Chương 10. Kế thừa

Chương 10 giới thiệu các chủ đề sau:

- Khái niệm kế thừa
- Gọi hàm khởi tạo của lớp cha
- Ghi đè phương thức của lớp cha
- Các thành phần `protected`
- Chuỗi kế thừa
- Lớp `Object`
- Tính đa hình
- Các lớp và các phương thức trừu tượng
- Các giao diện (interface)
- Các lớp trong ẩn danh
- Các giao diện chức năng và biểu thức lambda

1. Khái niệm kế thừa

Kế thừa cho phép một lớp mới mở rộng một lớp có sẵn. Lớp mới sẽ kế thừa các thành phần của lớp mà nó mở rộng đồng thời thêm các thành phần cho riêng nó. Lớp được mở rộng được gọi là lớp cha (*superclass*) hoặc lớp cơ sở (*base class*), lớp mở rộng được gọi là lớp con (*subclass*) hoặc là lớp dẫn xuất (*derived class*).

Để một lớp có thể kế thừa một lớp khác, sinh viên thêm từ khóa `extends` vào phần header của lớp con, ví dụ, với header

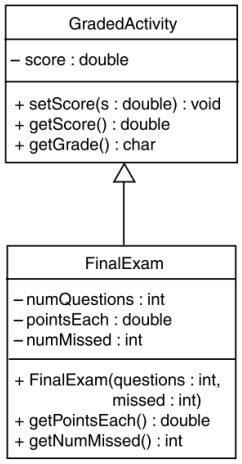
```
public class FinalExam extends FinalExam
```

lớp `FinalExam` sẽ kế thừa lớp `GradedActivity`, lớp `FinalExam` là lớp con (subclass) và lớp `FinalExam` là lớp cha (superclass).

Các thành phần không phải là `private` của lớp `GradedActivity` sẽ được kế thừa trong lớp `FinalExam`. Lớp `FinalExam` có thể sử dụng các thành phần được kế thừa mà không cần phải khai báo lại.

Các thành phần `private` và hàm khởi tạo của lớp cha sẽ không được kế thừa trong lớp con.

Để thể hiện mối liên hệ kế thừa trong UML, sinh viên có thể sử dụng một đường thẳng để kết nối giữa 2 lớp và đường này sẽ có một mũi tên hướng về lớp cha, như hình sau:



Nếu lớp cha có hàm khởi tạo mặc định hoặc hàm khởi tạo mặc định không tham số, hàm khởi tạo của lớp cha sẽ được Java tự động gọi trước khi thực thi hàm khởi tạo của lớp con.

2. Gọi hàm khởi tạo của lớp cha

Để gọi hàm khởi tạo của lớp cha một cách tường minh, sinh viên có thể sử dụng từ khóa `super`. Từ khóa `super` trỏ tới lớp cha của đối tượng và có thể được sử dụng để truy xuất các thành phần được kế thừa từ lớp cha.

Để gọi hàm khởi tạo của lớp cha, sinh viên cần lưu ý một số điều sau:

- Câu lệnh gọi hàm khởi tạo của lớp cha chỉ được viết trong hàm khởi tạo của lớp con. Không thể gọi hàm khởi tạo của lớp cha từ bất cứ phương thức nào khác của lớp con.
- Câu lệnh `super` phải là câu lệnh đầu tiên trong hàm khởi tạo của lớp con. Lý do là hàm khởi tạo của lớp cha phải được thực thi trước khi chương trình trong hàm khởi tạo của lớp con được thực thi.
- Nếu hàm khởi tạo của lớp con không gọi hàm khởi tạo của lớp cha một cách tường minh, hàm khởi tạo mặc định hoặc hàm khởi tạo không tham số của lớp cha sẽ được Java tự động gọi. Điều này tương đương với việc đặt câu lệnh sau vào dòng đầu tiên trong hàm khởi tạo của lớp con.

```
super();
```

3. Ghi đè các phương thức của lớp cha

Một phương thức của lớp con có thể có cùng tên và danh sách tham số với một phương thức của lớp cha. Hiện tượng này được gọi là ghi đè (overriding).

