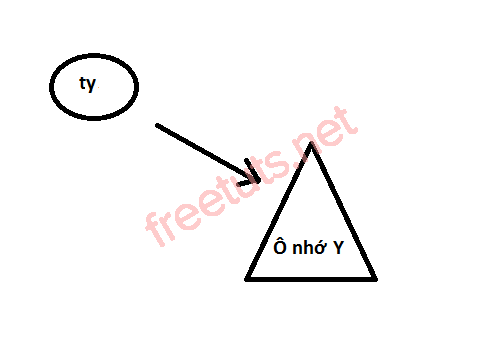
Cơ chế gom rác tự động còn gọi là **garbage collection**. Cơ chế này tự động thu hồi bộ nhớ khi ô nhớ đó không còn đối tượng nào quản lý.

**Ví dụ:** Giả sử chúng ta tạo ra một đối tượng teo của lớp KhachHang: KhachHang teo = new KhachHang();. Trong trường hợp này, từ khóa new sẽ tạo ra một ô nhớ X nào đó và lúc này đối tượng teo sẽ trỏ tới ô nhớ X như hình sau:

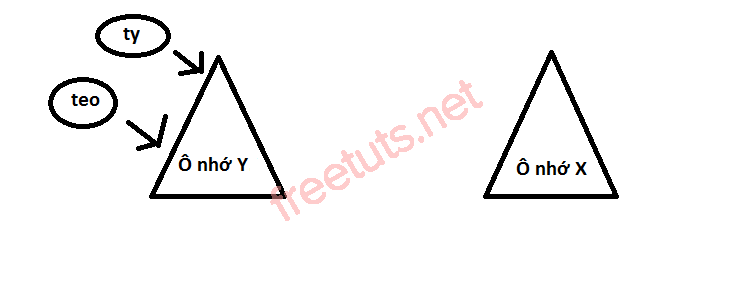
*Bài viết này được đăng tại [free tuts .net]*



Tương tự, chúng ta tạo ra đối tượng ty của lớp KhachHang: KhachHang ty = new KhachHang(); thì lúc này sẽ có một đối tượng ty trỏ tới ô nhớ Y như hình sau:



Hai hình trên cho thấy 2 đối tượng teo và ty trỏ tới (hay còn gọi là quản lý) 2 ô nhớ độc lập, tức là đối tượng teo thao tác trên vùng nhớ X sẽ không ảnh hưởng gì tới đối tượng ty và ngược lai. Bây giờ giả sử ta thực hiện lệnh: teo = ty (gán teo bằng ty) thì hệ thống máy tính sẽ làm việc theo cơ chế như sau: teo trỏ tới ô nhớ mà ty đang quản lý, hay nói cách khác là trong ô nhớ Y bây giờ có 2 đối tượng teo và ty cùng quản lý. Các bạn xem hình sau:



Tại ô nhớ Y lúc này có 2 đối tượng teo và ty cùng trỏ tới, như vậy đã xuất hiện alias ở ô nhớ Y. Lúc này sẽ xảy ra 2 hiện tượng như sau:

* Tại ô nhớ Y, nếu teo thay đổi thông tin sẽ làm cho ty thay đổi thông tin và ngược lại (vì cả 2 đối tượng này cùng quản lý một ô nhớ).
* Ô nhớ X bây giờ không còn đối tượng nào quản lý sẽ dẫn đến trường hợp: hệ thống sẽ tự động thu hồi ô nhớ đã cấp phát (hủy ô nhớ X đã cấp phát trước đó). Cơ chế này gọi là cơ chế gom rác tự động.

Để hiểu rõ hơn, các bạn theo dõi ví dụ sau:

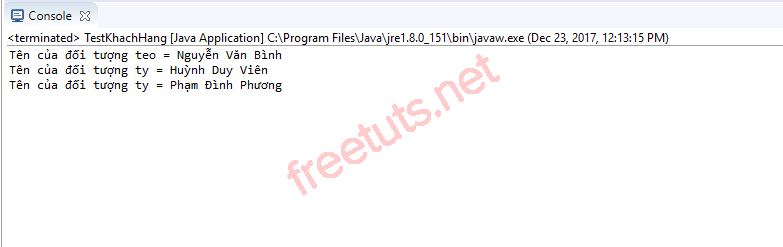
**KhachHang.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | package alias;    public class KhachHang {      private String ma, ten, diaChi;      private int tuoi;        public KhachHang() {          super();      }        public KhachHang(String ma, String ten, String diaChi, int tuoi) {          super();          this.ma = ma;          this.ten = ten;          this.diaChi = diaChi;          this.tuoi = tuoi;      }        public String getMa() {          return ma;      }        public void setMa(String ma) {          this.ma = ma;      }        public String getTen() {          return ten;      }        public void setTen(String ten) {          this.ten = ten;      }        public String getDiaChi() {          return diaChi;      }        public void setDiaChi(String diaChi) {          this.diaChi = diaChi;      }        public int getTuoi() {          return tuoi;      }        public void setTuoi(int tuoi) {          this.tuoi = tuoi;      }        @Override      public String toString() {          return this.ma + ";" + this.ten + ";" + this.tuoi + ";" + this.diaChi;      }  } |

