LẬP TRÌNH DI ĐỘNG

Bài 3: Java OOP & Collections

Nội dung

1. Java OOP
   - Lớp (class)
   - Đối tượng (object)
   - Gói (package)
   - Các thành phần trong class
   - Sự kế thừa
   - Giao diện (interface)
   - Kiểm soát truy cập
2. Java Collections

Phần 1

Java OOP

Lớp (class)

- Thành phần đặc biệt và quan trọng nhất trong các ngôn ngữ lập trình hướng đối tượng
- Định nghĩa một kiểu dữ liệu mới, thường là tương hợp với bài toán thực tế
  - Trong java, tất cả các class đều kế thừa từ class Object hoặc con cháu của Object
- Java cung cấp thư viện rất lớn các class có sẵn
- Lập trình viên: sử dụng, thay đổi, kết hợp các class đó để phù hợp với mục đích của mình

Lớp (class)

- Ví dụ: cần làm việc với kiểu dữ liệu phân số
- Các làm không hướng đối tượng:
  - Tử số lưu ở biến a
  - Mẫu số lưu ở biến b
  - Viết các hàm toán học làm việc với a và b
- Các làm hướng đối tượng:
  - Tạo một kiểu dữ liệu mới: PhanSo, bên trong có 2 thành phần tử số và mẫu số
  - Viết các hàm đính kèm với kiểu PhanSo để thực hiện các phép toán

Lớp (class)

```
// khai báo
class PhanSo {
    int tuso, mauso;
    public PhanSo() { tuso = 0; mauso = 1; }
    public PhanSo(int tu, int mau) {
        tuso = tu; mauso = mau;
    }
    ...
}

// sử dụng
PhanSo x = new PhanSo();          // phân số x = 0/1
PhanSo y = new PhanSo(1, 3);      // phân số y = 1/3
```

Đối tượng (object)

- Đối tượng (object): biến có kiểu class nào đó
- Ví dụ: class PhanSo => biến x có kiểu PhanSo => x là một đối tượng
- Vòng đời của object:
  1. Khai báo (chưa dùng được)
  2. Khai sinh (cấp phát bộ nhớ + constructor)
  3. Sử dụng
  4. Khai tử (destructor + thu hồi bộ nhớ đã cấp)
- Cơ chế "garbage collection" của java tự động thực hiện việc hủy đối tượng và thu hồi bộ nhớ (bước 4)

Đối tượng (object)

```
// định nghĩa lớp PhanSo
class PhanSo {
    …
}

// sử dụng các object có kiểu PhanSo
PhanSo x, y;              // khai báo
x = new PhanSo(1, 3);     // khai sinh
x.printInfo();            // sử dụng
y = x;                    // sử dụng
y.add(x);                 // sử dụng
x = y = null;             // đánh dấu khai tử
```

Gói (package)

- Gom một số class với nhau thành một nhóm
- Cú pháp:

```
package <tên-gói>;
```

- Quy định:
  - Nếu có thì phải đặt ở đầu file
  - Tên gói quy ước như tên biến trong java
  - Sau khi dịch thành file .class, các file này phải được đặt trong thư mục khớp với tên của gói, nếu không sẽ không nạp được
- Trong java: package ~ folder

Gói (package)

- Đăng kí sử dụng gói/lớp: từ khóa "import"
- Lớp thuộc một package cần được khai báo public mới có thể sử dụng ngoài package đó
- Tại sao cần có package:
  - Giải quyết vấn đề có quá nhiều class trong ứng dụng
  - Nhóm các class thuộc cùng một mục đích, giúp việc nghiên cứu/sử dụng/bảo trì mã tốt hơn
  - Giải quyết việc trùng tên của class
  - Tăng hiệu suất làm việc của máy ảo Java
- Các gói kèm JRE cung cấp nhiều tiện ích lập trình

