# JAVA

# Chương 1

# Giới thiệu ngôn ngữ lập trình Java

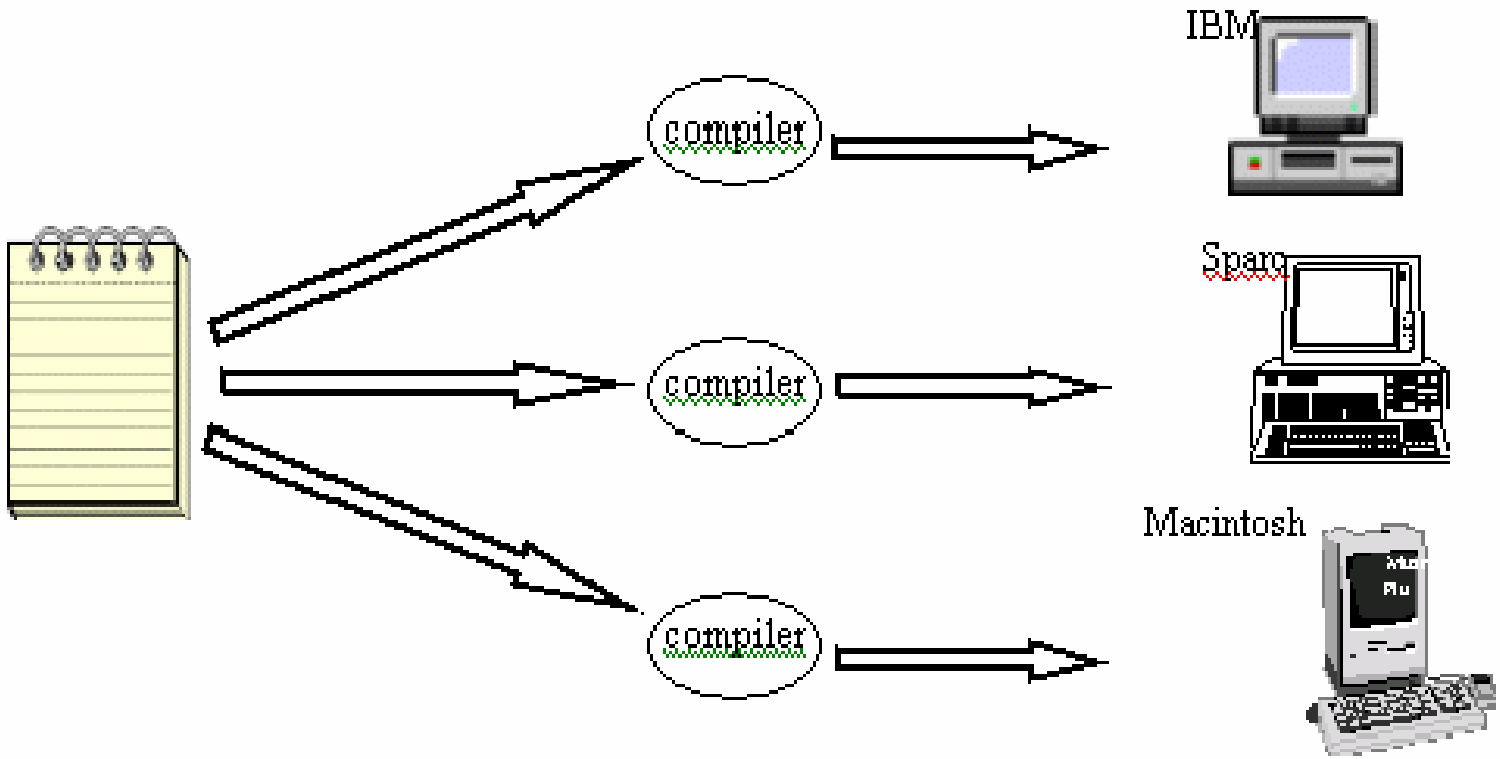
# Giới thiệu

- Sự phát triển của Java
- Hướng tới người dùng
- Giống với C / C++

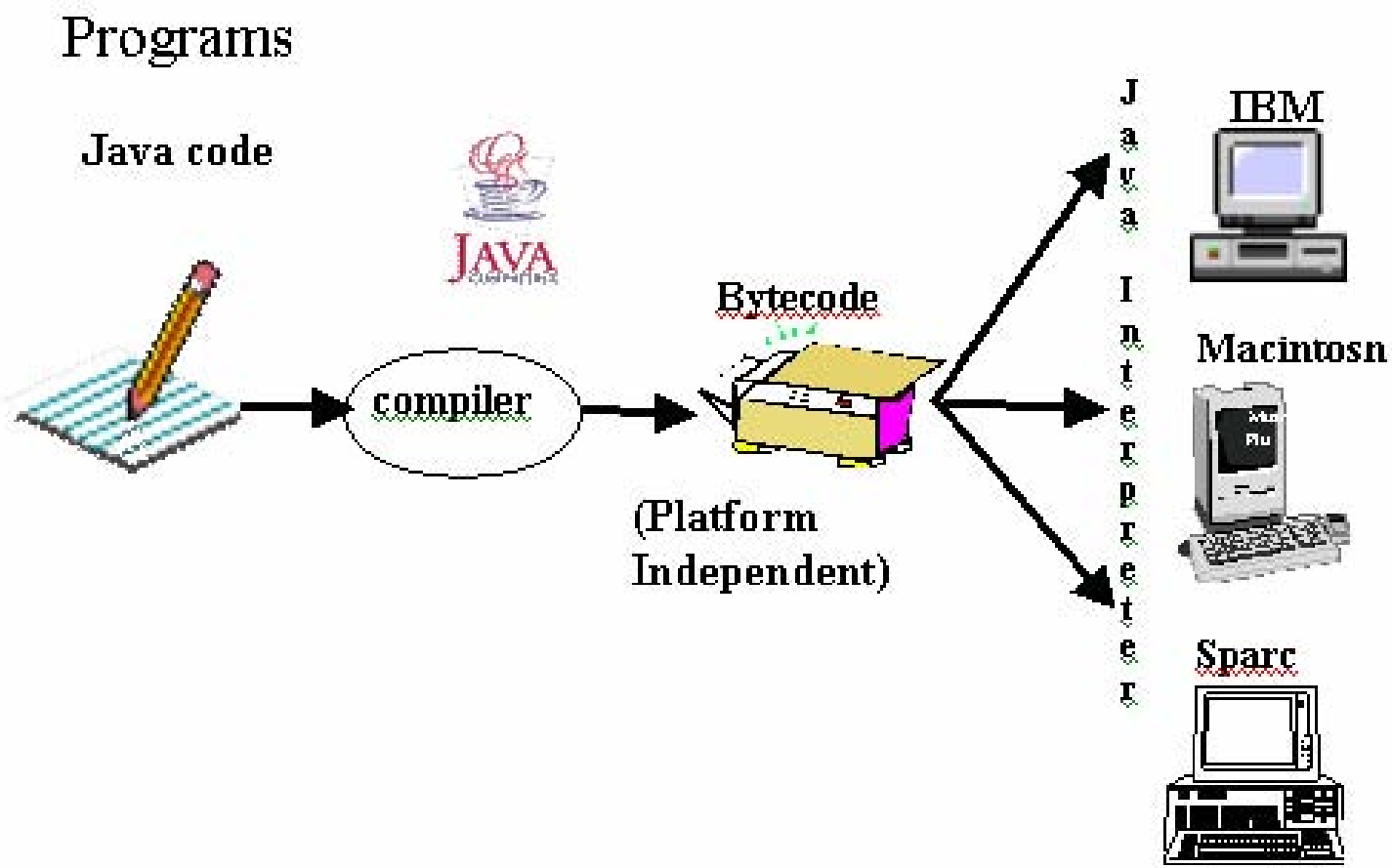
# Các đặc trưng của Java

- Đơn giản
- Hướng đối tượng
- Độc lập phần cứng
- Mạnh
- Bảo mật
- Phân tán
- Đa luồng
- Động

# Các chương trình dịch truyền thống



# Chương trình dịch Java



# Các loại chương trình Java

- Applets
- Ứng dụng độc lập (console Application)
- Ứng dụng giao diện (GUI Application)
- Servlet
- Ứng dụng cơ sở dữ liệu

# Máy ảo Java

- Là một phần mềm dựa trên cơ sở máy tính ảo
- Là tập hợp các lệnh logic để xác định hoạt động của máy tính
- Được xem như là một hệ điều hành thu nhỏ
- Nó thiết lập lớp trừu tượng cho:
  - Phần cứng bên dưới
  - Hệ điều hành
  - Mã đã biên dịch

# Quá trình dịch chương trình Java

- Trình biên dịch chuyển mã nguồn thành tập các lệnh không phụ thuộc vào phần cứng cụ thể
- Trình thông dịch trên mỗi máy chuyển tập lệnh này thành chương trình thực thi
- Máy ảo tạo ra một môi trường để thực thi các lệnh bằng cách:
  - Nạp các file .class
  - Quản lý bộ nhớ
  - Dọn “rác”

# Trình dịch Java
# Java Development Kit

- Java 1.0 - Sử dụng lần đầu vào năm 1995
- Java 1.1 – Đưa ra năm 1997
- Java 2 – Phiên bản mới nhất

# Bộ công cụ JDK

- Trình biên dịch, 'javac'
  - **javac [options] sourcecodename.java**
- Trình thông dịch, 'java'
  - **java [options] classname**
- Trình dịch ngược, 'javap'
  - **javap [options] classname**
- Công cụ sinh tài liệu, 'javadoc'
  - **javadoc [options] sourcecodename.java**

- Chương trình tìm lỗi - Debug, 'jdb‘
  - **jdb [options] sourcecodename.java**

    OR
  - **jdb -host -password [options] sourcecodename.java**

- Chương trình xem Applet , 'appletviewer‘
  - **appletviewer [options] sourcecodename.java / url**

# Các gói chuẩn của Java

- **java.lang**
- **java.applet**
- **java.awt**
- **java.io**
- **java.util**
- **java.net**
- **java.awt.event**
- **java.rmi**
- **java.security**
- **java.sql**

# Các đặc trưng mới của Java2

- Swing
- Kéo và thả
- Java 2D API
- Âm thanh
- RMI

# Chương 2
# Các phần tử cơ bản ngôn ngữ Java

# Cấu trúc một chương trình Java

- Xác lập thông tin môi trường
- Khai báo lớp đối tượng (Class)
- Các thành phần (Tokens):
  - Định danh
  - Từ khóa / từ dự phòng
  - Ký tự phân cách
  - Nguyên dạng (Literals)
  - Toán tử

# Ví dụ một chương trình Java mẫu

```
//  This is a simple program called “Ex1.java”

class Ex1
{
    public static void main(String args[])
    {
        System.out.println(“My first program in Java”);
    }
}
```

# Biên dịch chương trình java

- **..\jdk\bin>javac Ex1.java**
- **..\jdk\bin>java Ex1**

- Kết quả:

  **My first program in Java**

# Truyền đối số trong dòng lệnh

```
class Pass
{
   public static void main(String parameters[])
  {
     System.out.println("This is what the main method
     received");
     System.out.println(parameters[0]);
     System.out.println(parameters[1]);
     System.out.println(parameters[2]);
  }
}
```

# Truyền đối số trong dòng lệnh (Tiếp theo…)

C:\WINNT\System32\cmd.exe

```
D:\TEMP>java Pass We Change Lives
This is what the main method received
We
Change
Lives

D:\TEMP>_
```

# Các phần tử cơ bản củangôn ngữ Java

- Lớp và phương thức (Classes & Methods)
- Kiểu dữ liệu
- Biến số
- Toán tử
- Cấu trúc điều khiển

