# Java Object-Oriented Programming

- Giảng viên : Nguyễn Đức Hiển
  - Email : ndhien@udn.vn
  - Website :

- Thời lượng
  - Lý thuyết : 2 tín chỉ (30 tiết)
  - Thực hành + thảo luận : 1 tín chỉ

# Lập trình hướng đối tượng Java

## (Java Object-Oriented Programming)

# Nội dung

- Mô hình hướng đối tượng
- Lớp và đối tượng
- Cách xây dựng lớp với Java
- Một số gói chuẩn của Java

# Mô hình hướng đối tượng

- Chương trình được tổ chức xung quanh các đối tượng hơn là các chức năng.
- Thiết chương trình trên cơ sở dữ liệu được định nghĩa như thế nào và cách nó có thể được thao tác hơn là thứ tự lôgic của chương trình.
- Java nắm lấy mô hình này như lõi của thiết kế của nó.

# Lớp trong Java

- Một lớp định nghĩa một kiểu dữ liệu mới chứa:
  - Các trường (các thành viên dữ liệu, các biến thực thể,…)
  - Các phương thức (các thủ tục thao tác trên các trường hay cung cấp chức năng khác nào đó)
- Mỗi thể hiện của một lớp (đối tượng) có một sự sao chép của tất cả các trường không tĩnh và các phương thức được định nghĩa trong lớp đó.
- Chỉ một bản sao của các trường tĩnh và các phương thức tĩnh tồn tại cho mỗi lớp.

# Khai báo lớp

```
[Ket nhap cac thu vien];

public class <Ten lop> [extends <Lop co so>]
{
    [ Khai bao cac truong du lieu ];
     ...
    [ Khai bao cac phuong thuc ];
}
```

# Trường dữ liệu (fields)

- Cách khai báo trường dữ liệu của lớp tương tự như khai báo biến trong chương trình.
- Cú pháp:
  - [Cách truy cập ] [Cách cập nhật] <kiểu dữ liệu> <Tên trường> [ = giá trị];
  - Cách truy cập
    - public
    - protected
    - private
  - Cách cập nhật
    - static
    - final

# Thuộc tính truy cập

- public
  - Có thể được truy cập từ ngoài
- private
  - Có thể được truy cập bởi bất kỳ phương thức nào bên trong lớp.
- protected
  - Có thể được truy cập bởi bất kỳ lớp nào trong cùng gói và các lớp dẫn xuất.
- Lưu ý nếu không chỉ rõ thuộc tính truy cập thì mặc định là public.

# Phương thức

- Phương thức được định nghĩa như là một hành động hay hành vi của đối tượng.
- Cú pháp:

  [Cách truy cập] [Cách cập nhật ] <kiểu trả về>
  <Tên phương thức> [ throws <biệt lệ>]
  {
  <Các lệnh của phương thức>
  }