Dòng chứa kí hiệu `@Override` có thể được thêm vào trước dòng bắt đầu khai báo phương thức của lớp con để trình biên dịch Java nhận biết phương thức ghi chồng. Sinh viên nên sử dụng ký hiệu này bởi vì nếu phương thức được khai báo sau dòng `@Override` không ghi đè phương thức của lớp cha, Java sẽ thông báo lỗi. Điều này sẽ giúp sinh viên xây dựng được phương thức ghi đè và sử dụng đúng phiên bản cần thiết của phương thức.

Ghi chồng (overloading) và ghi đè (overriding):

Ghi chồng xảy ra khi một phương thức có cùng tên nhưng khác danh sách tham số với một phương thức khác. Ghi chồng có thể xảy ra trong một lớp hoặc giữa một phương thức của lớp con và một phương thức của lớp cha. Ghi đè chỉ xảy ra trong quan hệ kế thừa.

Khi lớp con ghi chồng phương thức lớp cha, đối tượng được tạo từ lớp con có thể gọi được cả 2 phương thức. khi lớp con ghi đề phương thức của lớp cha, đối tượng của lớp con chỉ có thể gọi được phương thức đã được ghi đè trong lớp con.

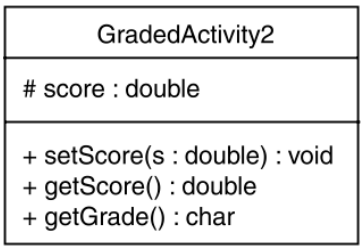
Để một phương thức không bị ghi đè, sinh viên có thể thêm từ khóa final vào phần header khai báo phương thức. Ví dụ, lớp con kế thừa không thể ghi đè phương thức sau:

```
public final void message()
```

4. Các thành phần `protected`

Các thành phần `protected` có thể được truy xuất bởi các phương thức trong lớp và các phương thức của lớp con. Ngoài ra, các thành phần `protected` cũng có thể được truy xuất bởi các phương thức trong các lớp nằm cùng một gói (package) với lớp chứa chúng. Các thành phần `protected` có thể xem là có khả năng truy xuất nằm giữa khả năng truy xuất của các thành phần private và public.

Chỉ báo `protected` được ký hiệu bằng dấu # trong UML. Ví dụ, trong UML sau, thành phần `score` là một thành phần `protected`.



Truy cập gói: nếu sinh viên không đặt chỉ báo truy xuất trước thành phần của lớp, thành phần đó sẽ có kiểu báo truy cập gói. Điều này có nghĩa là các phương thức của các lớp trong cùng một gói có thể truy xuất thành phần này.

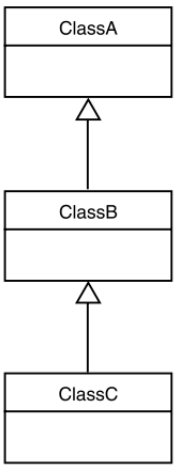
Các bảng sau đây tóm tắt các quyền truy xuất của các thành phần trong lớp:

Chỉ báo truy xuất	Lớp con trong cùng một gói có quyền truy xuất?	Các lớp khác trong cùng một gói có quyền truy xuất?
`default`	Có	Có
`public`	Có	Có
`protected`	Có	Có
`private`	Không	Không

Chỉ báo truy xuất	Lớp con nằm ngoài gói có quyền truy xuất?	Các lớp khác nằm ngoài gói có quyền truy xuất?
`default`	Không	Không
`public`	Có	Có
`protected`	Có	Không
`private`	Không	Không

5. Chuỗi kế thừa

Đôi khi lớp cha trong một quan hệ kế thừa cũng là lớp con trong mối quan hệ kế thừa khác. Điều này tạo ra một chuỗi kế thừa. Chuỗi kế thừa cũng được mô tả trong UML, như hình sau đây:



6. Lớp `Object`

Tất cả các lớp trong Java, bao gồm các lớp có trong Java API và các lớp được tạo từ lập trình viên đều kế thừa từ lớp `Object`. Lớp `Object` là một phần trong gói `java.lang`.