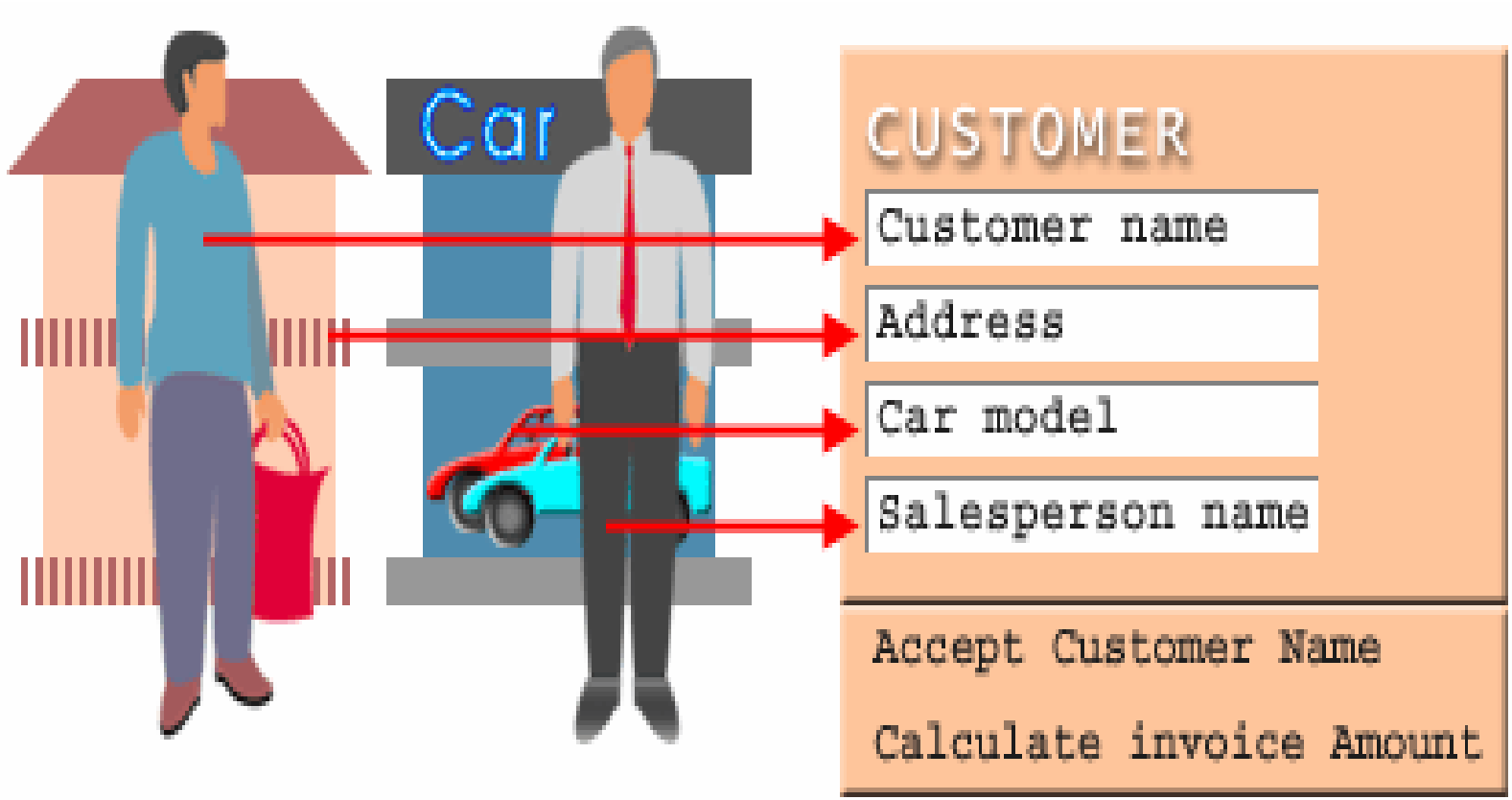
# Lớp trong Java

- Cú pháp khai báo lớp (Class)

```
class Classname
{
        var_datatype variablename;
        :

        met_datatype methodname(parameter_list)
        :
}
```

# Lớp mẫu


Car
CUSTOMER
Customer name
Address
Car model
Salesperson name
Accept Customer Name
Calculate invoice Amount

# Các lớp lồng nhau (Nested Classes)

- Việc định nghĩa một lớp bên trong một lớp khác được gọi là “xếp lồng” (Nesting)
- Các kiểu xếp lồng:
  - Tĩnh (Static)
  - Động (Non-static)

# Kiểu dữ liệu

- Kiểu dữ liệu cơ sở (Primitive Data Types)
- Kiểu dữ liệu tham chiếu (Reference data types)

# Kiểu dữ liệu cơ sở

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

# Kiểu dữ liệu tham chiếu

- Mảng (Array)
- Lớp (Class)
- Interface

# Ép kiểu (Type Casting)

- Kiểu dữ liệu này được chuyển đổi sang một kiểu dữ liệu khác
- Ví dụ

```
float c = 34.89675f;
int b = (int)c + 10;
```

# Biến số

- Khai báo biến số gồm 3 thành phần:
  - Kiểu dữ liệu của biến số
  - Tên biến
  - Giá trị ban đầu của biến (không bắt buộc)
- Cú pháp

**datatype identifier [=value][, identifier[=value]...];**

# Những từ khóa của Java

| | | | |
|---|---|---|---|
| abstract | boolean | break | byte |
| case | catch | char | class |
| const | continue | default | do |
| double | else | extends | final |
| finally | float | for | goto |
| if | implements | import | instanceof |
| int | interface | long | native |
| new | package | private | protected |
| public | return | short | static |
| super | switch | synchronized | this |
| throw | throws | transient | try |
| void | volatile | while | |

# Khai báo mảng

- Ba cách để khai báo mảng:
  - **datatype identifier [ ];**
  - **datatype identifier [ ] = new datatype[size];**
  - **datatype identifier [ ]= {value1,value2,….valueN};**

# Phương thức (Methods in Classes)

- Phương thức được định nghĩa như là một hành động hoặc một tác vụ thật sự của đối tượng
- Cú pháp

```
access_specifier modifier datatype
    method_name(parameter_list)
{
//body of method
}
```

# Ví dụ về sử dụng phương thức

```
class Temp {
    static int x = 10;                    // variable
    public static void show( ) {    // method
        System.out.println(x);
    }
    public static void main(String args[ ]) {
       Temp t = new Temp( );            // object 1
        t.show( );                      // method call
       Temp t1 = new Temp( );        // object 2
        t1.x = 20;
        t1.show( );
   }
}
```

# Access specifiers

- public
- private
- protected

# Method Modifiers

- static
- abstract
- final
- native
- synchronized
- volatile

# Những phương thức được nạp chồng :
(Methods Overloading)

- Những phương thức được nạp chồng :
  - Cùng ở trong một lớp
  - Có cùng tên
  - Khác nhau về danh sách tham số
- Những phương thức được nạp chồng là một hình thức đa hình (polymorphism) trong quá trình biên dịch (compile time)

# Ghi đè phương thức (Methods Overriding)

- Những phương thức được ghi đè:
  - Có mặt trong lớp cha (superclass) cũng như lớp kế thừa (subclass)
  - Được định nghĩa lại trong lớp kế thừa (subclass)
- Những phương thức được ghi đè là một hình thức đa hình (polymorphism) trong quá trình thực thi (Runtime)

# Phương thức khởi tạo (Class Constructors)

- Là một phương thức đặc biệt dùng để khởi tạo giá trị cho các biến thành viên của lớp đối tượng
- Có cùng tên với tên lớp và không có giá trị trả về
- Được gọi khi đối tượng được tạo ra
- Có 2 loại:
  - Tường minh (Explicit constructors)
  - Ngầm định (Implicit constructors)

## Phương thức khởi tạo của lớp dẫn xuất (Derived class constructors)

- Có cùng tên với lớp dẫn xuất (subclass)
- Mệnh đề gọi constructor của lớp cha (superclass) phải là mệnh đề đầu tiên trong constructor của lớp dẫn xuất (subclass)

# Các toán tử

- Các loại toán tử:
  - Toán tử số học (Arithmetic operators)
  - Toán tử dạng Bit (Bitwise operators)
  - Toán tử so sánh (Relational operators)
  - Toán tử logic (Logical operators)
  - Toán tử điều kiện (Conditional operator)
  - Toán tử gán (Assignment operator)

# Toán tử số học
# Arithmetic Operators

| | |
|---|---|
| + | Addition (Phép cộng) |
| - | Subtraction (Phép trừ) |
| * | Multiplication (Phép nhân) |
| / | Division (Phép chia) |
| % | Modulus (Lấy số dư) |
| ++ | Increment (Tăng dần) |
| -- | Decrement (Giảm dần) |

| | |
|---|---|
| += | Phép cộng và gán |
| -= | Phép trừ và gán |
| *= | Phép nhân và gán |
| /= | Phép chia và gán |
| %= | Phép lấy số dư và gán |

# Toán tử Bit
# (Bitwise Operators)

| | |
|---|---|
| ~ | Phủ định (NOT) |
| & | Và (AND) |
| \| | Hoặc (OR) |
| ^ | Exclusive OR |
| >> | Dịch sang phải (Shift right) |
| << | Dịch sang trái (Shift left) |

# Toán tử so sánh (Relational Operators)

| | |
|---|---|
| == | So sánh bằng |
| != | So sánh khác |
| < | Nhỏ hơn |
| > | Lớn hơn |
| <= | Nhỏ hơn hoặc bằng |
| >= | Lớn hơn hoặc bằng |

# Toán tử Logic
# (Logical Operators )

| | |
|---|---|
| && | Logical AND |
| \|\| | Logical OR |
| ! | Logical unary NOT |

# Toán tử điều kiện (Conditional Operator)

- Cú pháp

  **Biểu thức 1 ? Biểu thức 2 : Biểu thức 3;**

- **Biểu thức 1**

  Điều kiện kiểu Boolean trả về giá trị True hoặc False

- **Biểu thức 2**

  Trả về giá trị nếu kết quả của mệnh đề 1 là True

- **Biểu thức 3**

  Trả về giá trị nếu kết quả của mệnh đề 1 là

# Toán tử gán (Assignment Operator)

= Assignment (Phép gán)

Giá trị có thể được gán cho nhiều biến số

- Ví dụ

  **a = b = c = d = 90;**

# Thứ tự ưu tiên của các toán tử

| Thứ tự | Toán tử |
|---|---|
| 1. | trong ngoặc tính trước |
| 2. | Các toán tử đơn như +,-,++,-- |
| 3. | Các toán tử số học và các toán tử dịch như *,/,+,-,<<,>> |
| 4. | Các toán tử quan hệ như >,<,>=,<=,= =,!= |
| 5. | Các toán tử logic và Bit như &&,II,&,I,^ |
| 5. | Các toán tử gán như =,*=,/=,+=,-= |

- Thứ tự của các toán tử có thể được thay đổi bằng cách sử dụng các dấu ngoặc đơn trong mệnh đề

# Các kí tự định dạng xuất dữ liệu (Escape Sequences)

| **Escape Sequence** | **Mô tả** |
|---|---|
| \n | Xuống dòng mới |
| \r | Chuyển con trỏ đến đầu dòng hiện hành |
| \t | Chuyển con trỏ đến vị trí dừng Tab kế tiếp (ký tự Tab) |
| \\ | In dấu \ |
| \’ | In dấu nháy đơn (’) |
| \’’ | In dấu nháy kép (’’) |

# Các lệnh điều khiển

- Điều khiển rẻ nhánh:
  - Mệnh đề **if-else**
  - Mệnh đề **switch-case**

- Vòng lặp (Loops):
  - Vòng lặp **while**
  - Vòng lặp **do-while**
  - Vòng lặp **for**

# Lệnh **if-else**

- Cú pháp

```
if (condition)
{
action1 statements;
}
else
{
action2 statements;
}
```

# Lệnh **switch-case**

- Cú pháp

```
switch (expression)
{
    case 'value1': action1 statement(s);
                   break;
    case 'value2': action2 statement(s);
                   break;
    :
    :
    case 'valueN': actionN statement(s);
                   break;
```

# Lệnh lặp **while**

- Cú pháp

```
while(condition)
{
        action statements;
        :
        :
}
```

# Lệnh lặp **do-while**

- Cú pháp

```
do
{
        action statements;
        :
        :
} while(condition);
```

# Vòng lặp **for**

- Cú pháp

```
for(initialization statements; condition; increment statements)
{
    action statements;
    :
    :
}
```

# Chương 3

# Gói & Interface
# (Packages & Interfaces)

# Giới thiệu

- Những thành phần cơ bản của 1 chương trình Java:
  - Gói (Packages)
  - Giao diện (Interfaces)
- Những phần của một chương trình Java:
  - Lệnh khai báo gói(**package )**
  - Lệnh chỉ định gói được dùng (Lệnh i**mport)**
  - Khai báo lớp public (một file java chỉ chứa 1 lớp public class)
  - Các lớp khác (classes private to the package)
- Tập tin nguồn Java có thể chứa tất cả hoặc một vài trong số các phần trên.

