

Bài tập Java By Example



Lời giới thiệu

Sách bài tập do tập thể giáo viên AiTi-Aptech thiết kế và được sử dụng như một phần không thể tách rời khỏi giáo trình đang học của Aptech Ấn Độ với các học viên đang theo học tại Trung tâm.

Tập sách bài tập này là tài liệu lưu hành nội bộ, chỉ dành cho các học viên theo học tại Trung tâm đào tạo Lập trình viên Quốc tế AiTi-Aptech. Mọi hình thức sao chép lại nội dung của sách là vi phạm bản quyền và không tuân thủ Luật Sở hữu trí tuệ của Nhà nước Việt Nam.

AiTi-Aptech luôn mong mỗi tạo dựng một môi trường học tập tốt cho các bạn học viên theo học tại trung tâm. Mọi ý kiến đóng góp về xây dựng Sách bài tập, cải tiến hệ thống xin gửi mail về contact.aiti@gmail.com hoặc đường dây nóng **(04) 6 64 8848**. Chúng tôi sẽ ghi nhận và cải tiến để có thể cung cấp cho các bạn một môi trường học tập ngày một tốt hơn.

“Sự nghiệp tương lai của các bạn là thành công của chúng tôi”

Đội thiết kế Sách bài tập

Việt Nam luôn thiếu Lập trình viên đẳng cấp Quốc tế

Contents

Lời giới thiệu.....	3
B.Tham khảo.....	5
Session 2: Variables and Operators.....	6
Session 4: Introducing Classes.....	10
A.Tóm tắt lý thuyết.....	10
1. Lưu trữ mảng các đối tượng thông qua tham chiếu.....	10
2. Hệ quy trong Java.....	10
3. Kiểu Enumerate trong Java.....	11
4. Thu hồi bộ nhớ với tiến trình Garbage Collection.....	15
C. BÀI TẬP.....	16
Session 5: Arrays.....	17
A.Tóm tắt lý thuyết.....	17
1. Giới thiệu mảng trong Java.....	17
2. Xử lý String trong Java.....	20
3. Làm việc với lớp StringBuilder.....	22
B. Tham khảo.....	22
C. BÀI TẬP.....	24
Session 6: Packages and Access Specifiers.....	25
A.Tóm tắt lý thuyết.....	25
1. Định nghĩa và cách sử dụng gói.....	25
2. Các từ khóa định mức truy xuất – Access Specifiers	25
3. Field and Method Modifier.....	26
Session 7: Inheritance and Interfaces.....	27
A.Tóm tắt lý thuyết.....	27
1. Kế thừa	27
2. Kiểu trả về hiệp biến (Covariant Return Types).....	27

3. Phng thức và thuô c tính ân	28
4. Overload – Na p ch ông.....	29
5. Interfaces – Giao điề n.....	30
B.Tham Khảo.....	30
C.Bài tập.....	31
Session 8: More On Classes.....	32
A.Tóm tắt lý thuyết.....	32
1. Phạm vi của biến (Scope):.....	32
2. Biến nguyên thủy:.....	32
3. Biến tham chiếu:.....	34
4. Từ khóa Static: Biến và Phương thức.....	34
5. Ưu điểm và nhược điểm của phương thức Instance:.....	35
Session 9: Ngoại lệ- Exceptions.....	37
A.Tóm tắt lý thuyết.....	37
1. Giới thiệu:.....	37
2. Mục đích của việc xử lý ngoại lệ:.....	37
3. Mô hình xử lý ngoại lệ:.....	37
4. Các ngoại lệ được định nghĩa với lệnh ‘throw’ và ‘throws’:.....	39
5. Assertion-Xác nhận:.....	40
B.Tham khảo.....	41
C.Bài tập.....	41

Session 1: Introduction to Java

A.Tóm tắt lý thuyết

1.Lập trình hướng đối tượng

Hướng đối tượng (object orientation) cung cấp một kiểu mới để xây dựng phần mềm. Trong kiểu mới này, các đối tượng (object) và các lớp (class) là những khối xây dựng trong khi các phương thức (method), thông điệp (message), và sự thừa kế (inheritance) cung cấp các cơ chế chủ yếu.

2.Lớp và Đối tượng

a.Đối tượng

Khi chúng ta viết một chương trình hướng đối tượng cũng có nghĩa là chúng ta đang xây dựng một mô hình của một vài bộ phận trong thế giới thực. Tuy nhiên các đối tượng này có thể được biểu diễn hay mô hình trên máy tính.

Một đối tượng thế giới thực là một thực thể cụ thể mà thông thường bạn có thể sờ, nhìn thấy hay cảm nhận được. Tất cả các đối tượng trong thế giới thực đều có trạng thái (state) và hành động (behaviour).

b.Lớp

Trong thế giới thực thông thường có nhiều loại đối tượng cùng loại. Chẳng hạn chiếc xe đạp của bạn chỉ là một trong hàng tỉ chiếc xe đạp trên thế giới. Tương tự, trong một chương trình hướng đối tượng có thể có nhiều đối tượng cùng loại và chia sẻ những đặc điểm chung. Sử dụng thuật ngữ hướng đối tượng, chúng ta có thể nói rằng chiếc xe đạp của bạn là một thể hiện của lớp xe đạp. Các xe đạp có một vài trạng thái chung (bánh răng hiện tại, số vòng quay hiện tại, hai bánh xe) và các hành động (chuyển bánh răng, giảm tốc). Tuy nhiên, trạng thái của mỗi xe đạp là độc lập và có thể khác với các trạng thái của các xe đạp khác. Trước khi tạo ra các xe đạp, các nhà sản xuất thường thiết lập một bảng thiết kế (blueprint) mô tả các đặc điểm và các yếu tố cơ bản của xe đạp. Sau đó hàng loạt xe đạp sẽ được tạo ra từ bản thiết kế này. Không hiệu quả nếu như tạo ra một bản thiết kế mới cho mỗi xe đạp được sản xuất.

Trong phần mềm hướng đối tượng cũng có thể có nhiều đối tượng cùng loại chia sẻ những đặc điểm chung như là: các hình chữ nhật, các mẫu tin nhân viên, các đoạn phim, ... Giống như là các nhà sản xuất xe đạp, bạn có thể tạo ra một bảng thiết kế cho các đối tượng này. Một bảng thiết kế phần mềm cho các đối tượng được gọi là lớp (class).