**TestKhachHang.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package alias;    public class TestKhachHang {        public static void main(String[] args) {          KhachHang teo = new KhachHang();          teo.setTen("Nguyễn Văn Bình");          KhachHang ty = new KhachHang();          ty.setTen("Huỳnh Duy Viên");          System.out.println("Tên của đối tượng teo = " + teo.getTen());          System.out.println("Tên của đối tượng ty = " + ty.getTen());            teo = ty;          teo.setTen("Phạm Đình Phương");          System.out.println("Tên của đối tượng ty = " + ty.getTen());    // Phạm Đình Phương      }    } |

**Kết quả sau khi biên dịch chương trình:**



**Giải thích hoạt động của chương trình trên:**

Trong lớp TestKhachHang, dòng code teo = ty; phát sinh ra 2 hiện tượng: đầu tiên là alias (teo và ty cùng trỏ tới 1 ô nhớ), thứ hai là cơ chế gom rác sẽ tự động thu hồi bộ nhớ cấp cho đối tượng teo (vì lúc này teo đã trỏ qua vùng nhớ của ty).

Vì teo đã trỏ qua vùng nhớ mà ty đang trỏ nên lúc này mọi sự thay đổi của teo sẽ ảnh hưởng đến ty và ngược lại, do đó khi teo.setTen("Phạm Đình Phương"); thì ty.setTen() cũng là "Phạm Đình Phương".

**2. Từ khóa static**

Trong Java, từ khóa static chủ yếu được dùng để quản lý bộ nhớ. Chúng ta có thể sử dụng từ khóa static với các biến, các phương thức, các khối lệnh. Từ khóa static thuộc về lớp chứ không thuộc về các đối tượng được tạo ra từ lớp. Các biến được khai báo static được gọi là biến tĩnh, phương thức được khai báo static được gọi là phương thức tĩnh. Sau đây chúng ta sẽ cùng nhau 2 khái niệm chính hay gặp đó là biến tĩnh và phương thức tĩnh.

### **2.1. Đặc điểm chung của các thành phần được khai báo static**

* Được cấp phát bộ nhớ một lần duy nhất ngay khi biên dịch chương trình và ra đời trước đối tượng.
* Có thể dùng chung cho mọi đối tượng.
* Được truy xuất trực tiếp từ tên lớp mà không cần phải khởi tạo đối tượng của lớp đó.
* Được hủy khi kết thúc chương trình.

### **2.2. Biến tĩnh/thuộc tính tĩnh (Static Variable)**

Biến tĩnh (thuộc tính tĩnh) là biến dùng chung cho mọi đối tượng thuộc lớp. Biến tĩnh cũng tương tự như hằng số nhưng biến tĩnh linh hoạt hơn đó là chúng ta có thể thay đổi giá trị khi cần thiết.

Cú pháp khai báo và truy cập biến tĩnh như sau:

**Cú pháp**

|  |  |
| --- | --- |
| 1 | [access modifier] static [kiểu dữ liệu] tenBien = [giá trị khởi tạo]; |

, trong đó:

* [access modifier] là phạm vi truy cập của biến.
* static là từ khóa để khai báo 1 biến là biến tĩnh.
* [kiểu dữ liệu] là kiểu dữ liệu của biến.
* tenBien là tên biến do người dùng đặt và phải tuân thủ theo quy tắc đặt tên biến trong Java.
* [giá trị khởi tạo] là giá trị ban đầu của biến tĩnh này. Nếu chúng ta không khai báo giá trị ban đầu thì biến này sẽ có giá trị mặc định tùy theo kiểu dữ liệu.