# Interfaces

- Chương trình Java chỉ có thể kế thừa từ 1 lớp duy nhất trong cùng một thời điểm, nhưng có thể dẫn xuất cùng lúc nhiều Interfaces
- Không được phép có những phương thức cụ thể (concrete methods)
- interface cần phải được hiện thực (implements).

# Các bước tạo interface

- Định nghĩa Interface
- Biên dịch Interface
- Hiện thực Interface

- Tính chất của interface:
  - Tất cả phương thức trong interface phải là **public.**
  - Các phương thức phải được định nghĩa trong lớp dẫn xuất giao diện đó.

# Sử dụng Interface

- Không thể dẫn xuất từ lớp khác, nhưng có thể dẫn xuất từ những interface khác
- Nếu một lớp dẫn xuất từ một interface mà interface đó dẫn xuất từ các interface khác thì lớp đó phải định nghĩa tất cả các phương thức có trong các interface đó
- Khi định nghĩa một interface mới thì một kiểu dữ liệu tham chiếu cũng được tạo ra.

# Gói (Packages)

- Tương tự nhưthư mục lưu trữ những lớp, interface và các gói con khác. Đó là những thành viên của gói

- Những ưu điểm khi dùng gói (Package):
  - Cho phép tổ chức các lớp vào những đơn vị nhỏ hơn
  - Giúp tránh được tình trạng trùng lặp khi đặt tên.
  - Cho phép bảo vệ các lớp đối tượng
  - Tên gói (Package) có thể được dùng để nhận dạng chức năng của các lớp.

- Những lưu ý khi tạo gói:
  - Mã nguồn phải bắt đầu bằng lệnh ‘package’
  - Mã nguồn phải nằm trong cùng thư mục mang tên của gói
  - Tên gói nên bắt đầu bằng ký tự thường (lower case) để phân biệt giữa lớp đối tượng và gói
  - Những lệnh khác phải viết phía dưới dòng khai báo gói là mệnh đề **import**, kế đến là các mệnh đề định nghĩa lớp đối tượng
  - Những lớp đối tượng trong gói cần phải được biên dịch
  - Để chương trình Java có thể sử dụng những gói này, ta phải **import** gói vào trong mã nguồn

- Import gói (Importing packages):
  - Xác định tập tin cần được import trong gói
  - Hoặc có thể import toàn bộ gói

# Các bước tạo ra gói (Package)

- Khai báo gói
- Import những gói chuẩn cần thiết
- Khai báo và định nghĩa các lớp đối tượng có trong gói
- Lưu các định nghĩa trên thành tập tin .**java**, và biên dịch những lớp đối tượng đã được định nghĩa trong gói.

# Sử dụng những gói do người dùng định nghĩa (user-defined packages)

- Mã nguồn của những chương trình này phải ở cùng thư mục của gói do người dùng định nghĩa.
- Để những chương trình Java khác sử dụng những gói này, import gói vào trong mã nguồn
- Import những lớp đối tượng cần dùng
- Import toàn bộ gói
- Tạo tham chiếu đến những thành viên của gói

# Xác lập CLASSPATH

- Là danh sách các thư mục, giúp cho việc tìm kiếm các tập tin lớp đối tượng tương ứng
- Nên xác lập CLASSPATH trong lúc thực thi (runtime), vì như vậy nó sẽ xác lập đường dẫn cho quá trình thực thi hiện hành

# Gói và điều khiển truy xuất (Packages & Access Control)

| | public | protected | No modifier | private |
|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes |
| Same package subclass | Yes | Yes | Yes | No |
| Same package non-subclass | Yes | Yes | Yes | No |
| Different package subclass | Yes | Yes | No | No |
| Different package non-subclass | Yes | No | No | No |

# Gói java.lang

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

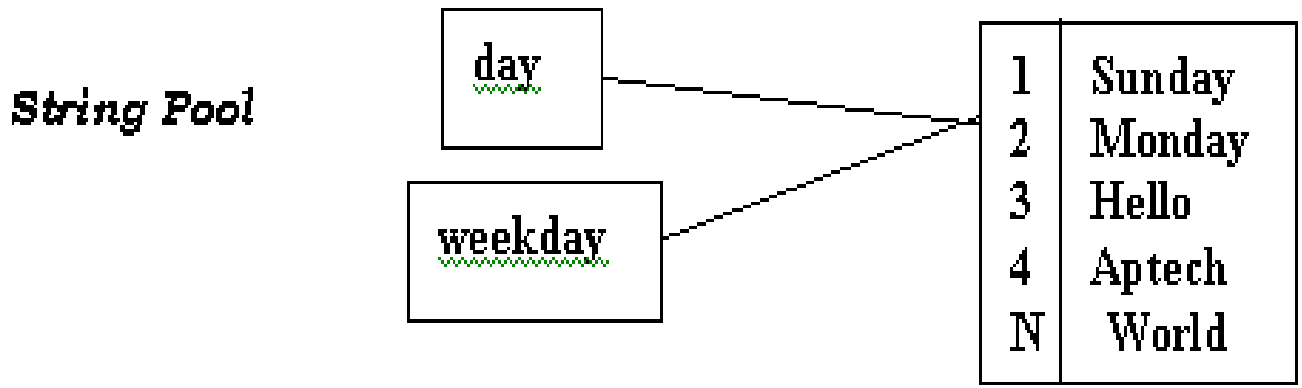
| Data type | Wrapper class |
|---|---|
| boolean | Boolean |
| byte | Byte |
| char | Character |
| double | Double |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |

# Lớp **String**

- Phương thức khởi tạo (Constructor):
  - **String str1 = new String();**
  - **String str2 = new String(“Hello World”);**
  - **char ch[ ] = {“A”,”B”,”C”,”D”,”E”};**
  - **String str3 = new String(ch);**
  - **String str4 = new String(ch,0,2);**

# String Pool

- ‘String Pool’ đại diện cho tất cả các ký tự được tạo ra trong chương trình
- Khái niệm ‘String Pool’



# Những phương thức của lớp **String**

- **charAt( )**
- **startsWith()**
- **endsWith( )**
- **copyValueOf( )**
- **toCharArray( )**
- **indexOf( )**
- **toUpperCase( )**
- **toLowerCase( )**
- **trim( )**
- **equals( )**

# Lớp **StringBuffer**

- Cung cấp những phương thức khác nhau để thao tác trên đối tượng string (chuỗi ký tự)
- Những đối tượng của lớp này khá linh hoạt
- Cung cấp những phương thức khởi tạo (constructor) đã được nạp chồng (overloaded)
- Những phương thức của lớp **StringBuffer**:
  - **append( )**
  - **insert( )**
  - **charAt( )**
  - **setCharAt( )**
  - **setLength( )**
  - **getChars( )**
  - **reverse( )**

# Lớp **java.lang.Math**

- **abs()**
- **ceil()**
- **floor()**
- **max()**
- **min()**
- **round()**
- **random()**
- **sqrt()**
- **sin()**
- **cos()**
- **tan()**

# Lớp **Runtime**

- Đóng gói (Encapsulates) môi trường thực thi
- Dùng để quản lý bộ nhớ, và thi hành những tiến trình cộng thêm
- Phương thức:
  - **exit(int)**
  - **freeMemory( )**
  - **getRuntime( )**
  - **gc( )**
  - **totalMemory( )**
  - **exec(String)**

# Lớp **System**

- Cung cấp những hạ tầng chuẩn như nhập (Input), xuất (Output) và các luồng lỗi(Error Streams)
- 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)**
  - **gc()**
  - **getProperties()**
  - **setProperties()**
  - **currentTimeMillis()**
  - **arraycopy(Object, int, Object, int, int)**

# Lớp **Class**

- Thể hiện (Instance) của lớp này che giấu tình trạng thực thi của đối tượng trong một ứng dụng Java
- Đối tượng hoặc thể hiện của lớp này có thể tạo ra bằng 1 trong 3 cách sau:
  - Sử dụng phương thức **getClass( )** của đối tượng
  - Sử dụng phương thức tĩnh **forName( )** của lớp để tạo ra một thể hiện của lớp đó trong lúc đặt tên cho lớp
  - Sử dụng đối tượng ClassLoader để nạp một lớp mới

# Lớp **Object**

- Là lớp cha (superclass) của tất cả các lớp
- Phương thức:
  - **equals(Object)**
  - **finalize()**
  - **notify()**
  - **notifyAll()**
  - **toString()**
  - **wait()**

# GÓI **java.util**

- Cung cấp phần lớn những lớp Java hữu dụng và thường xuyên cần đến trong hầu hết các ứng dụng
- Giới thiệu những lớp trừu tượng sau:
  - Hashtable
  - Random
  - Vector
  - StringTokenizer

# Lớp **Hashtable**

- Mở rộng lớp trừu tượng Dictionary
- Dùng để nối kết những khóa vào những giá trị cụ thể
- Phương thức khởi tạo Hashtable:
  - **Hashtable(int)**
  - **Hashtable(int, float)**
  - **Hashtable( )**

# Những phương thức của lớp **Hashtable**

- **clear()**
- **done()**
- **contains(Object)**
- **containsKey(Object)**
- **elements()**
- **get(Object key)**
- **isEmpty()**
- **keys()**
- **put(Object, Object)**
- **rehash()**
- **remove(Object key)**
- **size()**
- **toString()**

# Lớp **Random**

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

# Những phương thức của lớp **Random**

- **nextDouble()**
- **nextFloat()**
- **nextGaussian()**
- **nextInt()**
- **nextLong()**
- **setSeed(long)**

# Lớp **Vector**

- Cung cấp khả năng co giản cho mảng khi thêm phần tử vào mảng
- Lưu trữ những thành phần của kiểu Object
- Một Vector riêng rẽ có thể lưu trữ những phần tử khác nhau, đó là những instance của những lớp khác nhau
- Phương thức khởi tạo (Constructors):
  - **Vector(int)**
  - **Vector(int, int)**
  - **Vector()**