3.Lập trình hướng đối tượng với Java

Các đặc điểm của Java

- đơn giản
- hướng đối tượng
- phân tán
- thông dịch
- mạnh mẽ
- bảo mật
- kiến trúc trung tính
- khả chuyển
- hiệu quả cao
- đa tuyến
- linh động

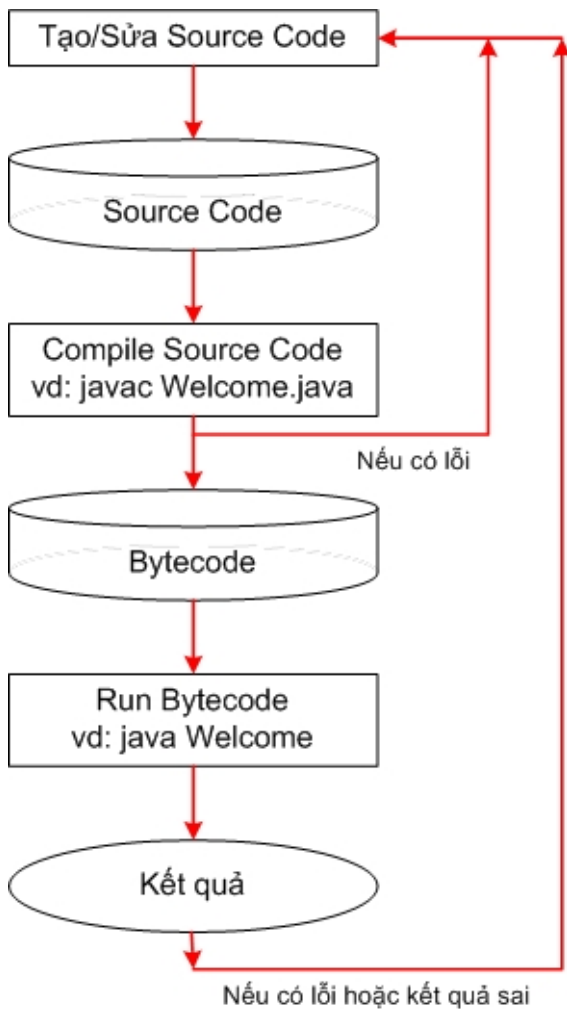
Người ta dùng java để lập trình ra hầu hết các loại ứng dụng :

- Ứng dụng cho điện thoại di động , máy tính siêu nhỏ
- Ứng dụng cho máy tính để bàn
- Ứng dụng web
- Ứng dụng cơ sở dữ liệu
- Ứng dụng phân tán

.....

Thế mạnh của java là có thể chạy trên các hệ nền , các kiểu phần cứng khác nhau .

Mô hình soạn thảo , dịch và chạy một chương trình java



B.Tham khảo

- 1.<http://www.eecs.utoledo.edu/~ledgard/oop/page2b.html>
- 2.http://vi.wikipedia.org/wiki/Lập_trình_hướng_đối_tượng
- 3.<http://java.sun.com/docs/books/tutorial/index.html>

C.Bài tập

- 1.Cài đặt Java 6 trên máy , cấu hình JDK_HOME,PATH, Viết chương trình Hello World
Tham khảo <http://java.sun.com/docs/books/tutorial/getStarted/cupojava/win32.html>
- 2.Làm bài tập trong link sau <http://java.sun.com/docs/books/tutorial/java/concepts/QandE/questions.html>

Session 2: Variables and Operators

A.Tóm tắt lý thuyết

1.Biến và kiểu dữ liệu

a.Tên hay định danh cho biến

- Một tên là một chuỗi các ký tự gồm các chữ, số, dấu gạch dưới (_), và dấu dollar (\$).
- Một tên phải bắt đầu bởi một chữ, dấu gạch dưới (_), hoặc dấu dollar (\$). Nó không thể bắt đầu bởi một số.
- Một tên không thể là một từ khóa.
- Một tên không thể là true, false, hoặc null.
- Một tên có thể có độ dài bất kỳ.

b.Khai báo biến

Công thức: `datatype variableName;`

Ví dụ:

```
int x;      // Khai báo x là một      // biến nguyên (integer);
double bankinh
char a;
```

Java là ngôn ngữ định kiểu mạnh mẽ (strongly-typed) , bao gồm các kiểu nguyên thủy:

- | | |
|---------|-----------|
| • byte | • float |
| • short | • double |
| • int | • char |
| • long | • boolean |

Các lớp được xây dựng trên các kiểu nguyên thủy này (hoặc mảng của các kiểu nguyên thủy)
Ngoài các kiểu nguyên thủy, Java hỗ trợ rất nhiều các kiểu dữ liệu phức tạp , đó là các lớp được xây dựng từ các kiểu nguyên thủy.

Như **String** để biểu diễn chuỗi ký tự , ArrayList để biểu diễn mảng động ...

2.Xuất nhập trong java

Java có thể dùng để lập trình nhiều kiểu ứng dụng . Với mỗi kiểu ứng dụng sẽ có thư viện hỗ trợ xuất nhập riêng .

Trong các ứng dụng dòng lệnh chúng ta cần thao tác với hai dòng xuất nhập chuẩn là **System.in** và **System.out**

a.Sử dụng đối tượng xuất **System.out**

Đối tượng System.out hỗ trợ rất nhiều phương thức có thể xuất dữ liệu ra thiết bị xuất (thường là màn hình dòng lệnh). Sử dụng chủ yếu là phương thức print và printf

ví dụ :

```
System.out.print("Hello");  
System.out.print(123);  
Dùng printf khi muốn sử dụng chuỗi định dạng (tương tự hàm printf trong ANSI C).  
System.out.printf("The number is %d",123);
```

b.Sử dụng đối tượng nhập **System.in**

Để nhập dữ liệu vào chương trình Java qua System.in ta có thể sử dụng ngay các phương thức có sẵn trong System.in như phương thức read.