Các thành phần trong class

- Biến
  - Biến của class
  - Biến của object
- Hàm thành phần (method)
- Hàm tạo (constructor)
  - Constructor mặc định
  - Constructor tự tạo
- Hàm hủy (destructor)
- Class con
- Hằng số

Sự kế thừa

- Kĩ thuật quan trọng và cơ bản nhất của OOP
- Cho phép tạo ra một kiểu dữ liệu mới, thừa hưởng tất cả những đặc điểm của kiểu dữ liệu cũ và thêm một số đặc điểm mới
- Cú pháp

```
class MyApp extends MyDefaultApp {
}
```

- Nếu không có phát biểu extends: java mặc định coi class kế thừa từ class Object (lớp gốc của mọi class trong java)

Sự kế thừa

- Ở lớp con, có thể viết lại một số hàm của lớp cha để "phù hợp với tình hình mới"
- Lớp con đương nhiên cùng kiểu dữ liệu với lớp cha

```
MyDefaultApp x = new MyApp();
```

- Chú ý: khi gọi thực hiện hàm, java tự động gọi hàm tương hợp với biến, không phụ thuộc vào kiểu biến (polymorphism)
- Hai từ khóa quan trọng:
  - super
  - this

Giao diện (interface)

- Mô tả các đặc trưng mà đối tượng sẽ có
- Cú pháp:

```
interface Hinh2D {
  double chuVi();
  double dienTich();
}
```

- Chú ý:
  - Các hàm bên trong không có phần thân và luôn là public
  - Một class có thể kế thừa nhiều interface thông qua từ khóa implements
  - Một interface có thể kế thừa nhiều interface khác thông qua từ khóa extends

Giao diện (interface)

- Tương tự như class, interface định nghĩa một kiểm dữ liệu mới
- Một class kế thừa interface thì phải viết lại (override) tất cả các hàm bên trong nó hoặc phải khai báo là abstract
- Có thể khai báo một biến kiểu interface nhưng không thể khởi tạo biến đó
- Có thể sử dụng kĩ thuật anonymous class để viết lại trực tiếp một interface

Kiểm soát truy cập

- Java có 4 kiểu kiểm soát truy cập:
  - public (công khai): có thể truy cập từ mọi nơi
  - protected (bảo vệ): có thể truy cập từ bản thân class hoặc con cháu
  - default (mặc định, không viết gì cả): cho phép truy cập nếu cùng package
  - private (riêng tư): chỉ có thể truy cập bởi chính class
- Các kiểu kiểm soát truy cập cho phép người viết đảm bảo tính logic của chương trình không bị phá vỡ một cách vô ý

Phần 2

Java Collections

Array

- Kiểu dữ liệu cơ bản, dùng thường xuyên nhất
- Là kiểu reference: chú ý khi làm tham số của hàm
- Đặc biệt chú ý việc khởi tạo trước khi dùng

```
// Tạo mảng 100 String, sau đó phải khởi tạo String
String[] a = new String [100];
// Tạo mảng 2 chiều 20 dòng 30 cột
int[][] b = new int [20][30];
// Tạo mảng 2 chiều 20 dòng, sau đó phải khởi tạo
// từng cột, mỗi cột có thể có kích thước riêng
int[][] c = new int [20][];
for (int i = 0; i < c.length; i++)
    c[i] = new int [10];
```

Array

- Mảng N chiều được xem là mảng 1 chiều trong đó mỗi phần tử là một mảng N-1 chiều
  - Phải khởi tạo các phần tử khi sử dụng
  - Không ràng buộc kích thước các phần tử phải như nhau
- Lớp java.util.Arrays: cung cấp nhiều hàm dạng static tiện ích khi làm việc với mảng
  - Hàm sort: sắp xếp các phần tử của mảng
  - Hàm equals: so sánh hai mảng
  - Hàm binarySearch: tìm kiếm nhị phân trong mảng
  - Hàm fill: điền giá trị vào mảng
  - Chú ý: Arrays chỉ làm việc với mảng object