# Những phương thức của lớp **Vector**

- **addElement(Object)**
- **capacity( )**
- **clone( )**
- **contains(Object)**
- **copyInto(Object [ ])**
- **elementAt(int)**
- **elements( )**
- **ensureCapacity(int)**
- **firstElement( )**
- **indexOf(Object)**
- **indexOf(Object, int)**
- **insertElementAt(Objec**
- **isEmpty( )**
- **lastElement( )**
- **lastIndexOf(Object)**
- **lastIndexOf(Object, int)**
- **removeAllElements( )**
- **removeElement(Object)**
- **removeElementAt(int)**
- **setElementAt(Object, int)**

# Lớp **StringTokenizer**

- Có thể được dùng để tách một chuỗi thành những thành phần cấu thành của nó (constituent tokens)
- Ký tự phân cách có thể được chỉ định khi một đối tượng **StringTokenizer** được khởi tạo
- Phương thức khởi tạo (Constructors):
  - **StringTokenizer(String)**
  - **StringTokenizer(String, String)**
  - **StringTokenizer(String, String, Boolean)**
- Lớp **StringTokenizer** sử dụng giao diện liệt kê (enumeration interface)

# Những phương thức của lớp **StringTokenizer**

- **countTokens( )**
- **hasMoreElements( )**
- **hasMoreTokens( )**
- **nextElement( )**
- **nextToken( )**
- **nextToken(String)**

# Chương 4

# Xử lý biệt lệ

# Giới thiệu về biệt lệ

- Là một kiểu lỗi đặc biệt
- Nó xảy ra trong thời gian thực thi đoạn lệnh
- Thông thường các điều kiện thực thi chương trình gây ra biệt lệ
- Nếu các điều kiện này không được quan tâm, thì việc thực thi có thể kết thúc đột ngột

# Mục đích của việc xử lý biệt lệ

- Giảm thiểu việc kết thúc bất thường của hệ thống và của chương trình.
- Ví dụ, thao tác xuất/nhập trong một tập tin, nếu việc chuyển đổi kiểu dữ liệu không thực hiện đúng, một biệt lệ sẽ xảy ra và chương trình bị hủy mà không đóng tập tin. Lúc đó tập tin sẽ bị hư hại và các nguồn tài nguyên được cập phát cho tập tin không được thu hồi lại cho hệ thống.

# Xử lý biệt lệ

- Khi một biệt lệ xảy ra, đối tượng tương ứng với biệt lệ đó sẽ được tạo ra.
- Đối tượng này sau đó được truyền tới phương thức nơi mà biệt lệ xảy ra.
- Đối tượng này chức các thông tin chi tiết về biệt lệ. Thông tin này có thể nhận được và xử lý.
- Lớp ’throwable’ mà Java cung cấp là lớp trên nhất của lớp biệt lệ.

# Mô hình xử lý biệt lệ

- Mô hình được biết đến là mô hình ‘catch and throw’
- Khi một lỗi xảy ra, biệt lệ sẽ được chặn và được vào một khối.
- Từ khóa để xử lý biệt lệ:
  - try
  - catch
  - throw
  - throws
  - finally

# Cấu trúc của mô hình xử lý biệt lệ

- **Cú pháp**

```
try { ....  }
catch(Exception e1) { .... }
catch(Exception e2) { .... }
catch(Exception eN) { .... }
finally { .... }
```

# Mô hình ‘Catch and Throw’ nâng cao

- Người lập trình chỉ quan tâm tới các lỗi khi cần thiết.
- Một thông báo lỗi có thể được cung cấp trong exception-handler.

# Khối ‘try’ và ‘catch’

- Được sử dụng để thực hiện trong mô hình ‘catch and throw’ của xử lý biệt lệ.
- Khối lệnh ‘try’ gồm tập hợp các lệnh thực thi
- Một phương thức mà có thể bắt biệt lệ, cũng bao gồm khối lệnh ‘try’.
- Một hoặc nhiều khối lệnh ‘catch’ có thể tiếp theo sau một khối lệnh ‘try’
- Khối lệnh ‘catch’ này bắt biệt lệ trong khối lệnh ‘try’.

# Khối lệnh ‘try’ và ‘catch’ Blocks (tt)

- Để bắt bất kỳ loại biệt lệ nào, ta phải chỉ ra kiểu biệt lệ là ‘Exception’

  **catch(Exception e)**

- Khi biệt lệ bị bắt không biết thuộc kiểu nào, chúng ta có thể sử dụng lớp ‘Exception’ để bắt biệt lệ đó.
- Lỗi sẽ được truyền thông qua khối lệnh ‘try catch’ cho tới khi chúng bắt gặp một ‘catch’ tham chiếu tới nó, hoặc chương trình sẽ bị kết thúc

# Khối lệnh chứa nhiều Catch

- Các khối chứa nhiều ‘catch()’ xử lý các kiểu biệt lệ khác nhau một cách độc lập.
- Ví dụ

```
try
{   doFileProcessing();
    displayResults();     }
catch(LookupException e)
{   handleLookupException(e);     }
catch(Exception e)
{
System.err.println(“Error:”+e.printStackTrace
```

# Khối lệnh chứa nhiều Catch (tt)

- Khi sử dụng các ‘try’ lồng nhau, khối ‘try’ bên trong được thi hành đầu tiên
- Bất kỳ biệt lệ nào bị chặn trong khối lệnh ‘try’ sẽ bị bắt giữ trong khối lệnh ‘catch’ tiếp ngay sau.
- Nếu khối lệnh ‘catch’ thích hợp không được tìm thấy, thì các khối ‘catch’ của khối ‘try’ bên ngoài sẽ được xem xét
- Ngược lại, Java Runtime Environment sẽ xử lý biệt lệ.

# Khối ‘finally’

- Thực hiện tất cả các việc thu dọn khi biệt lệ xảy ra
- Có thể sử dụng kết hợp với khối ‘try’
- Chứa các câu lệnh thu hồi tài nguyên về cho hệ thống hay lệnh in ra các câu thông báo:
  - Đóng tập tin
  - Đóng lại bộ kết quả (được sử dụng trong chương trình cơ sở dữ liệu)
  - Đóng lại các kết nối được tạo trong cơ sở dữ liệu.

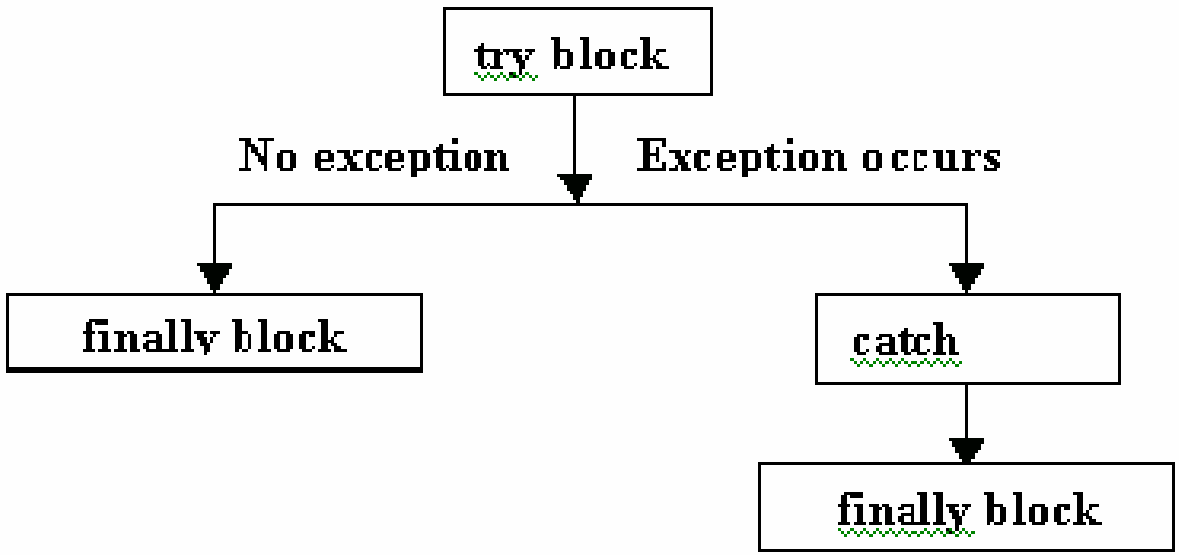
# Khối ‘finally’ (tt)

- Ví dụ

```
try
{
    doSomethingThatMightThrowAnException( );
}
finally
{
    cleanup( );
}
```

# Khối ‘finally’ (tt)

- Là tùy chọn không bắt buộc
- Được đặt sau khối ‘catch’
- Khối ‘finally’ bảo đảm lúc nào cũng được thực hiện bất chấp biệt lệ có xảy ra hay không.




Flow of the ‘try’, ‘catch’ and ‘finally’ blocks

# Các biệt lệ được định nghĩa với lệnh ‘throw’ và ‘throws’

- Các biệt lệ thì được chặn với sự trợ giúp của từ khóa ‘throw’
- Từ khóa ‘throw’ chỉ ra một biệt lệ vừa xảy ra.
- Toán hạng của throw là một đối tượng của một lớp, mà lớp này được dẫn xuất từ lớp ‘Throwable’
- Ví dụ của lệnh ‘throw’

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

# Các biệt lệ được định nghĩa với lệnh ‘throw’ và ‘throws’(tt)

- Một phương thức đơn có thể chặn nhiều hơn một biệt lệ
- Ví dụ từ khóa ‘throw’ xử lý nhiều biệt lệ

```
public class Example {
   public void exceptionExample( ) throws
ExException,    LookupException  {
        try
        {   // statements     }
        catch(ExException exmp)
        { .... }
        catch(LookupException lkpex)
        { .... }              }       }
```

# Các biệt lệ được định nghĩa với lệnh ‘throw’ và ‘throws’(tt)

- Lớp ‘Exception’ thực thi giao diện ‘Throwable’ và cung cấp các tính năng hữu dụng để phân phối cho các biệt lệ.
- Một lớp con của lớp Exception là một biệt lệ mới có thể bắt giữ độc lập các loại Throwable khác.

