Chương 2

Các kiến thức cơ bản về lập trình Java

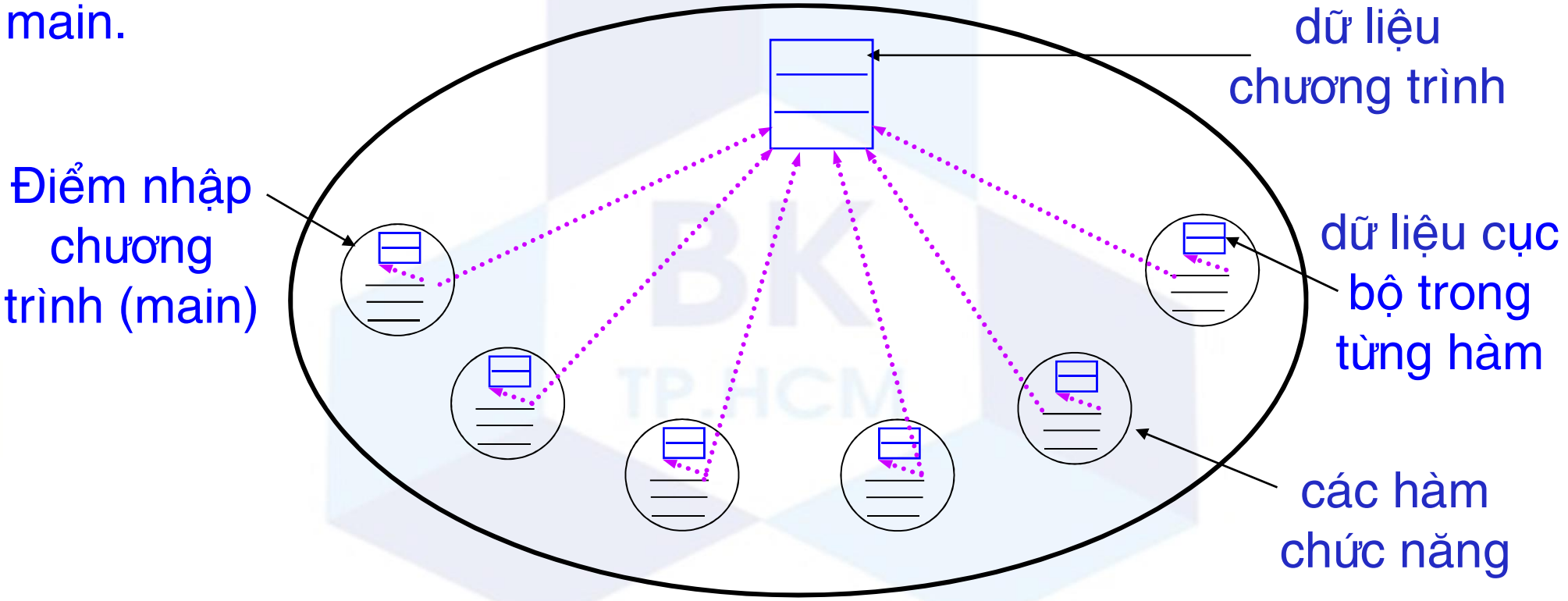
2.0 Dẫn nhập
2.1 Cấu trúc của 1 ứng dụng Java nhỏ
2.2 Kiểu dữ liệu định sẵn
2.3 Kiểu liệt kê
2.4 Kiểu array
2.5 Phân tích top-down theo hướng đối tượng
2.6 Package
2.7 Kết chương

2.0 Dẫn nhập

- Chương này giới thiệu cấu trúc của chương trình Java nhỏ và đơn giản gồm 1 số biến dữ liệu và 1 số hàm xử lý các biến dữ liệu, từ đó tổng kết các kiểu dữ liệu khác nhau có thể được dùng trong 1 chương trình.
- Chương này cũng giới thiệu phương pháp đặt tên cho các phần tử cấu thành ứng dụng lớn 1 cách khoa học, cách chứa các phần tử cấu thành ứng dụng lớn trong các module vật lý.

2.1 Cấu trúc của 1 ứng dụng Java nhỏ

Trong môn kỹ thuật lập trình hay môn lập trình hướng đối tượng, chúng ta đã viết được 1 số ứng dụng nhỏ và đơn giản. Tương tự như vậy, 1 ứng dụng Java đơn giản là 1 class gồm nhiều thuộc tính dữ liệu và nhiều hàm chức năng. Chương trình bắt đầu chạy từ hàm main.