  - Cách cập nhật
    - static
    - final
    - abstract

# Ví dụ lớp Circle

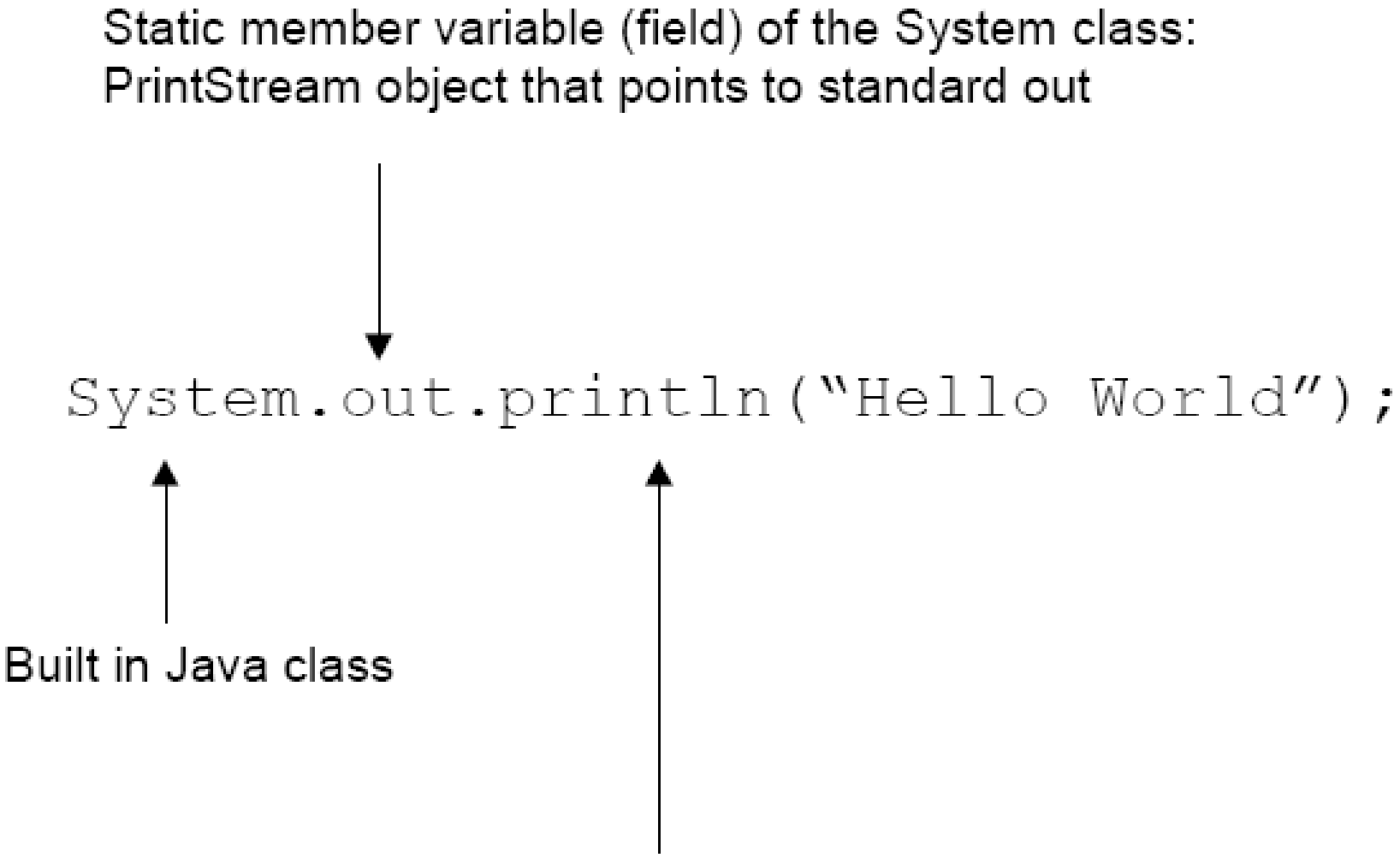
```
public class Circle
{
    ////Cac truong du lieu
    static final double PI = 3.14;
    double radius;

    ////Cac phuong thuc lop
    public double area()
    {
        //// Ma thuc hien cua phuong thuc
    }
    public double round()
    {
        //// Todo
    }
}
```

# Thành viên tĩnh (static)

- Các trường tĩnh
  - Có thể được truy nhập từ bên ngoài của lớp bằng cách sử dụng tên lớp
  - Có thể được truy nhập từ bên trong bất kỳ phương thức thành viên lớp nào mà không có tên lớp
- Các phương thức tĩnh
  - Không được truy nhập tới phương thức không tĩnh hay các trường của lớp
  - Có ý nghĩa một khi các thành viên tĩnh không liên quan đến bất kỳ đối tượng cụ thể nào và thậm chí tồn tại trước khi đối tượng của lớp được tạo.