Biến tĩnh được truy xuất trực tiếp từ tên lớp. Cú pháp như sau:

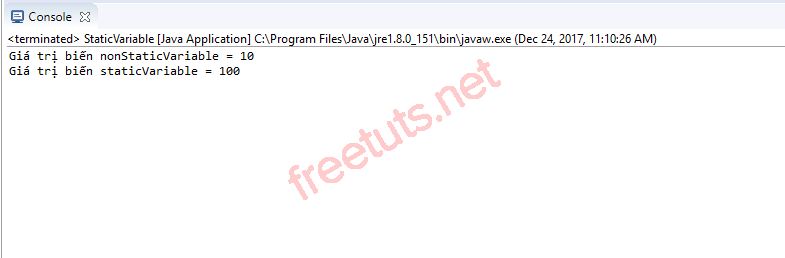
**Cú pháp**

|  |  |
| --- | --- |
| 1 | [Tên\_lớp].[Tên\_thuộc\_tính]; |

**StaticVariable.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | package example\_static;    public class StaticVariable {        int nonStaticVariable = 10;      static int staticVariable = 100;    // khai báo biến tĩnh và khởi tạo giá trị        public int getNonStaticVariable() {          return nonStaticVariable;      }        public void setNonStaticVariable(int nonStaticVariable) {          this.nonStaticVariable = nonStaticVariable;      }        public static void main(String[] args) {          StaticVariable demo = new StaticVariable();          System.out.println("Giá trị biến nonStaticVariable = " + demo.getNonStaticVariable());            // truy cập thuộc tính tĩnh          System.out.println("Giá trị biến staticVariable = " + StaticVariable.staticVariable);      }    } |

Kết quả sau khi biên dịch chương trình:



### **2.3. Phương thức tĩnh (Static Method)**

**Cú pháp**

|  |  |
| --- | --- |
| 1  2  3 | <access modifier] static [kiểu\_trả\_về] tenPhuongThuc {      // ...  } |

, trong đó:

* [access modifier] là phạm vi truy cập của phương thức.
* static là từ khóa để khai báo 1 phương thức là phương thức tĩnh.
* [kiểu trả về] là kiểu trả về của phương thức.
* tenPhuongThuc là tên phương thức do người dùng đặt và phải tuân thủ theo quy tắc đặt tên phương thức trong Java.

Phương thức tĩnh được truy xuất trực tiếp từ tên lớp. Cú pháp:

**Cú pháp**

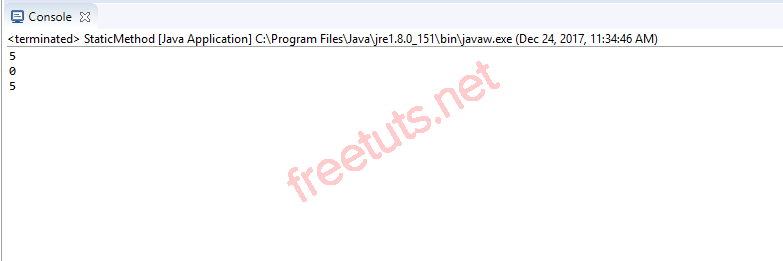
|  |  |
| --- | --- |
| 1 | [Tên\_lớp].[Tên\_phương\_thức]; |

**Lưu ý:** Các phương thức được khai báo static không thể gọi phương thức hay thuộc tính không được khai báo static. Lý do là vì các phương thức tĩnh ra đời trước các thuộc tính, phương thức thông thường nên chúng chưa thấy được các phương thức thông thường khác.

**StaticMethod.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package example\_static;    public class StaticMethod {      public static int x = 5;      public int y;        public static void fn() {          // Dòng code bên dưới sẽ sinh ra lỗi          // vì phương thức static không thể gọi thuộc tính không được khai báo tĩnh          // tại vì cơ chế của static là nó được tạo ra trước khi đối tượng được tạo          // do đó y được tạo sau cùng          // System.out.println(y);            System.out.println(x);      }        // phương thức không khai báo static      // thì có thể gọi thuộc tính static      public void fn2() {          System.out.println(y);          System.out.println(x);      }        public static void main(String[] args) {          StaticMethod.fn();  // gọi phương thức tĩnh          StaticMethod staticMethod = new StaticMethod();          staticMethod.fn2();      }  } |

Kết quả sau khi biên dịch chương trình:



### **2.4. Khi nào chúng ta sử dụng static?**

Chúng ta sẽ sử dụng static khi chúng ta cần chia sẻ các phương thức, thuộc tính cho toàn bộ hệ thống phần mềm của mình, ví dụ như các hàm đọc file, ghi file, kết nối cơ sở dữ liệu,....