Khi một lớp không có từ khóa `extend` trong phần header, Java sẽ tự động xem lớp này kế thừa trực tiếp từ lớp `Object`. Ví dụ, khi sinh viên khai báo lớp như sau:

```
public class MyClass
{
      (Khai báo các thành phần)
}
```

Java sẽ xem lớp `MyClass` như được viết như sau:

```
public class MyClass extends Object
{
      (Khai báo các thành phần)
}
```

Vì tất cả các lớp trong Java đều kế thừa từ lớp `Object` nên các lớp này sẽ kế thừa các thành phần của lớp `Object`. Hai trong các thành phần hữu dụng nhất của lớp `Object` là các phương thức `toString` và `equals` (là các phương thức mặc định đã được thảo luận ở Chương 8).

7. Tính đa hình

Một biến tham chiếu của lớp cha có thể tham chiếu đến các đối tượng của lớp con. Đây được gọi là tính đa hình. Tính đa hình là khả năng có nhiều dạng. Trong Java, một biến tham chiếu có tính đa hình bởi vì nó có thể tham chiếu đến đối tượng được tạo từ lớp khai báo biến tham chiếu và các đối tượng của lớp khác, miễn là các lớp khác này là các lớp con của lớp khai báo biến tham chiếu.

Mặc dù biến tham chiếu của lớp cha có thể tham chiếu đến các đối tượng của các lớp con, biến tham chiếu chỉ có thể gọi các phương thức được định nghĩa trong lớp cha. Các phương thức được định nghĩa thêm trong lớp con không thể gọi qua biến tham chiếu của lớp cha.

Tính đa hình và liên kết động:

Trong trường hợp lớp con đã ghi đè phương thức của lớp cha, khi gọi phương thức qua biến tham chiếu của lớp cha, máy ảo Java sẽ xác định phương thức gọi dựa trên loại đối tượng mà biến tham chiếu tham chiếu đến. Nếu biến tham chiếu của lớp cha tham chiếu đến đối tượng của lớp con, phương thức định nghĩa trong lớp con sẽ được gọi.

Sinh viên lưu ý: Biến tham chiếu của lớp con không thể tham chiếu đến đối tượng của lớp con.

Toán tử `instanceof`:

Là toán tử giúp xác định một đối tượng có phải là một hiện thân (đối tượng) của một lớp cụ thể hay không. Dạng chung của toán tử `instanceof` như sau:

```
refVar instanceof ClassName
```

Trong đó, `refVar` là biến tham chiếu đến đối tượng, `ClassName` là tên lớp.

Toán tử `instanceof` sẽ trả về một kết quả dạng `boolean`. Kết quả là `true` khi và chỉ khi đối tượng được tham chiếu bởi `refVar` là hiện thân của `ClassName`.

Nếu `refVar` là hiện thân của lớp con của `ClassName`, toán tử `instanceof` cũng trả về giá trị `true`.

8. Các lớp và các phương thức trừu tượng

Một phương thức trừu tượng là một phương thức xuất hiện trong lớp cha và phải được ghi đè trong lớp con. Phương thức trừu tượng không có thân, chỉ có header. Dạng chung của phương thức trừu tượng như sau:

```
Chỉ_báo_truy_xuất abstract kiểu_trả_về Tên_phương_Thức (Danh_sách_tham_số);
```

Từ khóa `abstract` xuất hiện trên header của phương thức, và header phương thức kết thúc bằng dấu chấm phẩy.

Phương thức trừu tượng phải được ghi đè trong lớp con, nếu lớp con không ghi đè phương thức trừu tượng, Java sẽ báo lỗi. Các phương thức trừu tượng được sử dụng để đảm bảo rằng các lớp con sẽ thực thi các phương thức.