2.1 Cấu trúc của 1 ứng dụng Java nhỏ

```
//định nghĩa package chứa phần mềm
package gptb2;
//import các package cần dùng
import java.io.*;
import java.util.*;
    public class GPTB2 {
        //định nghĩa các biến cần dùng
        static double a, b, c;
        static double delta;
        static double x1, x2;
        static Scanner input;
        static BufferedReader reader;
        static String buf;
```

2.1 Cấu trúc của 1 ứng dụng Java nhỏ

```
//định nghĩa hàm nhập 3 thông số a,b,c của phương trình bậc 2
 static void NhapABC() {
    try {
       //tạo đối tượng nhập dữ liệu thuộc kiểu cơ bản
       input = new Scanner(System.in);
       //chờ người dùng nhập a
      System.out.print("Nhập a : ");
      a = input.nextDouble();
       //chờ người dùng nhập b
      System.out.print("Nhập b : "); b = input.nextDouble();
       //chờ người dùng nhập c
      System.out.print("Nhập c : ");  c = input.nextDouble();
   } catch(Exception e){}
 }
```

2.1 Cấu trúc của 1 ứng dụng Java nhỏ

```
//định nghĩa hàm tính nghiệm của phương trình bậc 2
static void GiaiPT() {
    //tính biệt số delta của phương trình
    delta = b * b - 4 * a * c;
    if (delta >= 0) //nếu có nghiệm thực
    {
        x1 = (-b + Math.sqrt(delta)) / 2 / a;
        x2 = (-b - Math.sqrt(delta)) / 2 / a;
    }
}
```

2.1 Cấu trúc của 1 ứng dụng Java nhỏ

```
//định nghĩa hàm xuất kết quả
 static void XuatKetqua() {
    if (delta < 0)
       //báo vô nghiệm
      System.out.println("Phương trình vô nghiệm");
    else //báo có 2 nghiệm
    {
       System.out.println("Phương trình có 2 nghiệm thực : ");
       System.out.println("X1 = " + x1);
       System.out.println("X2 = " + x2);
    }
 }
```

2.1 Cấu trúc của 1 ứng dụng Java nhỏ

```
    //định nghĩa chương trình (hàm Main)
    public static void main(String[] args)
    {
        NhapABC();          //1. nhập a,b,c
        GiaiPT();           //2. giải phương trình
        XuatKetqua();       //3. xuất kết quả
    }
} //kết thúc class
```

2.1 Cấu trúc của 1 ứng dụng Java nhỏ

Quan sát cấu trúc của chương trình Java nhỏ ở các slide trước, chúng ta có 1 số nhận xét sau :

1. Dữ liệu chương trình thường rất phong phú, đa dạng về chủng loại → Cơ chế định nghĩa kiểu dữ liệu nào được dùng để đảm bảo người lập trình có thể định nghĩa kiểu riêng mà ứng dụng của họ cần dùng ?
2. Nếu ứng dụng lớn chứa rất nhiều hàm chức năng và phải xử lý rất nhiều dữ liệu thì rất khó quản lý chúng trong 1 class đơn giản → cần 1 cấu trúc phù hợp để quản lý ứng dụng lớn.
3. Chương trình thường phải nhờ các hàm chức năng ở các class khác để hỗ trợ mình. Thí dụ ta đã gọi hàm print, println của class System.out để xuất dữ liệu ra màn hình → Cơ chế nhờ vả nào được dùng để đảm bảo các thành phần trong ứng dụng không "quậy phá" nhau?

2.2 Kiểu dữ liệu cơ bản định sẵn

- Các thuật giải chức năng của chương trình sẽ xử lý dữ liệu. Dữ liệu của chương trình thường rất phong phú, đa dạng về chủng loại. Trước hết ngôn ngữ Java (hay bất kỳ ngôn ngữ lập trình nào) phải định nghĩa 1 số kiểu được dùng phổ biến nhất trong các ứng dụng, ta gọi các kiểu này là "kiểu định sẵn".
- Mỗi dữ liệu thường được để trong 1 biến. Phát biểu định nghĩa biến sẽ đặc tả các thông tin về biến đó :
  1. tên nhận dạng để truy xuất.
  2. kiểu dữ liệu để xác định các giá trị nào được lưu trong biến.
  3. giá trị ban đầu mà biến chứa...