Tuy nhiên các phương thức này chỉ hỗ trợ nhập dữ liệu dạng thô (mã ASCII của ký tự gõ vào). Nếu muốn xuất nhập cả một chuỗi, hay các kiểu dữ liệu khác char, sẽ khá phức tạp. Tuy nhiên từ phiên bản java 1.5 chúng ta có một lớp mới để làm việc này Scanner (trong gói **java.util**);

Ví dụ để nhập một số int từ bàn phím :

```
Scanner sc = new Scanner(System.in);  
  
int i = sc.nextInt();
```

c.Sử dụng tham số truyền vào hàm *main*

Để truyền tham số vào hàm main ta thực hiện truyền trong khi thực hiện câu lệnh java để chạy chương trình

ví dụ : lớp MyClass

1. public class MyClass
2. {
3. public static void main(String args[])
4. {
5. System.out.println(args[0]);
6. }
7. }
- 8.

```
java MyClass Hello  
sẽ in ra Hello
```

B.Tham khảo

- 1.<http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html>
- 2.<http://java.sun.com/j2se/1.5.0/docs/api/java/util/Scanner.html>

C.Bài tập

- 1.Viết chương trình cộng hai số nguyên được nhập từ bàn phím
- 2.Làm bài tập tại link <http://java.sun.com/docs/books/tutorial/java/data/QandE/numbers-questions.html>

Session 3: Decision-Making and Iterations

A.Tóm tắt lý thuyết

Trong java các cấu trúc điều khiển được thừa kế hoàn toàn từ C , bao gồm

- Cấu trúc rẽ nhánh if-else
- Cấu trúc lặp for, while , do-while

B.Tham khảo

- 1.<http://java.sun.com/docs/books/tutorial/java/nutsandbolts/flow.html>

C.Bài tập

1. Viết chương trình yêu cầu người dùng nhập vào một số tự nhiên . Thực hiện kiểm tra số nhập vào là chẵn hay lẻ .
- 2.Làm bài tập tại link
http://java.sun.com/docs/books/tutorial/java/nutsandbolts/QandE/questions_flow.html
- 3.Viết chương trình in ra dãy fibonacci

Session 4: Introducing Classes

A. Tóm tắt lý thuyết

1. Lưu trữ mảng các đối tượng thông qua tham chiếu

- Trong Java, một mảng các đối tượng chỉ lưu các tham chiếu (references) tới các đối tượng đó.
- Có thể khởi tạo mảng các đối tượng theo cách sau

```
Employee [] staff = new Employee[3];
```

- Sử dụng vòng lặp để khởi tạo đối tượng

```
for(int i=0; i<staff.length(); i++)  
{  
    staff[i] = new Employee();  
}
```

Khởi tạo đối tượng của mảng với tham số cho hàm khởi tạo

```
staff[0] = new Employee("Peter",37,"102 Road");  
staff[1] = new Employee("Peter",37,"102 Road");  
staff[2] = new Employee("Peter",37,"102 Road");
```

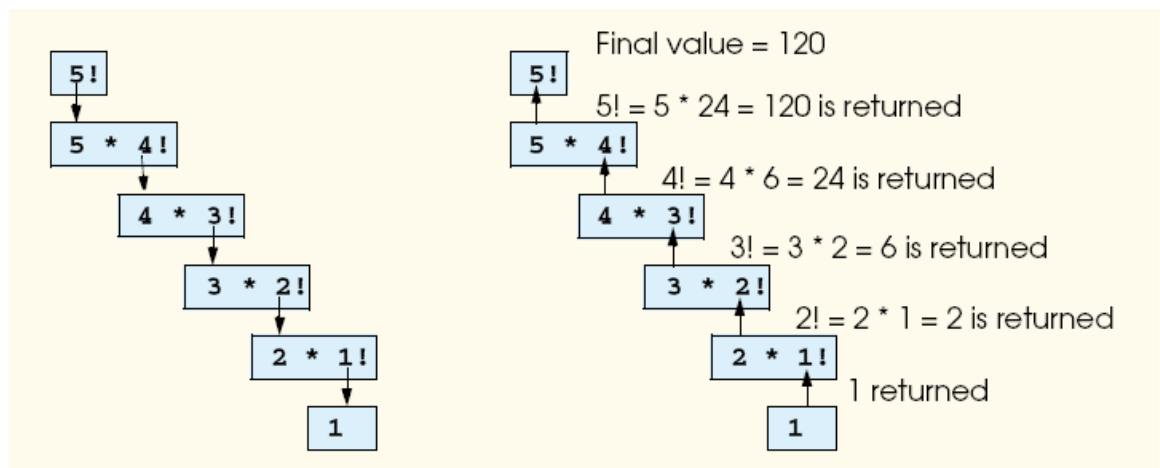
2. Định quy trong Java

- Khái niệm đệ quy : Nếu lời giải của bài toán P được giải quyết bằng bài toán P' có dạng giống như P thì lời giải đó gọi là lời giải đệ quy. Giải thuật tương ứng với lời giải như vậy gọi là giải thuật đệ quy.
- Định nghĩa phương thức đệ quy : phương thức đệ quy là phương thức thực hiện gọi chính nó trong thân phương thức. Một phương thức đệ quy được định nghĩa bởi 2 phần :

- Phần neo (anchor) : Phần này được thực hiện khi công việc quá đơn giản, có thể trả lại kết quả trực tiếp chứ không cần một phương thức con nào cả.
 - Phần đệ quy : Trong trường hợp phương thức chưa thể giải bằng phần neo, cần xác định các phương thức con và gọi đệ quy giải các phương thức con đó. Khi đã có kết quả trả về của những phương thức con thì phối hợp kết quả của chúng lại để trả lại kết quả của phương thức gọi ban đầu.
- Ví dụ phương thức đệ quy (Tính giai thừa)

```
public class Factorial {
    public static long factorial(long x) {
        if (x < 0) throw new IllegalArgumentException("x phải >= 0");
        if (x <= 1) return 1;           //Phần neo
        else return x * factorial(x-1); //Phần đệ quy
    }
}
```

Minh họa



- Các đặc điểm của phương thức đệ quy :
 - Số lần gọi các phương thức là chiều sâu đệ quy.
 - Phương thức đệ quy có thể dẫn tới tràn vùng nhớ Stack
 - Mọi phương thức đệ quy phải có điều kiện kết thúc đệ quy
 - Pha Winding bắt đầu khi phương thức đệ quy được gọi và kết thúc khi điều kiện neo được thực hiện – pha này tương ứng với push vào stack một block bộ nhớ bằng giá trị trả về của phương thức đệ quy
 - Pha Unwinding xảy ra ngay sau pha winding, kết thúc khi kết quả lần gọi đầu tiên của hàm được trả về - pha này tương đương với pop ra từng block trong stack.