# Ví dụ


Static member variable (field) of the System class:
PrintStream object that points to standard out
System.out.println("Hello World");
Built in Java class
Member method of the PrintStream class

# Khởi tạo dữ liệu

- Ba cách để khởi tạo các biến thành viên lớp:
  - Ngay trong thân lớp khi khai báo
  - Khối khởi tạo
  - Phương thức khởi tạo (Constructor)

```
class MyClass
{
    String name = "Java 5.0";
    int a = 10;
    ...
}
```

```
class MyClass {
    String name;
    int a;
    MyClass() {
        name = "Java 5.0";
        a = 0;
    }
}
```

```
class MyClass
{
    String name;
    int a;
    {
        name = "Java 5.0";
        a = 10;
    }
}
```

# Phương thức khởi tạo (Constructor)

- Constructor là một phương thức đặc biệt được dùng để khởi tạo các thành viên lớp với dữ liệu được xác định trong thời gian khởi tạo.
- Constructor được khai báo trùng tên với tên lớp và không có kiểu trả về.
- Một số lưu ý:
  - Nếu bạn không tạo contructor, Java tự động tạo ra một constructor mặc định không có đối số và không làm gì cả.
  - Nếu bạn đã tạo ra một constructor, constructor mặc định sẽ không được tạo ra.

# Ví dụ

```
public class Circle
{
    static final double PI = 3.14;
    double radius;

    public Circle(){
        radius = 0;
    }
    public Circle( double a){
        radius = a;
    }
}
```

# Phương thức nạp chồng (overloading)

- Các phương thức có tên giống nhau trong một lớp nhưng có các đối số khác nhau.
- Ví dụ:

```
public class Car
{
    String model;
    double cost;

    public double getPrice() {
        return cost;
    }
    public double getPrice( double vat) {
        return cost + cost * vat;
    }
}
```

# Biến this

- Tồn tại bên trong lớp và tham chiếu đến đối tượng hiện hành (this current object)
- Dùng để chỉ rõ phạm vi các thành viên của lớp
- Ví dụ:

```
public class Circle
{
    static final double PI = 3.14;
    double radius;
    ...
    public Circle( double radius){
        this.radius = radius;
    }
    ....
}
```

# Phương thức finalize()

- Java không có phương thức huỷ bỏ đối tượng (destructor)
- Java có các trình dọn dẹp cài đặt sẵn (garbage collection system), còn gọi là bộ thu gom rác (Garbage Collector), nó tự động dọn sạch các đối tượng không còn được tham chiếu trong chương trình.
- Mỗi lớp có phương thức finalize() được gọi khi trình dọn dẹp, trước khi xoá một đối tượng.
- Ta có thể phụ dọn dẹp một số tiến trình không còn tác dụng bằng cách cài đặt phương thức finalize()

# Ví dụ tạo lớp và đối tượng

```
public class Product {
    public int value = 10;
    public static int count = 0;
    public Product(){
        count++;
    }
    public void show( ) {
        System.out.println("Product value: " + value);
        System.out.println("Total product: " + count);
    }
    public static void main(String args[ ]) {
        Product p1 = new Product( );
        Product p2 = new Product( );
        p2.value = 20;

        p1.show();
        p2.show( );
    }
}
```

?

# Môt số bài tập

- Lớp Point
  - Fields: x, y, count (static)
  - Methods: set( x, y), display(),…
- Lớp Circle
  - Fields: center (Point), radius, count (static)
  - Methods: set( center, radius), getCenter(), getRadius(), display(),…
- Lớp Stack
  - Fields: box (Object), top, count (static)
  - Methods: pop(), push(object), overflow(), empty(),…
- Lớp Queue

# Thừa kế (Inheritance)

- Đôi khi thiết kế những lớp chúng ta gặp mối quan hệ sau đây
  - Class2 là một dạng đặc biệt của Class1
- Trong tình huống này, chúng ta không muốn sao lại tất cả các chức năng và thuộc tính trong Class1
- Thay vào đó chúng ta tạo ra Class2 như một lớp phụ (lớp dẫn xuất) của Class1
- Class2 thừa kế tất cả các trường và phương thức được cung cấp trong Class1 và cũng có thể định nghĩa chức năng bổ sung cho những những đặc điểm riêng của nó.