# Danh sách các biệt lệ

- RuntimeException
- ArithmeticException
- IllegalAccessException
- IllegalArgumentException
- ArrayIndexOutOfBoundsException
- NullPointerException
- SecurityException
- ClassNotFoundException

# Danh sách các biệt lệ (tt)

- NumberFormatException
- AWTException
- IOException
- FileNotFoundException
- EOFException
- NoSuchMethodException
- InterruptedException

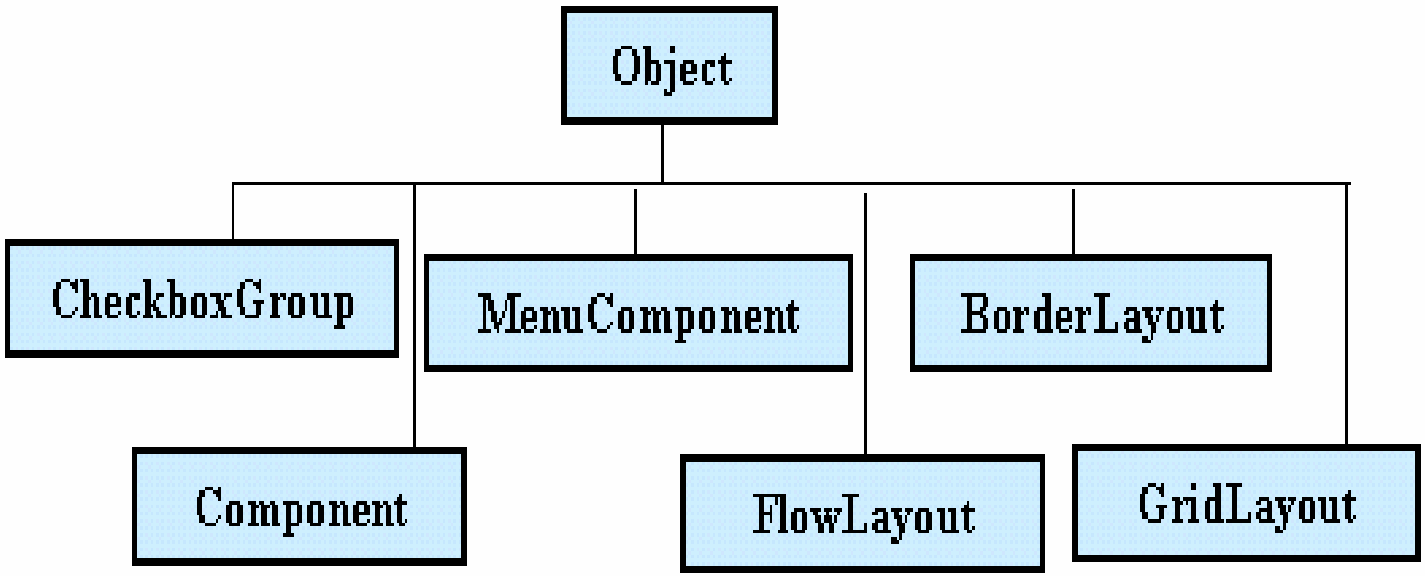
# Chương V

# LẬP TRÌNH GIAO DIỆN VỚI AWT

# GIỚI THIỆU VỀ AWT

- AWT viết tắt của **Abstract Windowing Toolkit**
- AWT là tập hợp các lớp Java cho phép chúng ta tạo một GUI
- Cung cấp các mục khác nhau để tạo hoạt động và hiệu ứng GUI như
  - Containers
  - Components
  - Layout managers
  - Graphics và drawing capabilities
  - Fonts
  - Events

- AWT bao gồm các lớp, interfaces và các gói




AWT class hierarchy

# Components

- Tất cả các thành phần cấu tạo nên chương trình GUI được gọi là component.
- **Ví dụ**
  - **Containers,**
  - textfields, labels, checkboxes, textareas
  - scrollbars, scrollpanes, dialog

# Containers

- Là thành phần mà có thể chứa các thành phần khác. có thể vẽ và tô màu.
- Có các frames, panes,latches, hooks
- Java.awt chứa một lớp có tên là Container. Lớp này dẫn xuất trực tiếp và không trực tiếp theo 2 cách là:
  - Frames
  - Panels

# Frames

- Là các cửa sổ
- Là lớp con của Windows
- Được hiển thị trong một cửa sổ và có đường viền

# Panels

- Là các vùng chứa trong một cửa sổ.
- Hiển thị trong một cửa sổ mà trình duyệt hoặc appletviewer cung cấp và không có đường viền.
- Được sử dụng để nhóm một số các thành phần
- Một panel không có sẳn vì thế chúng ta cần phải thêm nó vào frame.
- Hàm dựng
  - **Panel()**

# Dialog

- Là một lớp con của lớp Window
- Đối tượng dialog được cấu trúc như sau :

```
Frame myframe = new Frame("My frame");
String title = "Title";
boolean modal = true;
Dialog dlg = new Dialog( myframe, title, modal);
```

# Các Components khác

- **Ví dụ**
  - textfields, labels, checkboxes, textareas
  - scrollbars, scrollpanes, dialog

# Label

- Được dùng để hiển thị chuỗi (String)
- Các hàm dựng:
  - **Label( )**
  - **Label(String labeltext)**
  - **Label(String labeltext, int alignment)**
- Các phương thức:
  - **setFont(Font f)**
  - **setText(String s)**
  - **getText( )**

# TextField

- Là điều khiển text cho phép hiển thị text hoặc cho user nhập dữ liệu vào.
- Các hàm dựng:
  - **TextField( )**
  - **TextField(int columns)**
  - **TextField(String s)**
  - **TextField(String s, int columns)**
- Các phương thức:
  - **setEchoChar(char)**
  - **setTest(String s)**
  - **getText( )**
  - **setEditable(boolean)**
  - **isEditable( )**

# TextArea

- Được dùng khi text có nội dung từ hai dòng trở lên
- Là điều khiển text có thể soạn thảo được với nhiều dòng
- Các bước để tạo TextArea:
  - Tạo một phần tử (element)
  - Chỉ ra số dòng hay số cột (tùy chọn)
  - Chỉ ra vị trí của điều khiển trên màn hình

# TextArea (tt…)

- Các hàm dựng:
  - **TextArea( )**
  - **TextArea(int rows, int cols )**
  - **TextArea(String text)**
  - **TextArea(String text, int rows, int cols)**

# Các phương thức của TextArea

- **setText(String)**
- **getText( )**
- **setEditable(boolean)**
- **isEditable( )**
- **insertText(String, int)**
- **replaceText(String, int, int)**

# Button

- Các nút Push hay Command là cách dễ nhất để lấy các sư kiện của user
- Các bước để tạo button:
  - Tạo một phần tử button, nên tạo cho nó một caption để chỉ ra mục đích của nó
  - Chỉ ra vị trí đặt button trên màn hình
  - Hlển thị ra trên màn hình
- Các hàm dựng:
  - **Button( )**
  - **Button(String text)**

# Checkboxes and RadioButtons

- Checkboxes được dùng khi cho phép user nhiều cọn chọn lựa
- Radiobuttons được dùng để user chỉ ra một lựa chọn duy nhất
- Các bước để tạo checkbox hoặc radiobutton:
  - Tạo một phần tử (element)
  - Khởi tạo giá trị ban đầu (có giá trị selected hay unselected)
  - Chỉ ra vị trí trên màn hình
  - Hiển thị ra màn hình
- Các hàm dựng để tạo checkbox:
  - **Checkbox( )**
  - **Checkbox(String text)**
- Để tạo radiobutton, ta phải tạo đối tượng CheckBoxGroup trước khi tạo button

# Choice Lists

- Lớ p ‘Choice’ cho phép ta tạo danh sách có nhiều chọn lựa
- Khi list được tạo lần đầu tiên, nó được khởi tạo là empty
- Các bước để tạo danh sách chọn lựa:
  - Tạo một phần tử
  - Thêm các mục (có kiểu Strings) vào danh sách đó, từng mục một
  - Chỉ ra vị trí trên màn hình
  - Hiển thị ra màn hình
- Ví dụ

```
Choice colors=new Choice( );
colors.addItem(“Red”);
colors.addItem(“Green”);
```

# Trình quản lý bố trí Layout Manager

- Các loại layout khác nhau:
  - Flow Layout
  - Border Layout
  - Card Layout
  - Grid Layout
  - GridBag Layout
- Trình quản lý layout được thiết lập bằng cách gọi phương thức ‘setLayout( )’

# FlowLayout

- Là trình quản lý layout mặc định cho các applet và các panel
- Với FlowLayout các thành phần sẽ được xắp xếp từ góc trái trên đến góc phải dưới của màn hình
- Các constructor:

```
FlowLayout mylayout = new FlowLayout();
FlowLayout exLayout = new
 flowLayout(FlowLayout.RIGHT);
```

# BorderLayout

- Là trình quản lý layout mặc định cho Window, Frame và Dialog
- Trình quản lý này có thể xắp xếp đến 5 thành phần trong container
- Các thành phần có thể được đặt vào 5 hướng NORTH, EAST, SOUTH, WEST và CENTER của container
- **Ví dụ:** Để thêm một thành phần vào vùng North của container

```
Button b1= new Button("North Button");
setLayout(new BorderLayout( ));
add(b1, BorderLayout.NORTH);
```

# CardLayout

- Có thể lưu trữ một danh sách các kiểu layout khác nhau
- Mỗi layout được xem như một thẻ (card)
- Thẻ thường là đối tượng Panel
- Một thành phần độc lập như button sẽ điều khiển các thẻ  được đặt ở phía trên nhất
- Các bước để tạo CardLayout:
  - Bố trí layout của panel chính là CardLayout
  - Lần lượt thêm các panel khác vào panel chính

# GridLayout

- Hỗ trợ việc chia container thành một lưới
- Các thành phần được bố trí trong các dòng và cột
- Một ô lưới nên chứa ít nhất một thành phần
- Kiểu layout này được sử dụng khi tất cả các thành phần có cùng kích thước
- Hàm constructor

```
GridLayout gl = new GridLayout(no. of rows, no. of columns);
```

# GridBagLayout

- Bố trí các thành phần một cách chính xác
- Các thành phần không cần có cùng kích thước
- Các thành phần được xắp xếp trong một lưới chứa các dòng và các cột
- Thứ tự đặt các thành phần không tuân theo hướng từ trái-sang-phải và trên-xuống-dưới
- Hàm constructor
  **GridBagLayout gb = new GridBagLayout( );**

# GridBagLayout

- Để sử dụng layout này, bạn cần phải biết thông tin về kích cỡ và cách bố trí của các thành phần
- Lớp ‘GridBagLayoutConstraints’ lưu trữ tất cả các thông tin mà lớp GridLayout yêu cầu: Vị trí và kích thước mỗi thành phần

# Xử lý các sự kiện

- Các sự kiện (Events) được xử lý bằng các công cụ sau:
  - Abstract Windowing Toolkit
  - Trình duyệt.
  - Các trình xử lý sự kiện do các lập trình viên tạo riêng.
- Các ứng dụng cần đăng ký trình xử lý sự kiện với đối đối tượng
- Các trình xử lý này được gọi khi có một sự kiện tương ứng xảy ra

# Xử lý các sự kiện (tt…)

- Event Listener sẽ lắng nghe một sự kiện cụ thể mà một đối tượng tạo ra
- Mỗi event listener cung cấp các phương thức để xử lý các sự kiện này
- Lớp có cài đặt listener cần định nghĩa những phương thức này

# Xử lý các sự kiện(tt…)

- Các bước cần tuân thủ để sử dụng mô hình Event Listener:
  - Cài đặt Listener tương ứng
  - Nhận diện được tất cả các thành phần tạo ra sự kiện
  - Nhận diện được tất cả các sự kiện được xử lý
  - Cài đặt các phương thức của listener, và viết các đoạn mã để xử lý sự kiện trong các phương thức đó
- Interface định nghĩa các phương thức khác nhau để xử lý mỗi sự kiện

# Các sự kiện và Listener tương ứng

- ActionEvent
- AdjustmentEvent
- ComponentEvent
- FocusEvent
- ItemEvent
- WindowEvent
- TextEvent
- MouseEvent
- KeyEvent

- ActionListener
- AdjustmentListener
- ComponentListener
- FocusListener
- ItemListener
- WindowListener
- TextListener
- MouseListener
- MouseMotionListener
- KeyListener

# Menus

- Các loại menu :
  - Pull-down
  - Pop-up menu
- Chỉ có thể đặt các thanh menubar vào trong các Frame mà thôi
- Các thành phần của menu:
  - Menubar
  - MenuItems

# Chương VI

# Applets

# Applets

- Là một chương trình Java mà chạy với sự hỗ trợ của trình duyệt web
- Tất cả các applets là lớp con của lớp ‘Applet’
- Để tạo một applet, bạn cần import hai gói sau:
  - **java.applet**
  - **java.awt**

# Cấu trúc applet

- Định nghĩa một applet từ bốn sự kiện xảy ra trong quá trình thực thi
- Đối với mỗi sự kiện được định nghĩa bởi một phương thức tương ứng.
- Các phương thức:
  - **init( )**
  - **start( )**
  - **stop( )**
  - **destroy( )**

- Các phương thức khác:
  - **paint( )**
  - **repaint( )**
  - **showStatus( )**
  - **getAppletInfo( )**
- Các phương thức init(), start(), stop(), destroy(), and paint() được thừa kế từ applet.
- Mỗi phương thức này mặc định là rỗng. Vì thế các phương thức này phải được nạp chồng.