3. Kiểu Enumerate trong Java

- Một kiểu enumerate là kiểu dữ liệu tạo bởi tập cố định các hằng (constants).
- Một số ví dụ về kiểu enum


```
public enum Day {  
  
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY  
  
};
```

```
public enum Size {  
  
    SMALL, MEDIUM, LARGE, EXTRA  
  
};
```

- Trong Java 5.0. Kiểu enum được định nghĩa theo cách trên thực sự là một class, class Day có 7 hàm đối tượng, class Size có 4 đối tượng. Có thể định nghĩa một kiểu Enum phức tạp hơn bằng cách thêm hàm khởi tạo, các trường, phương thức như ví dụ sau:

```
public enum Size {  
  
    SMALL("S"), MEDIUM("M"), LARGE("L"), EXTRA ("XL");  
  
    private Size(String abbreviation) {    this.abbreviation = abbreviation;  
    }  
  
    public String getAbbreviation() {  
        return abbreviation;  
    }  
  
    private String abbreviation;  
  
}
```

- Tất cả các lớp thuộc kiểu enum định nghĩa ở trên đều kế thừa lớp java.lang.Enum của Java, do vậy sẽ kế thừa các phương thức của lớp này (tra API)
- Ví dụ về cách làm việc với kiểu enum :

```
public class EnumTest {
```

```
public static void main(String[] args) {  
  
    Scanner in = new Scanner(System.in);  
    System.out.print("Enter a size: (SMALL, MEDIUM, LARGE, EXTRA) ");  
    String input = in.next().toUpperCase();  
  
    Size size = Enum.valueOf(Size.class, input);  
    System.out.println("size=" + size);  
  
    System.out.println("abbreviation=" + size.getAbbreviation());  
  
    if (size == Size.EXTRA)  
        System.out.println("Good job--you paid attention to the _.");  
}  
}
```

- Ưu điểm khi sử dụng kiểu enum :
 - o Khi cần sử dụng một tập cố định các hằng.
 - o Dễ nhớ, dễ đọc code, dễ sử dụng và tốn ít resources bộ nhớ chính.
 - o Kiểu được kiểm tra tại thời điểm biên dịch (type-safe)
 - o Hiệu năng được so sánh gần tương đương khi thực hiện với hằng số int
- Nhược điểm :
 - o Khó sử dụng với người mới làm quen lập trình

4. Thu hồi bộ nhớ với tiến trình Garbage Collection

- Tiến trình Garbage Collection được thực hiện bởi Garbage Collector trong Java có nhiệm vụ "tái sử dụng bộ nhớ - hay memory recycling"

- Garbage Collector thực hiện các nhiệm vụ sau :
 - o Thu thập các đối tượng (obj) không còn tham chiếu và tiến hành lập lịch loại bỏ chúng khỏi vùng Heap do JVM quản lý
 - o Chống phân mảnh vùng nhớ Heap
- Ưu điểm của Garbage Collection :
 - o Giúp lập trình viên không phải xử lý giải phóng bộ nhớ khi lập trình (vốn gây ra lỗi trên các ngôn ngữ không sử dụng Garbage Collector), giảm thiểu Memory leaks.
 - o Bảo đảm tính toàn vẹn của chương trình Java, GC cũng là một phần quan trọng trong khả năng bảo mật và tính an toàn của JVM
- Nhược điểm của Garbage Collection :
 - o Ảnh hưởng tới hiệu năng chương trình do JVM phải theo dõi các đối tượng khi đang được tham chiếu trong quá trình thực thi mã, sau đó phải finalize và giải phóng các đối tượng không còn tham chiếu trực tiếp.

B.Tham khảo

- Java by Example ACCP.i7.
- Core Java 8th Edition by Cay Horstman.
- Beginning Java 5.
- Giai Thuật và Lập Trình by Le Minh Hoang
- Web – java.sun.com/docs/

C. BÀI TẬP

//Bài tập từ CD Java By Example

//Bài tập thêm

Bài 1 : Một dãy Fibonacci có dạng như sau : 0,1,1,2,3,5,8,13,21..., dãy trên bắt đầu bởi 2 số 0 và 1, các số sau bằng tổng của 2 số trước liền kề.

- a. Viết phương thức đệ quy tính số thứ n của dãy Fibonacci : ví dụ fibo(4) = 2;
- b. Tương tự câu a nhưng không sử dụng đệ quy

Bài 2 : Viết chương trình giải bài toán Tháp Hanoi sử dụng phương pháp đệ quy và phương pháp lặp

Bài 3 : Viết chương trình sau sử dụng phương pháp đệ quy : Tính lũy thừa $\text{power}(\text{base}, \text{exponent})$ sẽ trả lại $\text{base}^{\text{exponent}}$, ví dụ $\text{power}(3,4) = 3*3*3*3$. Giá trị base và exponent nhập vào từ bàn phím, kiểm tra exponent phải lớn hơn hoặc bằng 1.

Session 5: Arrays

A.Tóm tắt lý thuyết

1. Giới thiệu mảng trong Java

- Định nghĩa mảng : Một mảng trong Java là đối tượng chứa một tập các biến đã được đặt tên và có cùng kiểu. Mỗi biến trong mảng gọi là một thành phần của mảng (array element). Để tham chiếu đến một thành phần của mảng phải sử dụng chỉ mục (index) của thành phần đó.
- Khai báo và khởi tạo mảng

```
int [] primes;
```

```
int primes[];
```

```
primes = new int[10]; //định nghĩa mảng 10 số
```

```
int [] primes = new int[10];
```

```
primes[9]; //thành phần thứ 10 của mảng
```

```
int [] primes = {2,3,5,7}; //khởi tạo mảng sẵn
```

```
//khởi tạo sử dụng vòng lặp
```

```
double [] data = new double[50];
```

```
for(int i = 0; i < data.length; i++) {
```

```
    data[i] = 1.0;
```

```
}
```

```
//khởi tạo sử dụng java.util.Arrays
```

```
double [] data = new double[50];  
Arrays.fill(data,1.0);
```

- Mảng của mảng hay mảng nhiều chiều

```
float [][] temp = new float[10][365];  
float [][] samples;  
samples = new float[5][];  
long [][][] beans = new long[10][20][30];
```

- Mảng các ký tự

```
char [] message = new char[50];  
java.util.Arrays.fill(message,'_');  
char [] vowels = {'a','e','o','u','i'};  
char [][] names = new char[5][50];
```

- Sắp xếp một mảng :

```
//Bubble Sorting  
int [] array = {2,7,4,13,5,6};  
for(int i=1; i<array.length; i++) {  
    for(int j=0; j<array.length-i;j++) {  
        if(array[j]>array[j+1]) {  
            int temp=array[j]; array[j]=array[j+1];  
            array[j+1] = temp;  
        }  
    }  
}
```

- Tìm kiếm mảng

```
//Tim kiem tuyen tinh – linear search

int [] array = {2,7,4,15,3,1};

int key = 2;

for(int i = 0; i<array.length; i++) {

    if(array[i]==key) {

        System.out.println("Result = " + array[i]);

        break;

    }

}

//Tim kiem nhi phan tren mang da sap xep Bsearch

//return -1 neu khong tim thay

public int binarySearch(int array[], int key) {

    int low = 0; // low element index

    int high = array.length - 1; // high element

    int middle; // middle element index

    // lap cho den khi low > high index

    while ( low <= high ) {

        middle = ( low + high ) / 2;

        if ( key == array[ middle ] )

            return middle;

        else if ( key < array[ middle ] )

            high = middle - 1;

        else

            low = middle + 1;

    }

}
```



```

        return -1;
    }