# Thừa kế (Inheritance)

- Để khai báo một lớp thừa kế từ một lớp khác sử dụng từ khóa extends
  - class SubClass extends BaseClass { …}
- Một lớp chỉ có thể là một lớp dẫn xuất của một lớp khác → Đơn thừa kế
- Lớp dẫn xuất sẽ thừa kế tất cả các trường và phương thức của lớp cơ sở
  - Các thành viên dữ liệu private của lớp cơ sở tồn tại trong lớp dẫn xuất nhưng chúng không được truy cập trực tiếp bởi bất kỳ phương thức nào của lớp dẫn xuất
  - Các thành viên dữ liệu static của lớp cơ sở cũng được thừa kế, có nghĩa rằng lớp dẫn xuất và lớp cơ sở chia sẻ một bản sao của các thành viên static.

# Phương thức khởi tạo lớp dẫn xuất

- Các constructor của lớp dẫn xuất nên đầu tiên có lời gọi một constructor của lớp cơ sở.
- Điều này có thể được thực hiện với từ khóa super

```
class ClassA extends ClassB {
    public ClassA() {
        super(...);
    }
}
```

- Một số lưu ý:
  - Nếu bạn không gọi một constructor của lớp cơ sở, Java sẽ tự động gọi constructor mặc định của lớp cơ sở lúc bắt đầu constructor của lớp
  - Nếu lớp cơ sở không có một constructor mặc định điều này sẽ phát sinh lỗi

# Phương thức nạp chồng (overloading)

- Những phương thức được nạp chồng:
  - Có mặt trong lớp cơ sở cũng như lớp dẫn xuất
  - Được định nghĩa lại trong lớp dẫn xuất
- Những phương thức được nạp chồng là một hình thức đa hình (polymorphism) trong quá trình thực thi
- Trong phương thức nạp chồng của lớp dẫn xuất muốn truy cập phiên bản của lớp cơ sở có thể sử dụng từ khóa super

# Ví dụ lớp thừa kế

```
class Person
{
    String  ten;
    int tuoi;
    public Person(){
        ten = "Nguyen Van B";
        tuoi = 0;
    }
    public void display(){
        System.out.print(ten);
        System.out.print(tuoi);
    }
}
```

```
class Student extends Person
{
    String lop;
    public Student(){
        super();
        lop ="K11T";
     }
     public void display(){
         super.dislay();
         System.out.print(lop);
    }
}
```

# Tính đa hình (Polymorphism)

- Sức mạnh thực sự của lập trình OOP là thông qua tính đa hình.
- Chẳng hạn chúng ta muốn làm việc với một danh sách các tài khoản người dùng trong một trường
- Chúng ta đã định nghĩa một lớp gọi là UserAccount với hai lớp dẫn xuất
  - StudentAccount
  - ProfessorAccount
- Tính đa hình cho phép chúng ta làm việc với một danh sách các UserAccount mà không biết rằng hay quan tâm kiểu tài khoản chúng là gì

# Tính đa hình (Polymorphism)

- Trong Java, chúng ta có thể gán một biến kiểu lớp nhất định bởi một thể hiện của lớp đó hay một thể hiện của lớp dẫn xuất của lớp đó
  - UserAccount myAccount = new StudentAccount();
- Bây giờ biến myAccount xem như một đối tượng kiểu UserAccount và nhưng thật sự nó là một đối tượng StudentAccount
- Chúng ta cũng có thể chuyển kiểu ngược trở lại kiểu StudentAccount nếu cần chức năng riêng của lớp StudentAccount
  - StudentAccount myStudentAccount = (StudentAccount)myAccount;

# Kiểm tra kiểu động

- Chẳng hạn, UserAccount có một phương thức có tên privileges() trả về một danh sách các quyền liên quan đến tài khoản người dùng
- StudentAccount và ProfessorAccount đều có phương thức nạp chồng privileges() với chức năng riêng của chúng.
- Biến UserAccount được khởi tạo với một đối tượng StudentAccount gọi phương thức privileges()
- Tại thời điểm chạy, Java kiểm tra kiểu của đối tượng này và thấy rằng nó là một StudentAccount và bởi vậy chạy phương thức privileges() phiên bản của StudentAccount thay vì phiên bản của UserAccount.