# Biên dịch và thực thi applet

- Một applet thì được biên dịch theo cú pháp sau

  **javac Applet1.java**

- Để thực thi một applet, tạo một tập tin HTML có sử dụng thẻ applet
  - Thẻ applet có hai thuộc tính:
    - Width
    - Height
  - Để truyền tham số tới applet, sử dụng thẻ ‘param’, và tiếp theo là thẻ ‘value’
- Applet có thể được thực thi bằng applet viewer

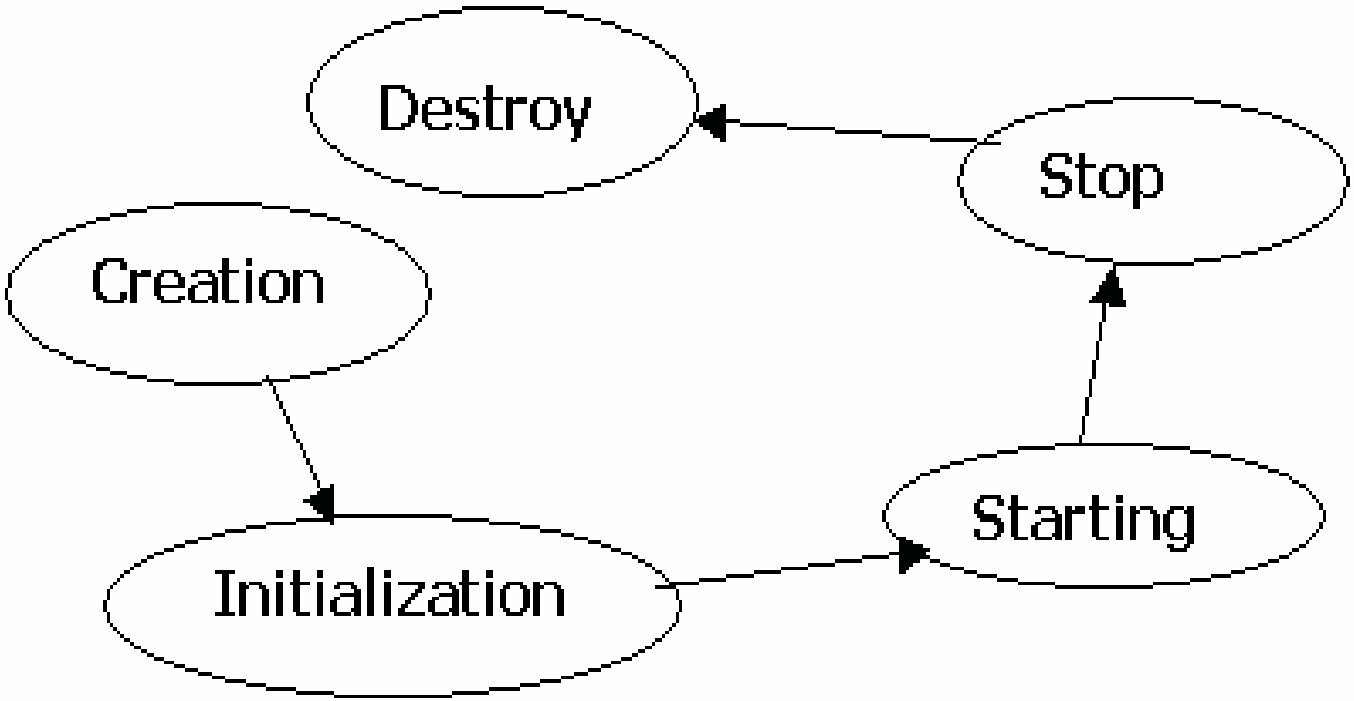
# Điểm khác biệt giữa applet và một ứng dụng

- Các ứng dụng khi thực thi phải sử dụng trình biên dịch Java, trong khi các applets thực thi được trên bất kỳ trình duyệt nào mà hổ trợ Java, hoặc sử dụng ‘AppletViewer’ trong JDK.
- Một ứng dụng bắt đầu với phương thức ‘main()’. Còn đối với applet thì không sử dụng phương thức này
- Một ứng dụng sử dụng ‘System.out.println()’ để hiển thị, trong khi môt applet thì sử dụng phương thức ‘drawstring()’ để hiển thị.

# Những hạn chế về bảo mật trong applet

- Không thể đọc hoặc viết các tập tin trên hệ thống tập tin của người sử dụng
- Không thể giao tiếp với một site trên internet. Mà chỉ giao tiếp với một dịch vụ trên trang web có applet.
- Không thể chạy bất kỳ chương trình nào trên hệ thống của người đọc
- Không thể load bất kỳ chương trình nào được lưu trên hệ thống của người sử dụng

# Chu trình sống của applet




Life cycle of an applet

# Truyền tham số tới một applet

- Để truyền tham số, sử dụng PARAM trong thẻ HTML
- Ví dụ

```
<applet code = "Mybutton1" width = “100” height = “100”>
<PARAM NAME = “Mybutton” value = “Display Dialog”>
</applet>
```

# Lớp đồ họa

- Được cung cấp bởi gói AWT
- Cung cấp một tập hợp các phương thức để vẽ như sau:
  - Oval
  - Rectangle
  - Square
  - Circle
  - Lines
  - Text in different fonts

# Graphical Background

- Các phương thức để vẽ nền :
  - **getGraphics( )**
  - **repaint( )**
  - **update(Graphics g)**
  - **paint(Graphics g)**

# Hiển thị chuỗi, ký tự và bytes

- Phương thức để vẽ hoặc hiển thị một chuỗi trên frame

  **Cú pháp**

  – **drawString(String str, int xCoor, int yCoor);**

- Phương thức để vẽ hoặc hiển thị các ký tự trên frame

  **Cú pháp**

  – **drawChars(char array[ ], int offset, int length, int xCoor, int yCoor);**

- Phương thức để vẽ hoặc hiển thị bytes trên frame

  **Cú pháp**

  – **drawBytes(byte array[ ], int offset, int length, int xCoor, int yCoor);**

# Vẽ các hình thể

- Phương thức được sử dụng để vẽ đường thẳng như sau

  **Cú pháp**

  – **drawLine(int x1, int y1, int x2, int y2);**

- Các phương thức được sử dụng để vẽ đường tròn như sau

  **Cú pháp**

  – **drawOval(int xCoor, int yCoor, int width, int height);**

  – **setColor(Color c);**

  – **fillOval(int xCoor, int yCoor, int width, int height);**

- Phương thức sử dụng để vẽ hình vuông:

  **Cú pháp**

  – **drawRect(int xCoor, int yCoor, int width, int height);**

  – **fillRect(int xCoor, int yCoor, int width, int height);**

- Các phương thức được sử dụng để vẽ hình vuông có góc tròn

  Cú pháp

  – **drawRoundRect(int xCoor, int yCoor, int width, int height, int arcWidth, int arcHeight);**

  – **fillRoundRect (int xCoor, int yCoor, int width, int height, int arcWidth, int arcHeight);**

# 3D Rectangles & Arcs

- Các phương thức được sử dụng để vẽ hình 3D **Cú pháp**
  - **draw3DRect(int xCoord, int yCoord, int width, int height, boolean raised);**
  - **drawArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);**
  - **fillArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);**

# Drawing PolyLines

- Các phương thức được sử dụng để vẽ nhiều được thẳng

**Cú pháp**

- **drawPolyline(int xArray[ ], int yArray[ ], int totalPoints);**
- **g.setFont(new Font("Times Roman", Font.BOLD,15));**

# Vẽ và tô các hình đa giác

- Các phương thức để vẽ và tô các hình đa giác

  **Cú pháp**

  - **drawPolygon(int x[ ], int y[ ], int numPoints);**
  - **fillPolygon(int x[ ], int y[ ], int numPoints);**

# Màu

- Java sử dụng màu RGB
- Bảng các giá trị màu

| Element | Range |
|---|---|
| Red | 0-255 |
| Green | 0-255 |
| Blue | 0-255 |

- Cú pháp của hàm dựng để tạo một màu

**color(int red, int green, int blue);**

- Bảng trình bày các giá trị màu RGB thôn thường

| Color | Red | Green | Blue |
|---|---|---|---|
| White | 255 | 255 | 255 |
| Light Gray | 192 | 192 | 192 |
| Gray | 128 | 128 | 128 |
| Dark Gray | 64 | 64 | 64 |
| Black | 0 | 0 | 0 |
| Pink | 255 | 175 | 175 |
| Orange | 255 | 200 | 0 |
| Yellow | 255 | 255 | 0 |
| Magenta | 255 | 0 | 255 |

# Font

- Gói java.awt package cung cấp bởi lớp ‘Font’
- Các phương thức của lớp Font:
  - **getAllFont( )**
  - **getLocalGraphicsEnvironment( )**
  - **getFont( )**
  - **getFontList( )**

- Hàm dựng Font nhận 3 tham số
  - Tên font trong chuổi định dạng; tên này có trong phương thức getFontList( ).
  - Kiểu của font. Ví dụ như: Font.BOLD, Font.PLAIN, Font.ITALIC
  - Kích thước của font.
- Ví dụ

```
Font f1 = new Font("SansSerif", Font.ITALIC, 16);
g.setFont(f1);
```

# Lớp FontMetrics

- Đo lường các ký tự khác nhau hiển thị trong các font khác nhau.
- Việc đo lường bao gồm ‘height’, ‘baseline’, ‘ascent’, ‘descent’ và ‘leading’ của font.
- Nó không cụ thể vì nó là một lớp trừu tượng

# Lớp FontMetrics (tiếp theo…)

- Phương thức:
  - **getFontMetrics(f1)**
  - **getHeight( )**
  - **getAscent( )**
  - **getDescent( )**
  - **getLeading( )**
  - **getName( )**

# Kiểu vẽ

- Các đối tượng để vẽ được sử dụng.
- Method used to make old and new contents visible on the screen

  **setXORMode(Color c)**
- Method used to revert to the overwrite mode

  **setPaintMode( )**

Chương VII

# Lập trình đa tuyến

# Tuyến

- Lập trình đa tuyến là một đặc trưng của Java
- Tuyến là đơn vị nhỏ nhất của đoạn mã có thể thi hành được mà thực hiện một công việc riêng biệt

# Đa tuyến

- Là khả năng làm việc với nhiều tuyến
- Đa tuyến chuyên sử dụng cho việc thực thi nhiều công việc đồngthời
- Đa tuyến giảm thời gian rỗi của hệ thống đến mức thấp nhất.