```

2. Xử lý String trong Java

Tóm tắt phương thức của java.lang.String	
char	charAt (int index) Trả lại giá trị char tại index cho trước
int	codePointAt (int index) Trả lại ký tự (unicode code point) tại index cho trước
int	codePointBefore (int index) Trả lại ký tự (unicode code point) trước index cho trước
int	codePointCount (int beginIndex, int endIndex) Trả lại số các unicode code point trong một khoảng
int	compareTo (String anotherString) So sánh 2 xâu
boolean	endsWith (String suffix) Kiểm tra xem xâu có kết thúc với một hậu tố không
boolean	equalsIgnoreCase (String anotherString) So sánh 2 xâu bỏ qua chữ hoa chữ thường
boolean	startsWith (String prefix) Kiểm tra xem xâu có bắt đầu với một tiền tố không
char[]	toCharArray () Chuyển một xâu tới một mảng ký tự
String	toLowerCase () Chuyển tất cả ký tự thành chữ thường
String	toUpperCase ()

	Chuyển tất cả ký tự thành chữ hoa
--	-----------------------------------

3. Làm việc với lớp `StringBuilder`

Tóm tắt phương thức của <code>java.lang.StringBuilder</code>	
<code>StringBuilder</code>	<code>appendCodePoint</code> (int codePoint) Thêm vào cuối xâu dạng string của code Point
<code>StringBuilder</code>	<code>delete</code> (int start, int end) Xóa ký tự trong substring
<code>StringBuilder</code>	<code>deleteCharAt</code> (int index) Xóa ký tự tại index
void	<code>getChars</code> (int srcBegin, int srcEnd, char[] dst, int dstBegin) Copy các ký tự đến một xâu đích
<code>StringBuilder</code>	<code>replace</code> (int start, int end, <code>String</code> str) Thay thế các ký tự trong xâu bởi một xâu khác
void	<code>setCharAt</code> (int index, char ch) Ký tự tại vị trí index được thay đổi tới ch
void	<code>setLength</code> (int newLength) Thiết đặt chiều dài của xâu
<code>String</code>	<code>substring</code> (int start, int end) Trả lại một xâu con

B. Tham khảo

- Java by Example ACCP.i7.
- Core Java 8th Edition by Cay Horstman.
- Beginning Java 5.
- Giai Thuật và Lập Trình by Le Minh Hoang
- Web – java.sun.com/docs/

C. BÀI TẬP

//Bài tập từ CD Java By Example

//Bài tập thêm :

Bài 1 : Viết một chương trình đọc vào một biến kiểu String đoạn text cho trước, trích rút các từ (words) trong đoạn text đó và sắp xếp chúng theo thứ tự bảng chữ cái. Hiển thị danh sách đã sắp xếp lên màn hình. Giải thuật sắp xếp tùy chọn.

Bài 2 : Định nghĩa một mảng của 10 xâu kiểu String bất kỳ sao cho mỗi xâu có định dạng "day/month/year", ví dụ "21/12/2000". Phân tích các thành phần trong xâu sau đó đưa ra màn hình 10 xâu dưới dạng như sau 21 December 2000.

Bài 3 : Viết một chương trình đảo ngược trật tự các ký tự trong từng từ của một xâu (nhập vào từ bàn phím). Ví dụ "To Be Or Not To Be" sẽ trở thành "oT eB rO toN oT eB"

Bài 4 : Viết một chương trình sử dụng các số ngẫu nhiên để tạo câu. Cài đặt 4 mảng các string là "article", "noun", "verb", "preposition". Tạo các câu bằng cách lựa chọn ngẫu nhiên mỗi từ từ một mảng theo trật tự sau : article-noun-verb-preposition-article. Các từ trong câu cách nhau bằng 1 dấu cách. In ra 10 câu ngẫu nhiên trên màn hình. Các mảng có thể như sau

Article : the, an, a, one, some, any

Noun : boy, girl, man, dog, car, town

Verb : drove, jumped, ridden, walked, kicked, hit

Preposition : to, from, over, on, under

Bài 5 : Viết một chương trình đọc vào một chuỗi, sau đó tokenize chuỗi đó sử dụng StringTokenizer và đưa ra các token theo thứ tự ngược lại.

Bài 6 : Viết một chương trình đọc vào chuỗi số bất kỳ : vd 1132422323, sau đó in ra màn hình dưới dạng : 1,132,422,323

Bài 7 : Viết một chương trình đọc vào chuỗi ký tự bất kỳ, sau đó đếm số lần xảy ra mỗi từ trong chuỗi ký tự đó và in ra màn hình theo định dạng : word – number of words

Session 6: Packages and Access Specifiers

A. Tóm tắt lý thuyết

1. Định nghĩa và cách sử dụng gói

- Một gói trong Java được coi như một thư mục, là nơi tổ chức các lớp và các giao diện. Một gói có thể chứa các gói con.

- Mục đích sử dụng gói :

- + Cho phép tổ chức các lớp thành các đơn vị nhỏ hơn, có thể dễ dàng xác định vị trí
- + Tránh việc đặt tên bị xung đột khi có quá nhiều lớp

- Cách tạo và sử dụng gói

+ Để xác định một lớp hoặc một giao diện là thành phần trong gói, **ở dòng đầu tiên của lớp** phải sử dụng khai báo sau :

package <packagename>;

+ ***Tên gói bao giờ cũng được bắt đầu bằng chữ thường và không có khoảng trắng***

+ ***Để sử dụng gói trong một lớp, sử dụng từ khóa import như sau***

import packagename.class1; // sử dụng một lớp thuộc gói đó

import packagename.* // sử dụng toàn bộ gói

Câu lệnh import phải được đặt sau câu lệnh khai báo gói và trước định nghĩa lớp

2. Các từ khóa định mức truy xuất – Access Specifiers

- Các từ khóa định mức truy xuất dùng để chỉ định quyền truy cập tới lớp và các thành phần của lớp

- Bảng tóm tắt quyền truy cập như sau :

	public	protected	No modifier	private
Class	Yes	Yes	Yes	Yes
Packages	Yes	Yes	Yes	No