# Lớp trừu tượng (abstract)

- Các lớp trừu tượng cung cấp một nguyên mẫu nhưng không cài đặt cho một số phương thức của nó bởi vì ngữ cảnh của cài đặt chỉ quan trọng trong các lớp dẫn xuất
- Khai báo một lớp trừu tượng với từ khóa abstract

```
public abstract class Animal {
    private String type;
    public abstract void sound();
    public Animal(String type) {
        this.type = type;
    }
    public String toString() {
        return "This is a " + type;
    }
}
```

# Lớp trừu tượng (abstract)

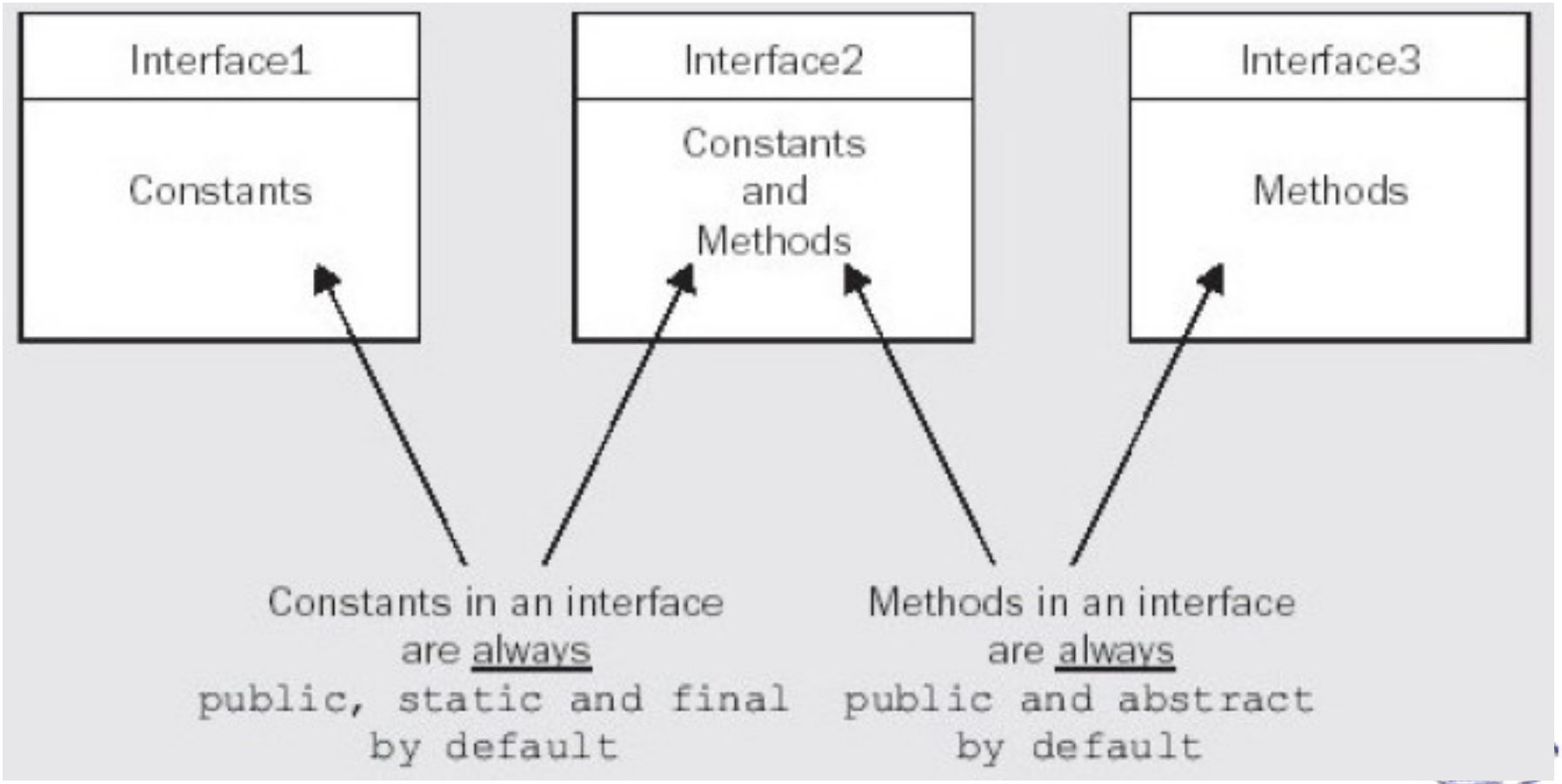
- Được dự định là các lớp cơ sở
- Không thể được khởi tạo (không tạo được đối tượng từ lớp trừu tượng)
- Bạn có thể khai báo các biến của một kiểu lớp trừu tượng nhưng chúng chỉ có thể chỉ được gán khởi tạo một lớp dẫn xuất của lớp trừu tượng này.