Khi một lớp chứa một phương thức trừu tượng, lớp đó không thể tạo ra đối tượng. Các phương thức trừu tượng thường được sử dụng trong các lớp trừu tượng. Lớp trừu tượng được tạo ra để các lớp khác kế thừa và người dùng không thể tạo đối tượng từ các lớp trừu tượng. Lớp trừu tượng được sử dụng để tạo dạng chung hoặc trừu tượng cho các lớp khác.

Để tạo lớp trừu tượng, từ khóa abstract được đặt vào phần header của lớp:

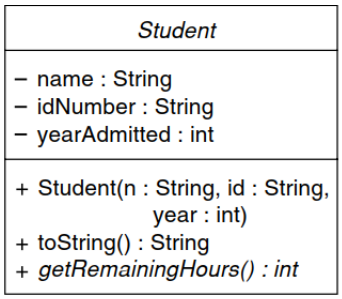
```
Chỉ_báo_truy_xuất abstract class Tên_Lớp
```

Vài điểm cần nhớ với lớp và phương thức trừu tượng:

- Phương thức và lớp trừu tượng được định nghĩa với từ khóa `abstract`.
- Các phương thức trừu tượng không có phần thân và phần header phải kết thúc bằng dấu chấm phẩy.

- Phương thức trừu tượng phải được ghi đè trong lớp con.
- Khi một lớp chứa một phương thức trừu tượng, lớp đó không được dùng để tạo đối tượng, mà chỉ dùng làm lớp cha.
- Một lớp trừu tượng không được dùng để tạo đối tượng, chỉ được dùng để làm lớp cha.

Các lớp trừu tượng được thể hiện như lớp bình thường trong UML, ngoại trừ tên của lớp và tên của các phương thức trừu tượng viết in nghiêng. Ví dụ, trong hình sau lớp `Student` là lớp trừu tượng và phương thức `getRemainingHous` là phương thức trừu tượng:



9. Interface

Trong dạng đơn giản nhất, một interface là một lớp chỉ chứa các phương thức trừu tượng. Một interface không thể tạo ra một đối tượng, thay vào đó, các lớp khác sẽ thực thi interface đó.

Một interface giống như một lớp, ngoại trừ từ khóa `interface` được sử dụng thay cho từ khóa `class`, và phương thức được định nghĩa trong interface không có phần thân, chỉ có phần header được kết thúc bằng dấu chấm phẩy. Dạng chung của một interface như sau:

```
public interface InterfaceName

{

    (Method headers . . .)

}
```

Tất cả các phương thức trong các interface đều phải là phương thức `public`. Sinh viên có thể thêm chỉ báo truy xuất `public` trong header của phương thức, nhưng điều này là không cần thiết.

Lớp thực thi interface phải thực thi phương thức được định nghĩa trong interface với tên và danh sách tham số phải giống với phương thức đó.

Khi muốn một lớp thực thi một interface, sinh viên cần thêm cụm từ chứa từ khóa `implements` và tên interface vào phần header của lớp, ví dụ, để lớp `Person` thực thi interface `Displayable`, sinh viên cần viết header của lớp như sau:

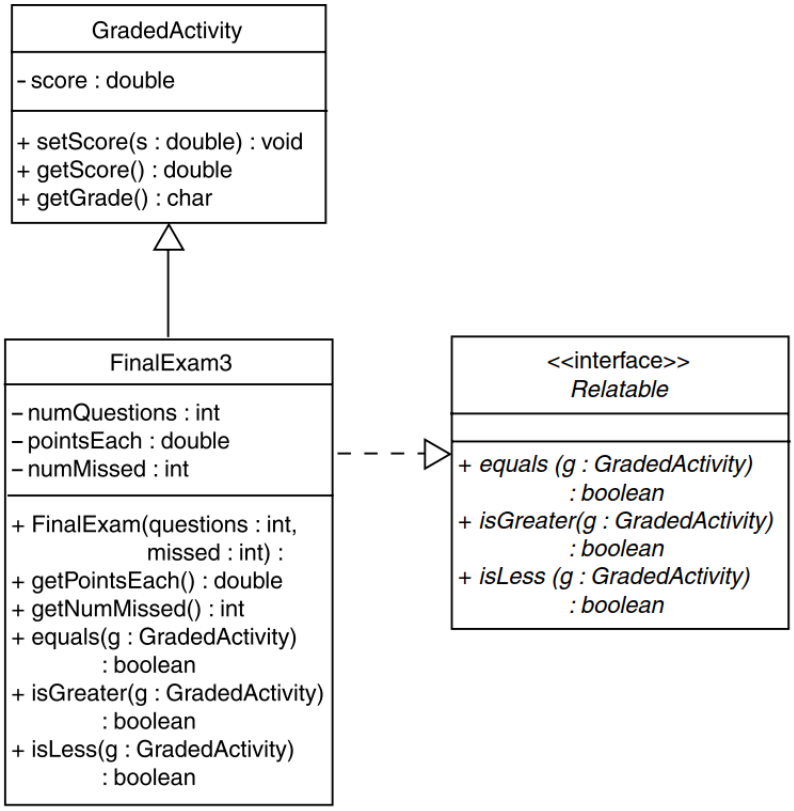
```
public class Person implements Displayable
```

Các thuộc tính của interface: Các thuộc tính của interface được xem như các hằng số `static`. Bởi vì đây là các hằng số `static`, sinh viên cần khởi tạo các thuộc tính. Nếu sinh viên không khởi tạo giá trị cho các thuộc tính này, Java sẽ báo lỗi.

Một lớp chỉ được kế thừa một lớp khác, nhưng Java cho phép một lớp thực thi nhiều interface. Lớp thực thi nhiều interface phải thực thi tất cả các phương thức được chỉ ra bởi các interface đó. Để thực thi nhiều

interface, sinh viên cần thêm tên các interface, phân cách nhau bằng dấu phẩy sau từ khóa `implements` trong phần header của lớp.

Các interface được thể hiện trong UML giống như lớp, ngoại trừ tên interface và tên phương thức viết in nghiêng và thẻ `<<interface>>` được đặt trên tên interface. Lớp thực thi và interface được nối với nhau bằng đường nét đứt của mũi tên hướng về interface, như hình sau:



Bắt đầu từ Java 8, các interface có thể có các phương thức mặc định. Phương thức mặc định là phương thức có phần thân và có từ khóa `default` trước kiểu dữ liệu trả về. Ví dụ:

```
public interface Displayable
{
   default void display()
   {
      System.out.println("This is the default display method.");
   }
}
```

Khi một lớp thực thi một interface, lớp có thể ghi đè phương thức mặc định hoặc là không.

Tính đa hình cũng được áp dụng với biến tham chiếu interface. Một biến tham chiếu interface có thể tham chiếu đến đối tượng của lớp thực thi interface đó. Tuy nhiên, biến tham chiếu này cũng chỉ gọi được phương thức đã định nghĩa trong interface và không thể gọi các phương thức khác được định nghĩa trong lớp thực thi.

10. Lớp trong ẩn danh

Lớp trong ẩn danh là lớp được định nghĩa trong một lớp khác và không có tên. Một lớp trong ẩn danh phải thực thi một interface hoặc kế thừa một lớp khác.

Lớp trong ẩn danh được sử dụng khi muốn thực hiện một lớp đơn giản và sử dụng một lần trong chương trình. Sinh viên có thể sử dụng từ khóa `new` để định nghĩa và tạo một đối tượng của lớp trong ẩn danh.

Cấu trúc chung để định nghĩa và tạo đối tượng từ lớp trong ẩn danh như sau:

```
new Tên_lớp_hoặc_interface() {
      (Các thuộc tính và phương thức của lớp ẩn danh…)
}
```

Cấu trúc trên sẽ trả về tham chiếu của một đối tượng của lớp trong ẩn danh. Lưu ý là các từ khóa `extends` hoặc `implements` không được sử dụng trong khai báo lớp trong ẩn danh.