2.2 Kiểu dữ liệu cơ bản định sẵn

- Biến thuộc kiểu định sẵn sẽ chứa trực tiếp giá trị, thí dụ biến nguyên chứa trực tiếp các số nguyên, biến thực chứa trực tiếp các số thực Ta gọi kiểu định sẵn là kiểu giá trị (value type) để phân biệt với kiểu tham khảo (reference type) trong lập trình hướng đối tượng ở các chương sau.
- Kiểu tham khảo (hay kiểu đối tượng) sẽ được trình bày trong chương 3 trở đi. Đây là kiểu quyết định trong lập trình hướng đối tượng. Một biến đối tượng là biến có kiểu là tên interface hay tên class. Biến đối tượng không chứa trực tiếp đối tượng, nó chỉ chứa thông tin để truy xuất được đối tượng Ta gọi kiểu đối tượng là kiểu tham khảo (reference type).

2.2 Kiểu dữ liệu cơ bản định sẵn

- boolean : kiểu luận lý, có 2 giá trị true và false.
- byte : kiểu nguyên 1 byte, có tầm trị từ -128 đến 127.
- char : kiểu ký tự Unicode 2 byte, có tầm trị từ mã 0000 đến FFFF.
- short : kiểu nguyên có dấu 2 byte, tầm trị từ -32768 đến 32767.
- int : kiểu nguyên có dấu 4 byte, tầm trị từ -2,147,483,648 đến 2,147,483,647.
- long : kiểu nguyên có dấu 8 byte, tầm trị từ -2^{63} đến $2^{63}-1$.
- float : kiểu thực chính xác đơn, dùng 4 byte để miêu tả 1 giá trị thực, có tầm trị từ $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$. Độ chính xác khoảng 7 ký số thập phân.
- double : kiểu thực chính xác kép, dùng 8 byte để miêu tả 1 giá trị thực, có tầm trị từ $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$. Độ chính xác khoảng 15 ký số thập phân.

2.3 Kiểu do người lập trình tự định nghĩa - Liệt kê

- Ngoài các kiểu cơ bản định sẵn, Java còn hỗ trợ người lập trình tự định nghĩa các kiểu dữ liệu đặc thù trong từng ứng dụng.
- Kiểu liệt kê bao gồm 1 tập hữu hạn và nhỏ các giá trị đặc thù cụ thể. Máy sẽ mã hóa các giá trị kiểu liệt kê thành kiểu byte, short...

```
//định nghĩa kiểu chứa các giá trị ngày trong tuần
enum DayInWeek {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
//định nghĩa biến chứa các giá trị ngày trong tuần
DayInWeek day = DayInWeek.Tue;
```

2.4 Kiểu do người lập trình tự định nghĩa - Array

- Trong trường hợp ta có nhiều dữ liệu cần xử lý thuộc cùng 1 kiểu (thường xảy ra), nếu ta định nghĩa từng biến đơn để miêu tả từng dữ liệu thì rất nặng nề, thuật giải xử lý chúng cũng gặp nhiều khó khăn. Trong trường hợp này, tốt nhất là dùng kiểu Array để quản lý nhiều dữ liệu cần xử lý. Array có thể là :
  - array 1 chiều.
  - array "jagged".

Array 1 chiều

```
int[] intList; //1.định nghĩa biến array là danh sách các số nguyên
//2. khi biết được số lượng, thiết lập số phần tử cho biến array
intList = new int[5];
//3. gán giá trị cho từng phần tử khi biết được giá trị của nó
intList[0] = 1; intList[1] = 3; intList[2] = 5;
intList[3] = 7; intList[4] = 9;
```

Nếu có đủ thông tin tại thời điểm lập trình, ta có thể viết lệnh định nghĩa biến array như sau :

```
int[] intList = new int[] {1, 3, 5, 7, 9};
```

hay đơn giản :

```
int[] intList = {1, 3, 5, 7, 9};
```

Array "jagged"

Array "jagged" là array mà từng phần tử là array khác, các array được chứa trong array "jagged" có thể là array 1 chiều, n chiều hay là array "jagged' khác.

```
int[][] matran; //1. định nghĩa biến array "jagged"
//2. khi biết được số lượng, thiết lập số phần tử cho biến array
matran = new int[3][];  //hay new int[3][2] nếu ma trận đồng nhất
for (int i = 0; i < 3; i++) matran[i] = new int[2];
//3. gán giá trị cho từng phần tử khi biết được giá trị của nó
matran[0][0] = 1; matran[0][1] = 2; matran[1][0] = 3;
matran[1][1] = 4; matran[2][0] = 5; matran[2][1] = 6;
```