# Giao tiếp (Interface)

- Để tránh sự phức tạp của đa thừa kế, Java thay thế bằng các giao tiếp.
- Khai báo một dãy các phương thức xử lý nhưng không chưa được cài đặt
- Tất cả các phương thức của một giao tiếp tự động public và astract.
- Có thể định nghĩa các hằng trong giao tiếp

```
public interface MyShape {
    double PI = 3.14;
    double area();
    double round();
    String display();
}
```

# Giao tiếp (Interface)



# Giao tiếp (Interface)

- Java cung cấp sự hỗ trợ cho đa thừa kế giả thông qua việc sử dụng các giao tiếp
  - Các lớp chỉ có thể extends một và chỉ một lớp
  - Tuy nhiên, các lớp có thể implements nhiều giao tiếp
    - public class Circle implements MyShape {…}
- Một lớp implements một giao tiếp phải cài đặt các phương thức được khai báo trong giao tiếp.
- Nếu một lớp implements nhiều hơn một giao tiếp có các phương thức giống nhau, thì chỉ cần cài đặt một phương thức cho cả hai giao tiếp.

# Gói (Packages)

- Gói là cách tổ chức nhiều lớp có liên quan kết hợp với nhau.
- Tương tự như thư mục, gói dùng để lưu trữ các lớp, giao tiếp và các gói con khác. Đó là những thành viên của gói
- Bạn có thể đặt một lớp trong một gói bởi việc khai báo gói bởi từ khóa package
- Chú ý:
  - Java rất chặt chẻ với quy ước đặt tên và tổ chức file và thư mục của nó.
  - Tất cả các lớp mà bạn đặt vào trong một gói phải nằm trong một thư mục với tên gói đó.

# Ví dụ tạo lớp trong gói

```
package mypackage;

import java.io.*;

public class MyExample {
    public static void main(String agrs[]) {
        try {
            System.out.print( "Nhap mot ky tu: ");
            char ch = (char) System.in.read();
            System.out.print("Ky tu la: " + ch);
        }catch ( Exception ex){
            System.out.print("Loi nhap xuat!");
        }
    }
}
```

# Sử dụng gói

- Để kết nhập các lớp của gói vào chương trình sử dụng từ khóa import
  - Ví dụ: import java.io.*;
- Một số gói chuẩn của java:
  - java.lang.*
  - java.io.*
  - java.util.*
  - java.awt.*
  - java.awt.event.*
  - java.sql.*
  - java.net.*
  - …

# Gói java.lang.*

- Mặc định thì bất cứ chương trình Java nào cũng import gói java.lang.*
- Cung cấp các lớp bao bọc (Wrapper) cho các kiểu dữ liệu đơn nguyên

| Data type | Wrapper class |
|---|---|
| boolean | Boolean |
| byte | Byte |
| char | Character |
| double | Double |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |

# Các lớp Bao bọc (Wrapper)

- Mỗi kiểu dữ liệu đơn nguyên có tương ứng một kiểu dữ liệu tham chiếu “lớp Bao bọc”.
- Có thể được dùng để đại diện cho các giá trị dữ liệu đơn nguyên như những đối tượng tham chiếu khi nó cần dùng.
- Các lớp bao bọc cho một số kiểu dữ liệu đơn nguyên: Byte, Short, Integer, Long, Float, Double, Boolean, Character,…

# Lớp Object

- Mọi lớp trong Java chỉ extends một và chỉ một lớp
- Nếu ta không chỉ rõ một lớp cơ sở, lớp tạo ra sẽ tự động extends lớp Object
  - Mọi lớp trong Java chứa các phương thức chức năng cơ bản được định nghĩa trong lớp Object
  - Xem Java API để có thông tin đầy đủ về lớp Object
- Kết quả cuối cùng của thiết kế này đó là các lớp Java tạo nên một cây thừa kế lớn với lớp Object ở trên cùng.