Sinh viên cần lưu ý một số yêu cầu và hạn chế của lớp trong ẩn danh:

- Một lớp trong ẩn danh phải thực thi hoặc kế thừa một lớp khác.
- Nếu lớp trong ẩn danh kế thừa một lớp khác, hàm khởi tạo không tham số của lớp cha sẽ được gọi trước khi đối tượng của lớp ẩn danh được tạo ra.
- Một lớp con ẩn danh phải ghi đè tất cả các phương thức trừu tượng được chỉ ra trong interface mà nó thực thi hoặc trong lớp mà nó kế thừa.
- Bởi vì lớp trong ẩn danh được định nghĩa trong một phương thức, lớp trong ẩn danh có thể truy xuất các biến cục bộ là hằng số hoặc có giá trị không đổi trong suốt thời gian thực thi của phương thức đó.

11. Interface chức năng và các biểu thức Lambda

Interface chức năng là interface chỉ có một phương thức trừu tượng. Biểu thức lambda được dùng để tạo đối tượng thực thi một interface chức năng.

Từ Java 8, interface chức năng và biểu thức lambda được sử dụng với nhau để đơn giản hóa chương trình, đặc biệt là khi muốn sử dụng lớp trong ẩn danh.

Biểu thức lambda là một phương thức ẩn danh hay còn gọi là một phương thức không tên. Giống phương thức bình thường, biểu thức lambda cũng có thể nhận đối số và trả về giá trị. Dạng chung của biểu thức lambda nhận một đối số và trả về một giá trị:

```
Đối_số -> Giá_trị_trả_về
```

Dấu mũi tên `->` được gọi là toán tử lambda.

Bên trái của toán tử lambda là tên của biến tham số, bên phải của toán tử lambda là giá trị trả về của biểu thức lambda.

Sinh viên có thể sử dụng biểu thức lambda để tạo một đối tượng thực thi một interface, như ví dụ sau:

```
IntCalculator square = x -> x * x;
```

Biến `square` là một biến tham chiếu của interface `IntCalculator`, lưu trữ địa chỉ của đối tượng được tạo từ biểu thức lambda.

Vì biểu thức lambda được sử dụng để thực thi một interface chức năng (chỉ có một phương thức trừu tượng) nên máy ảo Java sẽ tự động xác định tên phương thức đang được thực thi bởi biểu thức lambda.

Sinh viên không cần khai báo biến tham số của biểu thức lambda, Java sẽ tự hiểu.

Nếu phương thức trừu tượng trong interface chức năng không có giá trị trả về, biểu thức lambda cũng không có giá trị trả về, ví dụ:

```
x -> System.out.println(x);
```

Nếu phương thức trừu tượng trong interface chức năng có nhiều tham số, biểu thức lambda cũng có nhiều tham số. Các tham số được đặt trong cặp ngoặc đơn và cách nhau bởi dấu phẩy, ví dụ:

```
(a, b) -> a + b;
```

Nếu phương thức trừu tượng trong interface chức năng không có tham số, biểu thức lambda sẽ có cặp ngoặc đơn bên phải của toán tử lambda, ví dụ:

```
() -> System.out.println();
```

Sinh viên không cần chỉ ra kiểu dữ liệu của các thông số của biểu thức lambda, tuy nhiên, sinh viên có thể khai báo kiểu dữ liệu của các tham số này một các tường minh, ví dụ:

```
(int a, int b) -> a + b;
```

Biểu thức lambda có thể có nhiều câu lệnh trong phần thân. Các câu lệnh này được bỏ vào cặp ngoặc nhọn và phải có câu lệnh `return` nếu biểu thức trả về giá trị, ví dụ:

```
(int x) -> {
     int a = x * 2;
     return a;
};
```

Biểu thức lambda có thể truy xuất các biến là các hằng số hoặc các biến không thay đổi giá trị trong phương thức chưa biểu thức lambda.