Array "jagged"

Nếu có đủ thông tin tại thời điểm lập trình, ta có thể viết lệnh định nghĩa biến array như sau :

```
int[][] array = new int [3][2];
int[][] array = new int [3][];
array[0] = new int[] {1, 2, 3};
array[1] = new int[] {3, 4};
array[2] = new int[] {5, 6, 7, 8};
```

hay đơn giản :

```
int[][] array = new int [][] {new int[]{1, 2,3}, new int[]{3, 4}, new
    int[] {5,6,7,8}};
```

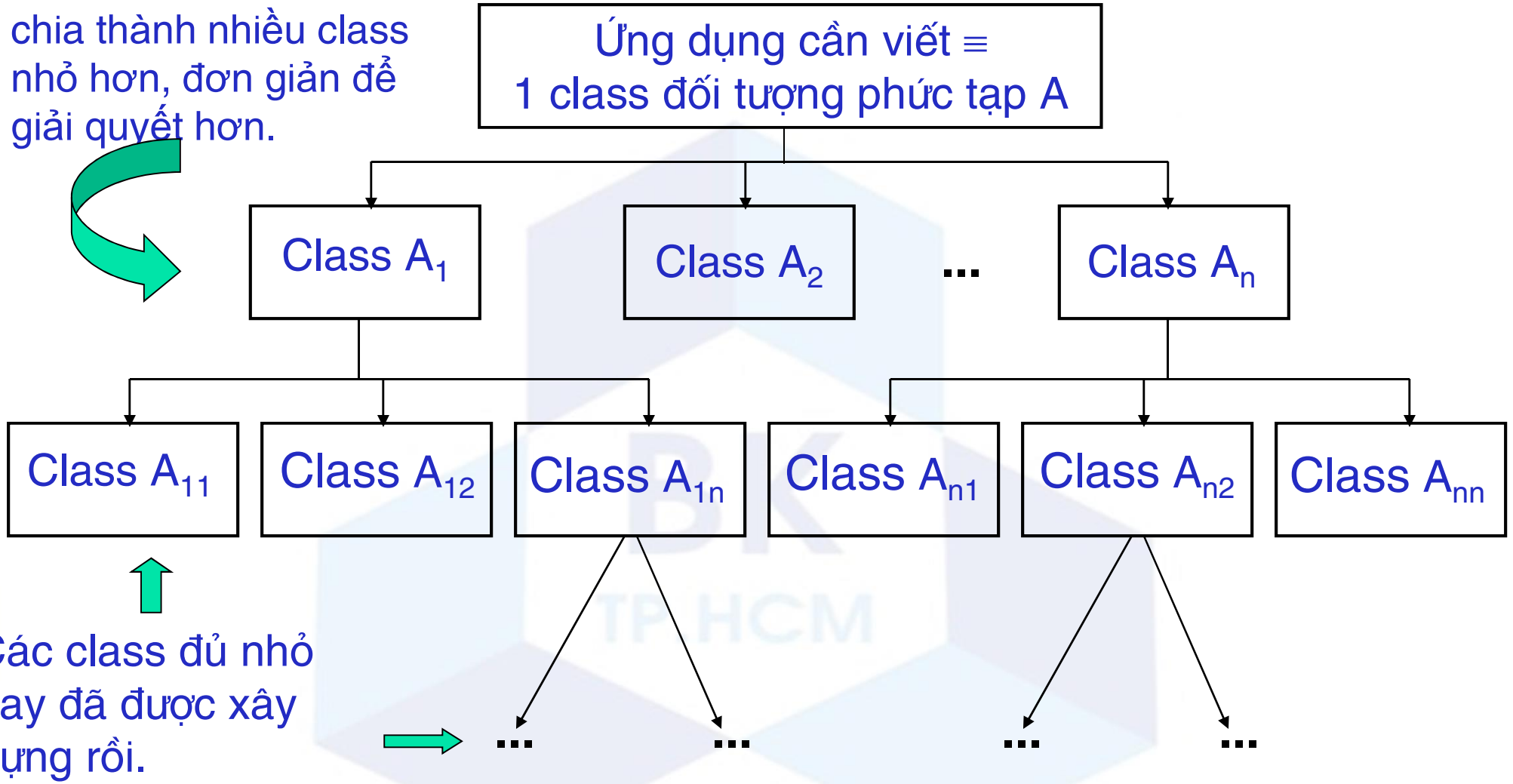
hay đơn giản hơn nữa :

```
int[][] array = {new int[]{1, 2,3}, new int[]{3, 4}, new int[] {5,6,7,8}};
```