# Lớp System

- Cung cấp những hạ tầng chuẩn như dòng nhập (Input), dòng xuất (Output) và dòng lỗi (Error)
- Cung cấp khả năng truy xuất đến những thuộc tính của hệ thống thực thi Java, và những thuộc tính môi trường như phiên bản, đường dẫn, nhà cung cấp…
- Phương thức:
  - exit(int)
  - getProperties()
  - setProperties()
  - currentTimeMillis()

# Lớp Math

- abs()
- ceil()
- floor()
- max()
- min()
- round()
- random()
- sqrt()
- sin()
- …

# Gói java.util.*

- Cung cấp phần lớn những lớp Java hữu dụng và thường xuyên cần đến trong hầu hết các ứng dụng
- Một số lớp:
  - Hashtable
  - Random
  - Vector
  - StringTokenizer
  - …

# Lớp Hastable

- Dùng lưu trữ dữ liệu kết hợp với khóa
  - Constructor: Hastable(), Hastable(int),…
  - Methods: clear(), done(), contains(obj), containskey(obj), elements(), get(key), isEmpty(), put(obj, key),…
- Ví dụ:

```
public class Practise {
   public static void main(String args[ ]) {
      Hashtable hash = new Hashtable();
      hash.put("one", new Integer(125));
      hash.put("two", new Integer(246));

      Integer item = (Integer) hash.get("one");
      if( item != null) {
         System.out.println("Data has 'one' key: " + item.intValue());
      }
   }
}
```

# Lớp Random

- Tạo ra những số ngẫu nhiên theo thuật toán pseudo
- Phương thức khởi tạo:
  - Random()
  - Random(long)
- Những phương thức nhận giá trị ngẫu nhiên:
  - nextDouble( )
  - nextFloat( )
  - nextGaussian( )
  - nextInt( )
  - nextLong( )
  - …

# Lớp Vector

- Cung cấp khả năng co giản cho mảng khi thêm hay bớt phần tử mảng
- Lưu trữ phần tử mảng kiểu Object, các phần tử có thể có kiểu khác nhau
- Một số phương thức:
  - Constructor: Vector(), Vector(int),…
  - Methods: addElement(obj), elementAt(int), elements(), firstElement(), insertElement(), isEmpty(), getSize(), setElemnetAt(obj, int), removeElementAt(int),…

# Lớp StringTokenizer

- Có thể được sử dụng để tách các chuỗi dựa vào các "dấu hiệu" như một dấu tách cố định.
- Dấu tách mặc định là một khoảng trắng (space)
- Ký tự tách có thể được chỉ định khi đối tượng StringTokenizer được khởi tạo
- Phương thức khởi tạo:
  - StringTokenizer(String input)
  - StringTokenizer(String input, String delimiters)
  - StringTokenizer(String input, String delimiters, Boolean)

# Lớp StringTokenizer

- Những phương thức của lớp StringTokenizer
  - countTokens( )
  - hasMoreElements( )
  - hasMoreTokens( )
  - nextElement( )
  - nextToken( )
- Cách sử dụng StringTokenizer

```
import java.util.StringTokenizer;
...
String st = "To be or not to be";
StringTokenizer stk = new StringTokenizer(st);
while( stk.hasMoreTokens()) {
    System.out.println("The next word is " +
        stk.nextToken());
}
```

# Ví dụ

```
import java.util.StringTokenizer;
class TokTest{
   public static void main(String[ ] args){
      if (args.length < 2)
         return;
      String input = args[0];
      String delimiters = args[1];
      StringTokenizer  tok = new StringTokenizer (input, delimiter);
      while (tok.hasMoreTokens()) {
         System.out.println( tok.nextToken());
      }
   }
}
```

```
C:\>java TokTest http://www.microsoft.com/~gates/ :/.
http
www
Microsoft
Com
~gates
```

**Thanks for listenning!!!**