Subclass	Yes	Yes	Yes	No
Different package	Yes	No	No	No

3. Field and Method Modifier

- i. Là những từ khóa hỗ trợ cho việc chỉ định những thuộc tính (field) và phương thức (method) dùng cho việc điều khiển các truy xuất đến người dùng
- ii. Có thể được dùng cùng với access specifier như **public** hoặc **protected**.
 - + **volatile** : chỉ sử dụng cho thuộc tính (field) , cho phép một thuộc tính được đồng bộ khi xử lý đa luồng, có nghĩa là nếu giá trị của thuộc tính thay đổi, tất cả các luồng truy xuất đến thuộc tính đó đều sẽ nhận thấy.
 - + **native** : chỉ sử dụng cho phương thức (method), cho biết sự thực hiện đầy đủ của phương thức đó được đặt bên ngoài lớp. Điển hình là các phương thức được thực hiện bên ngoài JVM và được đặt trong các thư viện không viết bằng Java
 - + **transient**: dùng để khai báo những thuộc tính (field) không được lưu trữ như một phần của đối tượng, phục vụ cho việc tiết kiệm bộ nhớ

Session 7: Inheritance and Interfaces

A. Tóm tắt lý thuyết

1. Kế thừa

- Là một khái niệm cho phép sử dụng lại các đoạn code có sẵn. Trong trường hợp người lập trình muốn tạo một lớp, trong khi đã có sẵn một lớp khác với những đoạn code có thể sử dụng lại, khi đó chỉ việc thừa kế lại lớp có sẵn.
- Một lớp thừa kế lại lớp có sẵn được gọi là lớp con (*sub-class*) , lớp được thừa kế gọi là lớp cha (*super-class*).
- Tất cả các lớp trong Java được ngầm định là lớp con của lớp Object.
- Một lớp con chỉ có thể thừa kế từ **duy nhất** một lớp cha
- Dùng từ khóa **extends** để biểu diễn sự thừa kế

```
class Mammal {}
```

```
class Whale extends Mammal {}
```

- Một lớp con sẽ kế thừa tất cả các thuộc tính, phương thức và cả những lớp được lồng trong lớp cha (nested-class). **Không kế thừa Constructor** .
- Lớp con sẽ kế thừa tất cả các thành phần được khai báo **protected** hoặc **public** của lớp cha, không phụ thuộc việc lớp con thuộc gói nào.

2. Kiểu trả về hiệp biến (Covariant Return Types)

- Là một tính năng mới trong J2SE 5.0 , cho phép khi override một method trong lớp cha có thể trả về một kiểu dữ liệu mà kiểu dữ liệu đó là sub-class của kiểu trả về trong lớp cha
- Ví dụ:

```
class Student {
```

```
    public Number getMarks () {
```

```
        return new Number(); // trả về một đối tượng thuộc lớp Number
```

```
    }
```

```
}
```

```

class ExchangeStudent extends Student {
    public Integer getMarks() {
        return new Integer(); // trả về một đối tượng thuộc lớp Integer
    }
}

```

Trong ví dụ trên, vì lớp **Integer** là sub-class của lớp **Number**, nên đã có sự ngầm định của việc ép kiểu từ **Number** về **Integer**

3. Phương thức và thuộc tính ẩn

- Phương thức ẩn

Bản chất của việc override một method được khai báo static là ẩn đi method đó.

Ví dụ :

```

class Foo {
    public static void method() {
        System.out.println("in Foo");
    }
}

class Bar extends Foo {
    public static void method() {
        System.out.println("in Bar");
    }
}

```

Bản chất của đoạn code trên không phải là một ví dụ về override một method được khai báo **static**. Đoạn code trên ví dụ cho việc method của lớp con ẩn đi method của lớp cha.

- Thuộc tính ẩn trong một lớp : Một thuộc tính của lớp con sẽ ẩn đi thuộc tính cùng tên của lớp cha, ngay cả trong trường hợp nó có cùng kiểu dữ liệu. Trong lớp con, chúng ta truy xuất đến thuộc tính của lớp cha bằng từ khóa **super**

4. Overload – Nạp chồng

Trong một class có thể khai báo hai hay nhiều phương thức có cùng tên, cùng kiểu trả về nhưng có số lượng tham số đầu vào khác nhau

- Ví dụ :

```
class Laboratory {  
    public Compound makeCompound(Chemical a, Chemical b) {  
        // Do something  
    }  
  
    public Compound makeCompound(Chemical a, Chemical b, Chemical c ) {  
        // Do something  
    }  
}
```

Như vậy việc thực thi phương thức sẽ phụ thuộc vào các tham số truyền vào trong quá trình gọi

Overloading có thể áp dụng với Constructor

5. Interfaces – Giao diện

- Tham chiếu interface có thể được sử dụng để chứa những thực thể của lớp thực thi interface đó
- Các biến được khai báo trong interface được ngầm định là **final** và **static**
- Interface có thể được thừa kế

B.Tham Khảo

C.Bài tập

(Bài tập cho chương 6,7)

1. Tạo một interface và sử dụng nó để hiển thị bình phương và lũy thừa 3 của một số
2. Tạo một package có chứa một class có hàm trả về giai thừa của đối số truyền vào. Gọi đến hàm đó từ một class trong package khác
3. Tạo một class trong đó có các hàm được overload để tính chu vi hình vuông, tam giác, chữ nhật. Nếu truyền một tham số sẽ trả về chu vi hình vuông, 2 tham số sẽ trả về chu vi hình chữ nhật, 3 tham số sẽ trả về chu vi hình tam giác
4. Làm tương tự bài 3 với các hàm tính diện tích

Session 8: More On Classes

A. Tóm tắt lý thuyết

1. Phạm vi của biến (Scope):

Có hai kiểu biến trong java:

- Biến nguyên thủy (Primitive variable)
- Biến tham chiếu (Reference variable)

2. Biến nguyên thủy:

Được sử dụng để lưu trữ giá trị thuộc kiểu dữ liệu nguyên thủy. Và được chia làm hai loại.

- Instance Variables
- Local Variables

➤ Instance Variable

Biến loại này được khai báo bên trong lớp nhưng nằm ngoài tất cả các method. Instance Variables là những trường của lớp mà chỉ được khởi gán duy nhất khi một lớp được khởi tạo. Các biến này tồn tại khi một instance class chưa bị hủy. Các biến này có thể được truy xuất nếu lớp hoặc class đó tồn tại.

➤ Local Variables:

Biến này được khai báo bên trong một phương thức và chỉ được truy xuất bên trong phương thức đó. Biến này chỉ được khởi tạo khi phương thức đó được gọi. Các biến này sẽ bị hủy khi phương thức đó thực hiện xong.