String

- Lớp cơ bản của java, dùng để lưu trữ các chuỗi kí tự
- String có thể được tạo ra từ:
  - Chuỗi kí tự
  - Mảng byte
  - Mảng int
  - String/StringBuffer/StringBuilder
- Đặc điểm cơ bản: dữ liệu tĩnh + string pool

String

- Các hàm thường dùng:
  - `int length()`
  - `int compareTo(s)`
  - `int indexOf(s)/lastIndexOf(s)`
  - `boolean endsWith(s)/startsWith(s)`
  - `String concat(s)/trim()`
  - `String substring(begin, end)`
  - `String replaceAll(r, s)`
- Nghiên cứu thêm: format, split

StringBuffer

- String với nội dung thay đổi được
- Một số hàm thường dùng:
  - `length()`
  - `append(x)`
  - `delete(start, end)`
  - `insert(index, x)`
  - `substring(start, end)`
  - `replace(start, end, x)`
  - `indexOf(s)/lastIndexOf(s)`
- Tham khảo thêm: StringBuilder – giống StringBuffer nhưng chạy nhanh hơn do không xử lý đồng bộ

Collections

- Thường dịch là “bộ chứa” hoặc “vật chứa”
- Khái niệm: Đối tượng có thể chứa bên trong nó các đối tượng khác (ví dụ như mảng)
- Khá nhiều đối tượng trong cuộc sống và toán học có tính chất tương tự:
  - Danh sách sinh viên
  - Danh bạ điện thoại
  - Kết quả thi đại học
  - Tập hợp các hình trong một trò chơi
  - Danh sách các bộ phim cần xem

ArrayList

- Mảng có thể kích thước thay đổi một cách dễ dàng
- Các hàm thông dụng:
  - add/clear/get/set/size
  - remove/indexOf/lastIndexOf/contains/toArray
- Duyệt bằng for hoặc interator:

```
Iterator itr = myList.iterator();
while(itr.hasNext())
  System.out.println(itr.next());
```

Vector

- Tương tự như ArrayList
- Hỗ trợ việc truy cập vào phần tử con thông qua chỉ số (tức là tương tự như mảng):
  - elementAt(index)
- Vector đòi hỏi đồng bộ
- Trường hợp không thực sự cần thiết nên thay thế bằng ArrayList

Hashtable

- Lưu trữ các cặp <khóa> - <giá trị>
- Các <khóa> không được trùng nhau
- Có nhiều kĩ thuật để giải quyết bài toán trên, Hashtable chỉ là một giải pháp
- Các hàm thông dụng:
  - put(key, value)
  - get(key)
  - containsKey(key)
  - containsValue(value)
  - size()

Hashtable

- Duyệt Hashtable:

```
Enumeration enu = myHash.elements();
while (enu.hasMoreElements())
    System.out.println(enu.nextElement());
```

HashMap<K,V>

- HashMap có nhiều điểm tương tự như Hashtable tuy nhiên không tự động đồng bộ (như vậy chạy nhanh hơn khi không bị truy cập bởi nhiều thread)
- HashMap<key,value> chứa các cặp key-value tương ứng (giống như từ điển), trong đó các key không được trùng nhau
- HashMap có cơ chế đặc biệt giúp tìm các value thông qua giá trị các key với tốc độ nhanh

HashMap<K,V>

- Các hàm cơ bản:
  - Hàm clear(): Xóa nội dung HashMap
  - Hàm containsKey(key): Trả về true nếu chứa key
  - Hàm containsKey(value): Trả về true nếu chứa ít nhất 1 value
  - Hàm get(key): Trả về value ứng với key
  - Hàm put(key, value): Chèn cặp (key, value) vào HashMap
  - Hàm size(): Trả về số cặp trong HashMap
  - Hàm remove(key): Xóa cặp (key,value) khỏi HashMap
  - Hàm keySet(): Trả về tập các key