2.5 Phương pháp phân tích từ-trên-xuống

- Như đã thấy ở slide trước, nếu ứng dụng lớn chứa rất nhiều hàm chức năng và phải xử lý rất nhiều dữ liệu thì rất khó quản lý chúng trong 1 class đơn giản cần 1 cấu trúc phù hợp để quản lý ứng dụng lớn. Phương pháp được dùng phổ biến nhất là phương pháp phân tích top-down.
- Nội dung của phương pháp này là phân rã class ứng dụng lớn thành n class nhỏ hơn (với n đủ nhỏ để việc phân rã đơn giản). Mỗi class nhỏ hơn, nếu còn quá phức tạp, lại được phân rã thành m class nhỏ hơn nữa (với m đủ nhỏ), cứ như vậy cho đến khi các class tìm được hoặc là class đã xây dựng rồi hoặc là class khá đơn giản, có thể xây dựng dễ dàng.
- Hình vẽ của slide kế cho thấy trực quan của việc phân tích top-down theo hướng đối tượng.

2.5 Phương pháp phân tích từ-trên-xuống



2.6 Package

- Trên mỗi máy có 1 hệ thống quản lý các đối tượng được dùng bởi nhiều ứng dụng đang chạy. Mỗi ứng dụng lớn gồm rất nhiều class đối tượng khác nhau. Mỗi phần tử trong hệ thống tổng thể đều phải có tên nhận dạng duy nhất. Để đặt tên các phần tử trong hệ thống lớn sao cho mỗi phần tử có tên hoàn toàn khác nhau (để tránh tranh chấp, nhập nhằng), Java cung cấp phương tiện package (giống như thư mục, folder).
- Package là 1 gói chứa nhiều phần tử như enum, class, interface và package con.... Mỗi package có tên theo dạng phân cấp. Để truy xuất 1 phần tử trong package, ta phải dùng tên dạng phân cấp, thí dụ javax.swing.JButton là tên của class JButton, class miêu tả đối tượng giao diện button trong các form ứng dụng.

2.6 Package

- Trong file mã nguồn Java, để truy xuất 1 phần tử trong package khác, ta có thể dùng 1 trong 2 cách :
  - dùng tên tuyệt đối dạng cây phân cấp. Thí dụ :

```
//định nghĩa 1 biến Button
javax.swing.JButton objButton;
```

  - dùng lệnh import <tên tuyệt đối>; Kể từ đây, ta có thể dùng tên cục bộ của phần tử được import. Thí dụ :

```
import javax.swing.*;  //import mọi phần tử trong package
JButton objButton;     //định nghĩa 1 biến Button
JTextField objText;    //định nghĩa 1 biến TextBox
```

2.6 Package

- Sun đã xây dựng sẵn hàng ngàn class, interface chức năng phổ biến và đặt chúng trong nhiều package khác nhau :
  - java.lang chứa các class và interface chức năng cơ bản nhất của hệ thống như System (nhập/xuất văn bản), Math (các hàm toán học),..
  - javax.swing chứa các đối tượng giao diện phổ dụng như JButton, JTextField, JList, JComboBox,...
  - java.awt chứa các đối tượng phục vụ xuất dữ liệu ra thiết bị vẽ như class Graphics,...
  - java.io chứa các class nhập/xuất dữ liệu ra file.
  - java.sql chứa các class truy xuất database theo chuẩn ODBC.
  - ...

2.7 Kết chương

- Chương này đã giới thiệu cấu trúc của chương trình Java nhỏ và đơn giản gồm 1 số biến dữ liệu và 1 số hàm xử lý các biến dữ liệu, từ đó tổng kết các kiểu dữ liệu khác nhau có thể được dùng trong 1 chương trình.
- Chương này cũng đã giới thiệu phương pháp đặt tên cho các phần tử cấu thành ứng dụng lớn 1 cách khoa học dựa trên package, cách chứa các phần tử cấu thành ứng dụng lớn trong các module vật lý.