Ví dụ:

```
public class Vehicle{  
  
    private int horsepower;    // Instance Variable  
  
    public void getEngineType(){  
  
        int numberOfCylinders;    // Local Variable
```

```
        horsepower = 1000;    // Can be accessed
    }

    public void getVehicleType(){

        // Cannot access this variable from this menthod

        numberOfCylinder = 8;
    }
}
```

3. Biến tham chiếu:

Biến tham chiếu được dùng để lưu trữ các đối tượng, cũng gồm hai loại:

- Instance Variables.
- Local Variables.

Ví dụ:

```
public class PixelPoint{
    Pixel pix; // Instance Reference Variable
    public void showPixel(){
        Pixel newPixel; // Local reference variable
    }
    public void setPixel(Pixel pix)
    {
        this.pix = pix;
    }
}
```

4. Từ khóa Static: Biến và Phương thức.

Biến static được gọi là biến lớp, nó tồn tại ở tất cả các instance của lớp. Tất cả các instance của lớp đều có một bản copy của biến static đó.

Phương thức cũng có thể static. Một phương thức có thể ẩn nhưng không thể ghi đè. Phương thức này được truy xuất trực tiếp bởi tên class. Phương thức này chỉ truy xuất được các biến local, các thuộc tính static và các đối số của phương thức đó.

Ưu điểm của phương thức static:

- Phương thức static có thể triệu gọi trực tiếp bởi tên class.
- Phương thức static thực hiện không ảnh hưởng đến trạng thái của các instance.
- Phương thức static có thể định nghĩa lại trong mỗi instance class.
- Sử dụng toán tử '.' để truy xuất các phương thức tĩnh từ tên class. (Class.method());

Nhược điểm của phương thức static:

- Phương thức static được gọi mà không cần phải khởi gán một đối tượng. Compiler chỉ kiểm tra kiểu của đối tượng nơi mà chứa phương thức static được gọi.
- Phương thức static không thể ghi đè, mặc dù nó có thể ẩn.

- Các thuộc tính không phải static sẽ không thể truy xuất được bên trong phương thức static.
- Các phương thức không phải static của lớp hoặc các instance của lớp không thể truy xuất các phương thức static.

5. Ưu điểm và nhược điểm của phương thức Instance:

➤ Ưu điểm:

- Phương thức instance có thể ghi đè hoặc nạp chồng.
- Phương thức ghi đè có thể trả về một kiểu khác.
- Các phương thức này chỉ được truy xuất thông qua một instance của lớp sử dụng toán tử '.'

➤ Nhược điểm:

- Phương thức static không thể gọi các phương thức instance.
- Phương thức có mức truy xuất private thì không thể kế thừa và cũng không thể ghi đè.
- Phương thức instance trong java chỉ trả lại một đối số duy nhất. Đối số đó có thể là kiểu dữ liệu nguyên thủy hoặc một object.

Session 9: Ngoại lệ- Exceptions

A. Tóm tắt lý thuyết

1. Giới thiệu:

Exception là một loại lỗi đặc biệt. Lỗi này xuất hiện vào lúc thực thi chương trình. Các trạng thái không bình thường xảy ra trong khi thi hành chương trình tạo ra các exception. Những trạng thái này không được biết trước trong khi ta đang xây dựng chương trình. Nếu bạn không xử lý các trạng thái này thì chương trình có thể bị kết thúc đột ngột.

2. Mục đích của việc xử lý ngoại lệ:

Một chương trình nên có cơ chế xử lý ngoại lệ thích hợp. Nếu không, chương trình sẽ bị ngắt khi một ngoại lệ xảy ra. Trong trường hợp đó, tất cả các nguồn tài nguyên mà hệ thống đã cấp không được giải phóng. Điều này gây lãng phí tài nguyên. Để tránh trường hợp này, tất cả các nguồn tài nguyên mà hệ thống cấp nên được thu hồi lại. Tiến trình này đòi hỏi cơ chế xử lý ngoại lệ thích hợp.

3. Mô hình xử lý ngoại lệ:

Trong Java, mô hình xử lý ngoại lệ giám sát việc thực thi mã để phát hiện ngoại lệ. Mô hình xử lý ngoại lệ của Java được gọi là 'catch and throw'. Trong mô hình này, khi một ngoại lệ xảy ra, ngoại lệ sẽ bị chặn và chương trình chuyển đến một khối xử lý ngoại lệ. Người lập trình phải xử lý các ngoại lệ khác nhau có thể phát sinh trong chương trình. Các ngoại lệ phải được xử lý, hoặc thoát khỏi chương trình khi nó xảy ra.

Ngôn ngữ Java cung cấp 5 từ khoá sau để xử lý các ngoại lệ:

- try
- catch
- throw
- throws
- finally

Dưới đây là cấu trúc của mô hình xử lý ngoại lệ:

```
try
{
    // đoạn mã có khả năng gây ra ngoại lệ
}
catch(Exception e1)
{

```

```

        // Nếu các lệnh trong khối 'try' tạo ra ngoại lệ có loại e1, thì thực hiện //xử lý
        ngoại lệ nếu không chuyển xuống khối 'catch' tiếp theo
    }
    catch(Exception e2)
    {
        // Nếu các lệnh trong khối 'try' tạo ra ngoại lệ có loại e2, thì thực hiện //xử lý
        ngoại lệ nếu không chuyển xuống khối 'catch' tiếp theo
    }
    catch(Exception eN)
    {
        // Nếu các lệnh trong khối 'try' tạo ra ngoại lệ có loại eN, thì thực hiện //xử lý
        ngoại lệ nếu không chuyển xuống khối 'catch' tiếp theo
    }
    finally
    {
        // khối lệnh nay luôn được thực hiện cho dù ngoại lệ có xảy ra hay không.
    }
}

```

- **Khối try và catch:** Khối **try** chứa một tập lệnh có thể thi hành được và Các ngoại lệ có thể bị chặn khi thi hành tập lệnh này. Lúc đó khối lệnh trong **catch** được gọi để xử lý các ngoại lệ đó. Mỗi một ngoại lệ xảy ra có thể được xử lý bởi một **catch** xác định.
- **Khối finally:** Khối này luôn được thực hiện cho dù không hay có ngoại lệ xảy ra. Khối **finally** thường thực hiện các công việc sau:
 - Đóng tập tin.
 - Đóng ResultSet (được sử dụng trong chương trình cơ sở dữ liệu).
 - Đóng lại các kết nối được tạo trong cơ sở dữ liệu.