# Tạo và quản lý tuyến (1)

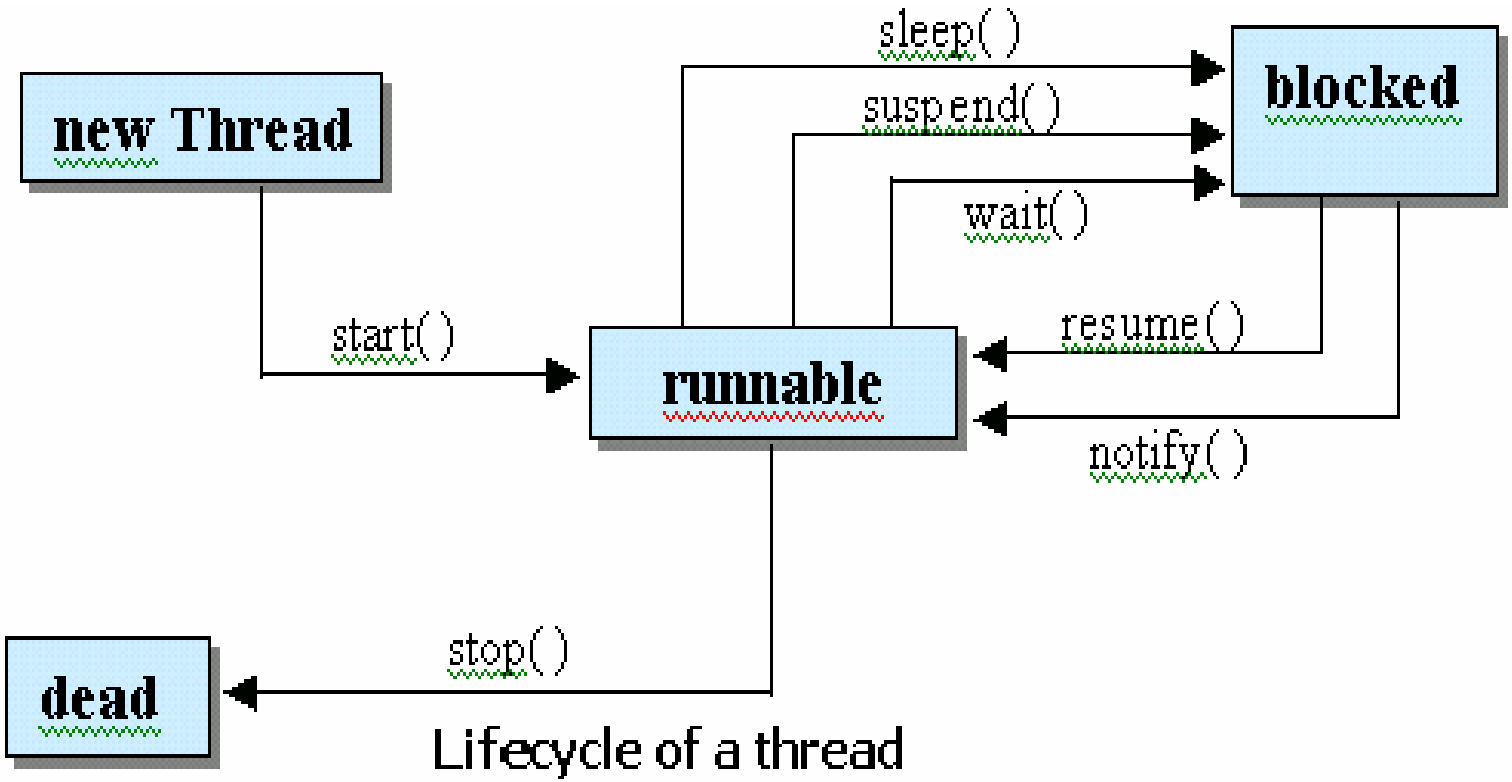
- Khi chương trình Java thực thi hàm main() tức là tuyến main được thực thi. Tuyến này được tạo ra một cách tự động. tại đây :

  - Các tuyến con sẽ được tạo ra từ đó

  - Nó là tuyến cuối cùng kết thúc việc thực hiện. Trong chốc lát tuyến chính ngừng thực thi, chương trình bị chấm dứt

- Tuyến có thể được tạo ra bằng 2 cách:
  - Dẫn xuất từ lớp Thread
  - Dẫn xuất từ Runnable.

# Vòng đời của một tuyến



# Trạng thái của tuyến và các phương thức của lớp tuyến

- **trạng thái:**
  - **born**
  - **ready to run**
  - **running**
  - **sleeping**
  - **waiting**
  - **ready**
  - **blocked**
  - **dead**
- **Phương thức:**
  - **start( )**
  - **sleep( )**
  - **wait( )**
  - **notify( )**
  - **run( )**
  - **stop( )**

# Các phương thức Khác

- **enumerate(Thread t)**
- **getName( )**
- **isAlive( )**
- **getPriority( )**
- **setName(String name)**
- **join( )**
- **isDaemon( )**
- **setDaemon(Boolean on)**
- **resume( )**
- **sleep( )**
- **start( )**

# Phân chia thời gian giữa các tuyến

- CPU chỉ thực thi chỉ một tuyến tại một thời điểm nhất định.
- Các tuyến có độ ưu tiên bằng nhau thì được phân chia thởi gian sử dụng bộ vi xử lý.

# tuyến Daemon(ngầm)

- Hai kiểu tuyến trong một chương trình Java:
  - Các tuyến người sử dụng
  - tuyến ngầm
- tuyến ngầm dọn rác

# Đa tuyến với Applets

- Các chương trình Java dựa trên Applet thường sử dụng nhiều hơn một tuyến
- Trong đa tuyến với Applets, Lớp ‘java.applet.Applet’ là lớp con được tạo ra một Applet người sử dụng đã định nghĩa
- nó không thể thực hiện được trực tiếp lớp con của lớp tuyến trong các applet
- Con đường để lớp con sử dụng lớp tuyến:
  - Sử dụng một đối tượng của tuyến người sử dụng định nghĩa, mà, lần lượt, dẫn xuất lớp tuyến
  - Thực thi chạy giao tiếp (interface)

# Sự đồng bộ tuyến

- Thâm nhập các tài nguyên/dữ liệu bởi nhiều tuyến
- Sự đồng bộ (Synchronization)
- Sự quan sát (Monitor)
- Mutex

# Mã đồng bộ

- Để thâm nhập sự quan sát của một đối tượng, lập trình viên sử dụng từ khóa ‘synchronized’ để gọi một phương thức hiệu chỉnh (modified method)
- Khi một tuyến đang được thực thi trong phạm vi một phương thức đồng bộ (synchronized), bất kỳ tuyến khác hoặc phương thức đồng bộ khác mà cố gắng gọi nó trong thời gian đó sẽ phải đợi

# Khuyết điểm của các phương thức đồng bộ

- Các trạng thái chắc chắn không lợi ích cho đa tuyến
- Trình biên dịch Java từ Sun không chứa nhiều phương thức đồng bộ
- Các phương thức đồng bộ chậm hơn từ ba đến bốn lần so với các phương thức tương ứng không đồng bộ.

# Kỹ thuật “wait-notify” (đợi – thông báo) (1)

- tuyến chia các tác vụ thành các đơn vị riêng biệt và logic (hợp lý)
- Để tránh trường hợp kiểm soát vòng, Java bao gồm một thiết kế tốt trong tiến trình kỹ thuật truyền thông sử dụng các phương thức “wait()” (đợi), “notify()” (thông báo) và “notifyAll()” (thông báo hết) :
  - **wait( )**
  - **notify( )**
  - **notifyAll( )**

# Kỹ thuật “wait-notify” (đợi – thông báo) (1)

- Các chức năng của các phương thức “wait()”, “notify()”, và “notifyAll()” là :
  - **wait( )**
  - **notify( )**
  - **notifyAll( )**
- tuyến ưu tiên cao nhất chạy đầu tiên
- Cú pháp của các phương thức:
  - **final void wait( ) throws IOException**
  - **final void notify( )**
  - **final void notifyAll( )**

# Một số điểm cần nhớ trong khi sử dụng phương thức wait():

- tuyến đang gọi đưa vào CPU
- tuyến đang gọi đưa vào khóa
- tuyến đang gọi đi vào vùng đợi của monitor

# Các điểm chính cần nhớ về phương thức notify()

- Một tuyến đưa ra ngoài vùng đợi của monitor, và vào trạng thái sẵn sàng
- tuyến mà đã được thông báo phải thu trở lại khóa của monitor trước khi nó có thể bắt đầu
- Phương thức notify() là không chính xác
- Trong một số trường hợp này, các phương thức của monitor đưa ra 2 sự đề phòng:
  - Trạng thái của monitor sẽ được kiểm tra trong một vòng lặp “while” tốt hơn là câu lệnh if
  - Sau khi thay đổi trạng thái của monitor, phương thức notifyAll() sẽ được sử dụng, tốt hơn phương thức notify().

# Sự bế tắt (Deadlocks)

- Một “deadlock” (sự bế tắt) xảy ra khi hai tuyến có một phụ thuộc vòng quanh trên một cặp đối tượng đồng bộ
- Nó khó để gỡ lỗi một bế tắt bởi những nguyên nhân sau:
  - Nó hiểm khi xảy ra, khi hai tuyến chia nhỏ thời gian trong cùng một con đường
  - Nó có thể bao hàm nhiều hơn hai tuyến và hai đối tượng đồng bộ
- Nếu một chương trình đa tuyến khóa kín thường xuyên, ngay lập tức kiểm tra lại điều kiện bế tắt

# Thu dọn “rác” (Garbage collection)

- Cải tạo hoặc làm trống bộ nhớ đã định vị cho các đối tượng mà các đối tượng này không sử dụng trong thời gian dài
- Sự dọn rác thực thi như là một tuyến riêng biệt có quyền ưu tiên thấp
- Sử dụng câu lệnh sau để tắt đi sự dọn rác trong ứng dụng:

  **java –noasyncgc…**

# Phương thức finalize() (hoàn thành)

- Java cung cấp một con đường để làm sạch một tiến trình trước khi điều khiển trở lại hệ điều hành
- Phương thức finalize(), nếu hiện diện, sẽ được thực thi trên mỗi đối tượng, trước khi sự dọn rác
- Câu lệnh của phương thức finalize() như sau :
  - **protected void finalize( ) throws Throwable**
- Tham chiếu không phải là sự dọn rác; chỉ các đối tượng mới được dọn rác

# Chương VIII

# Các luồng I/O

# Các luồng

- Các luồng là những đường ống dẫn để gửi và nhận thông tin trong các chương trình java.
- Khi một luồng đọc hoặc ghi , các luồng khác bị khoá.
- Nếu lỗi xẩy ra trong khi đọc hoặc ghi luồng, một ngoại lệ sẽ kích hoạt.
- Lớp ‘java.lang.System’ định nghĩa luồng nhập và xuất chuẩn.

# Các lớp luồng I/O

- Lớp System.out.
- Lớp System.in.
- Lớp System.err.

# Lớp InputStream

- Là lớp trừu tượng
- Định nghĩa cách nhận dữ liêu
- Cung cấp số phương thức dùng để đọc và các luồng dữ liệu làm đầu vào.
- Các phương thức:
  - **read( )**
  - **available( )**
  - **close ( )**
  - **mark ( )**
  - **markSupported( )**
  - **reset( )**
  - **skip( )**

# Lớp OutputStream

- Là lớp trừu tượng.
- Định nghĩa cách ghi dữ liệu vào luồng.
- Cung cấp tập các phương thức trợ giúp. trong việc tạo, ghi và xử lý các luồng xuất.
- Các phương thức:
  - **write(int)**
  - **write(byte[ ])**
  - **write(byte[ ], int, int)**
  - **flush( )**
  - **close( )**

# Nhập mảng các Byte

- Sử dụng các đệm bộ nhớ
- Lớp **ByteArrayInputStream**
- Tạo ra một luồng nhập từ đệm bộ nhớ không gì cả về mảng các byte.
  - Không hỗ trợ các phương thức mới
  - Các phương thức nộp chồng của lớp InputStream, giống như ‘read()’, ‘skip()’, ‘available()’ và ‘reset()’.

# Byte Array Output

- sử dụng các vùng đệm bộ nhớ
- Lớp **ByteArrayOutputStream**
  - Tạo ra một luồng kết xuất trên mảng byte
  - Cung cấp các khả năng bổ sung cho mảng kết xuất tăng trưởng nhằm chừa chổ cho dữ liệu mới ghi vào.
  - Cũng cung cấp các phương thức để chuyển đổi luồng tới mảng byte, hay đối tượng String.