4. Các ngoại lệ được định nghĩa với lệnh 'throw' và 'throws':

Các ngoại lệ có thể được tạo ra bằng cách sử dụng từ khoá 'throw'. Từ khoá 'throw' chỉ ra một ngoại lệ vừa xảy ra. Toán hạng của throw là một đối tượng thuộc lớp được thừa kế từ 'Throwable'.

Đoạn lệnh sau chỉ ra cách sử dụng của lệnh 'throw':

```

try
{
    if (flag<0)
    {
        throw new MyException(); // user-defined
    }
}

```

Đoạn mã sau minh họa cách sử dụng của từ khoá 'throws' để tạo nhiều ngoại lệ:

```

public class Example
{

```



```
// Các ngoại lệ cách nhau bởi dấu phẩy

public void exceptionExample() throws ExException, LookupException
{
    try
    {
        // các lệnh
    }
    catch(ExException exmp)
    {
    }
    catch(LookupException lkpex)
    {
    }
}
}
```

Trong ví dụ trên, phương thức 'exceptionExample' có từ khoá 'throws'. Từ khoá này được theo sau bởi danh sách các ngoại lệ mà phương thức này có thể tạo ra – Trong trường hợp này là 'ExException' và 'LookupException'. Hàm xử lý ngoại lệ cho các phương thức này nên khai báo các khối 'catch' để có thể xử lý tất cả các ngoại lệ mà các phương có thể gây ra.

5. Assertion-Xác nhận:

Một xác nhận là một lệnh trong Java cho phép người phát triển kiểm tra một giả định trong ứng dụng. Lệnh assert kiểm tra biểu thức boolean trong quá trình thực hiện. Biểu thức được tin tưởng là true nhưng nó sẽ đưa ra AssertionError nếu biểu thức là false.

Có hai cách viết lệnh assert:

```
Assert booleanExpression;
```

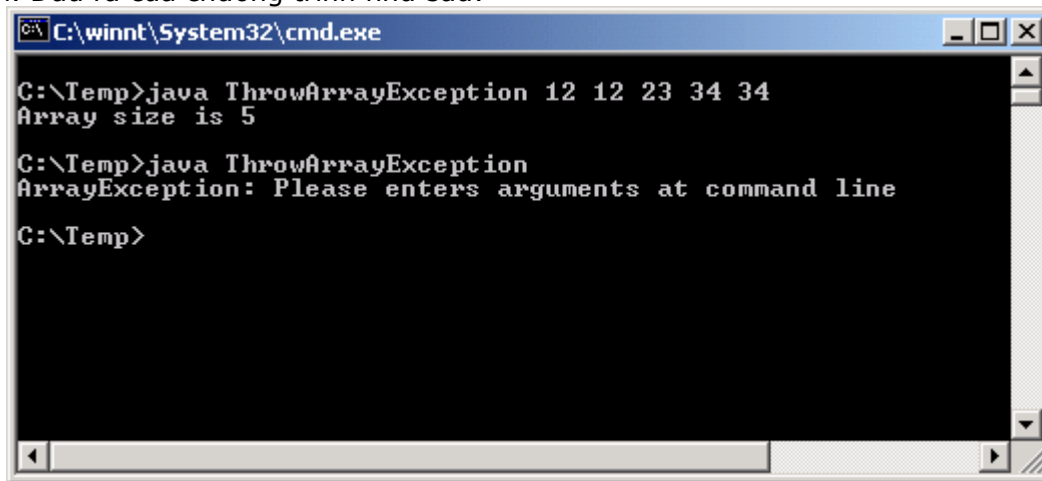
```
Assert booleanExpression;
```

Cách thứ nhất thực hiện biểu thức boolean và trả lại kết quả là true hay false. Lệnh thứ hai cũng giống như lệnh thứ nhất. Thêm vào đó nếu biểu thức boolean là false thì biểu thức hai được trả lại như là một thông tin chi tiết về AssertionError.

B.Tham khảo

C.Bài tập

BÀI 1.—Viết chương trình gây ra ngoại lệ khi người sử dụng không nhập tham số nào vào từ dòng lệnh. Chương trình phải hiển thị số tham số nếu có tham số được nhập vào từ dòng lệnh. Đầu ra của chương trình như sau:



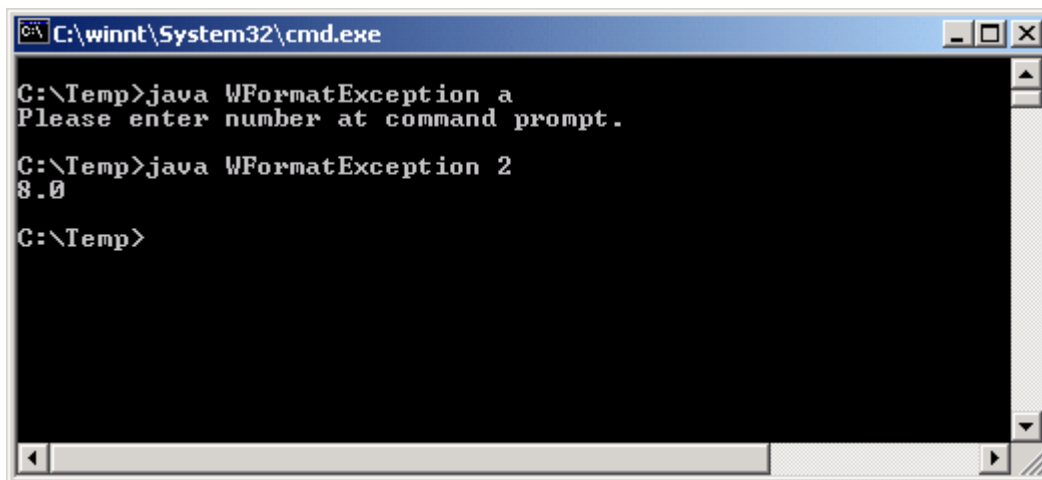
```
C:\winnt\System32\cmd.exe

C:\Temp>java ThrowArrayException 12 12 23 34 34
Array size is 5

C:\Temp>java ThrowArrayException
ArrayException: Please enters arguments at command line

C:\Temp>
```

BÀI 2.—Viết chương trình gây ra ngoại lệ, nếu không có số nào được nhập vào từ dòng lệnh. Ngược lại, chương trình hiển thị giá trị lập phương của số nhập vào như hình dưới đây:



```
C:\winnt\System32\cmd.exe

C:\Temp>java WFormatException a
Please enter number at command prompt.

C:\Temp>java WFormatException 2
8.0

C:\Temp>
```

BÀI 3.—Viết chương trình gây ra ngoại lệ nếu như lớp không thể truy nhập.