- Phương thức của lớp **ByteArrayOutputStream** :
  - **reset( )**
  - **size( )**
  - **writeTo( )**

# Các lớp nhập/xuất tập tin

- Các lớp này trợ giúp trong Java để hổ trợ các thao tác nhập và xuất:
  - File
  - FileDescriptor
  - FileInputStream
  - FileOutputStream
- Các lớp File, FileDescriptor, và RandomAccessFile được sử dụng hỗ trợ trực tiếp hoặc truy cập nhập/xuất ngẫu nhiên.

# Lớp tập tin

- Được sử dụng truy cập các đối tượng tập tin và thw mục
- Những tập tin có tên được đặt tên theo qui ước của hệ điều hành chủ
- Lớp này cung cấp phương thức khởi tạo để tạo ra các thư mục và tập tin
- Tất cả các thao tác thư mục và tập tin đều được sử dụng các phương thức truy cập và các phương thức thư mục mà các lớp tập tin cung cấp

# Lớp FileDescriptor

- Cung cấp việc truy cập tới các tập tin mô tả
- Không cung cấp bất kỳ tính rõ nét nào tới thông tin mà hệ điều hành duy trì.
- Cung cấp chỉ một phương thức gọi là ‘valid( )’

# Lớp FileInputStream

- Cho phép đầu vào đọc từ một tập tin trong một mẫu của một dòng
- Các đối tượng được tạo ra sử dụng chuỗi tên tập tin, tập tin, đối tượng FileDescriptor như một tham số.
- Các phương thức nạp chồng của lớp InputStream. nó cung cấp phương thức ‘finalize( )’ và ‘getFD( )’

# Lớp FileOutputStream

- Cho phép kết xuất để ghi ra một luồng tập tin
- Các đối tượng cũng tạo ra sử dụng một chuỗi tên tập tin, tạp tin, hay đối tượng FileDescriptor như một tham số.
- Lớp này nạp chồng các phương thức của lớp OutputStream và cung cấp phương thức ‘finalize( )’ và ‘getFD( )’

# Nhập xuất lọc

- Lọc:
  - Là kiểu luồng sửa đổi cách điều quản một luồng hiện có.
  - về cơ bản được sử dụng để thích ứng các luồng theo các nhu cầu của chương trình cụ thể.
  - Bộ lọc nằm giữa luồng nhập và luồng xuất.
  - Thực hiện một số tiến trình đặt biệt trên các byte được chuyển giao từ đầu vào đến kết xuất.
  - Có thể phối hợp để thực hiện một dãy các tuỳ chọn lọc.

# Lớp FilterInputStream

- Là lớp trừu tượng.
- Là cha của tất cả các lớp luồng nhập đã lọc.
- Cung cấp khả năng tạo ra một luồng từ luồng khác.
- Một luồng có thể đọc và cung cấp cung cấp dưới dạng kết xuất cho luồng khác.
- duy trì một dãy các đối tượng của lớp ‘InputStream’
- Cho phép tạo ra nhiều bộ lọc kết xích (chained filters
- ).

# Lớp FilterOutputStream

- Là dạng bổ trợ cho lớp ‘FilterInputStream’.
- Là cha của tất cả các lớp luồng kết xuất.
- Duy trì đối tượng của lớp ‘OutputStream’ như là một biến ‘out’.
- Dữ liệu ghi ra lớp này có thể sửa đổi để thực hiện các thao tác lọc, và sau đó phản hồi đến đối tượng ‘OutputStream’.

# Vùng đệm nhập/xuất

- Vùng đệm:
  - Là kho lưu trữ dữ liệu.
  - Có thể cung cấp dữ liệu thay vì quay trợ lại nguồn dữ liệu gốc ban đầu.
  - Java sử dụng vùng đệm nhập và kết xuất để tạm thời lập cache dữ liệu được đọc hoặc ghi vào một luồng.
- Trong khi thực hiện vùng đệm nhập:
  - Số lượng byte lớn được đọc cùng thời điểm, và lưu trữ trong  một vùng đệm nhập.
  - Khi chương trình đọc luồng nhập, các byte nhập được đọc vào vùng đệm nhập.

# Vùng đệm nhập/xuất (tt…)

- Trong trường hợp vùng đệm kết xuất, một chương trình ghi ra một luồng.
- Dữ liệu kết xuất được lưu trữ trong một vùng đệm kết xuất.
- Dữ liệu được lưu trữ cho đến khi vùng đệm trợ nên đầy, hay luồng kết xuất được xả trống.
- Kết thúc, vùng đệm kết xuất được chuyển gửi đến đích của luồng xuất.

# Lớp BufferedInputStream

- Tự động tạo ra và duy trì vùng đệm để hổ trợ vùng đệm nhập.
- bởi lớp ‘BufferedInputStream’ là một bộ đệm, nó có thể áp đụng cho một số các đối tượng nhất định của lớp ‘InputStream’.
- Cũng có thể phối hợp các tập tin đầu vào khác.
- Sử dụng vài biến để triển khai vùng đệm nhập.

# Lớp BufferedInputStream (Contd…)

- Định nghĩa hai phương thức thiết lập:
  - Một chó phép chỉ định kích thước của vùng đệm nhấp.
  - phương thức kia thì không.
- Cả hai phương thức thiết lập đều tiếp nhận một đối tượng của lớp ‘InputStream’ như một tham số.
- Nạp chồng các phương thức truy cập mà InputStream cung cấp, và không đưa vào bất kỳ phương thức mới nào.

# Lớp BufferedOutputStream

- Thực hiện vùng đệm kết xuất theo cách tương ứng với lớp ‘BufferedInputStream’.
- Định nghĩa hai phương thức thiết lập. Nó cho phép chúng ta ấn định kích thước của vùng đệm xuất trong một phương thức thiết lập, cũng giống như cung cấp kích thước vùng đệm mặc định.
- Nạp chồng tất cả phương thức của lớp ‘OutputStream’ và không đưa vào bất kỳ phương thức nào.

# Lớp Reader và Writer

- Là các lớp trừu tượng.
- Chúng nằm tại đỉnh của hệ phân cấp lớp, hỗ trợ việc đọc và ghi các luồng ký tự unicode.

# Lớp Reader

- Hỗ trợ các phương thức sau:
  - **read( )**
  - **reset( )**
  - **skip( )**
  - **mark( )**
  - **markSupported( )**
  - **close( )**
  - **ready( )**

# Lớp Writer

- Hỗ trợ các phương thức sau :
  - **write( )**
  - **flush( )**
  - **close( )**

# Nhập/xuất chuỗi và mảng ký tự

- Hỗ trợ nhập và xuất từ các vùng đệm bộ nhớ
- Hỗ trợ 8 bít ký tự nhập và kết xuất

- Lớp ‘CharArrayReader’ không bổ sung phương thức mới vào các phương thức mà lớp ‘Reader’ cung cấp.

# Nhập/xuất chuỗi và mảng ký tự (tt)

- Lớp ‘CharArrayWriter’ bổ sung phương thức sau đây vào phương thức của lớp ‘Writer’ cung cấp:
  - **reset( )**
  - **size( )**
  - **toCharArray( )**
  - **toString( )**
  - **writeTo( )**

# Nhập/xuất chuỗi và mảng ký tự (tt)

- Lớp ‘StringReader’ trợ giúp đọc các ký tự đầu vào từ sâu chuỗi.
- Nó không bổ sung bất kỳ phương thức nào mà lớp  Reader cung cấp.
- Lớp ‘StringWriter’ trợ giúp để ghi luồng kết xuất ký tự ra một đối tượng ‘StringBuffer’.
- Lớp này bổ sung thêm các phương thức sau:
  - **getBuffer( )**
  - **toString( )**

# Lớp PrinterWriter

- Thực hiện một kết xuất.
- Lớp này có phương thức bổ sung , trợ giúp in các kiểu dữ liệu cơ bản .
- Lớp PrintWriter thay thế lớp ‘PrintStream’
- Thực tế cải thiện lớp ‘PrintStream’; lớp này dùng một dấu tách dòng phụ thuộc nền tảng điểm các dòng thay vì ký tự ‘\n’.
- Cung cấp phần hỗ trợ  cho các ký tự unicode so với PrintStream.
- Các phương thức:
  - **checkError( )**

# Giao diện DataInput

- Được sử dụng để đọc các byte từ luồng nhị phân, và
- Is used to read bytes from a binary stream, and xây dựng lại dữ liệu trong một số kiểu dữ liệu nguyên thuỷ.
- Cho phép chúng ta chuyển đổi dữ liệu từ từ khuôn dạng UTF-8 được sửa đổi Java đến dạng chuỗi
- Định nghiã số phương thức, bao gồm các phương thức để đọc các kiểu dữ liệu nguyên thuỷ.

# Những phương thức giao diện DataInput

- **boolean readBoolean( )**
- **byte readByte( )**
- **char readChar( )**
- **short readShort( )**
- **long readLong( )**
- **float readFloat( )**
- **int readInt( )**
- **double readDouble**
- **String readUTF( )**
- **String readLine( )**

# Giao diện DataOutput

- Được sử dụng để xây dựng lại dữ liệu một số kiểu dữ liệu nguyên thuỷ vào trong dãy các byte
- Ghi các byte dữ liệu vào luồng nhị phân
- Cho phép chúng ta chuyển đổi một chuỗi vào khuôn dạng UTF-8 được sửa đổi Java và viết nó vào trong một dãy.
- Định nghĩa một số phương thức và tất cả phương thức kích hoạt IOException trong trường hợp lỗi.

# Các phương thức giao diện DataOutput

- **void writeBoolean(boolean b)**
- **void writeByte( int value)**
- **void writeChar(int value)**
- **void writeShort(int value)**
- **void writeLong(long value)**
- **void writeFloat(float value)**
- **void writeInt(int value)**
- **void writeDouble(double value)**
- **void writeUTF(String value)**

# Lớp RandomAccessFile

- Cung cấp khả năng thực hiện I/O theo các vị trí cụ thể bên trong một tập tin.
- dữ liệu có thể đọc hoặc ghi ngẫu nhiên ở những vị trí bên trong tập tin thay vi một kho lưu trữ thông tin liên tục.
- phương thức ‘seek( )’ hỗ trợ truy cập ngẫu nhiên.
- Thực hiện cả đầu vào và đầu ra dữ liệu.
- Hỗ trợ các cấp phép đọc và ghi tập tin cơ bản.
- Kế thừa các phương thức từ các lớp ‘DataInput’ và ‘DataOutput’

# Các phương thức của lớp RandomAccessFile

- **seek( )**
- **getFilePointer( )**
- **length( )**

# Gói java.awt.print

- Gồm có các giao diện
  - Pageable:
    - Định nghĩa các phương thức dùng để các đối tượng biểu thị các trang sẽ được in.
    - Chỉ định số trang đã được in, và trang hiện tại hay là tranh giới trang đã được in
  - Printable:
    - Chi định phương thức ‘print( )’ sử dụng để in một trang trên đối tượng ‘Graphics’
  - PrinterGraphics:
    - Cung cáp khả năng truy cập đối tượng ‘PrinterJob’

- Giao diê n ‘PrinterGraphics’ cung cấp các lớp sau:
  - Paper
  - Book
  - PageFormat
  - PrinterJob
- Gói ‘java.awt.print’ kích hoạt các ngoại lệ:
  - PrinterException
  - PrinterIOException
  - PrinterAbortException

# Chương IX

KẾT NỐI CSDL

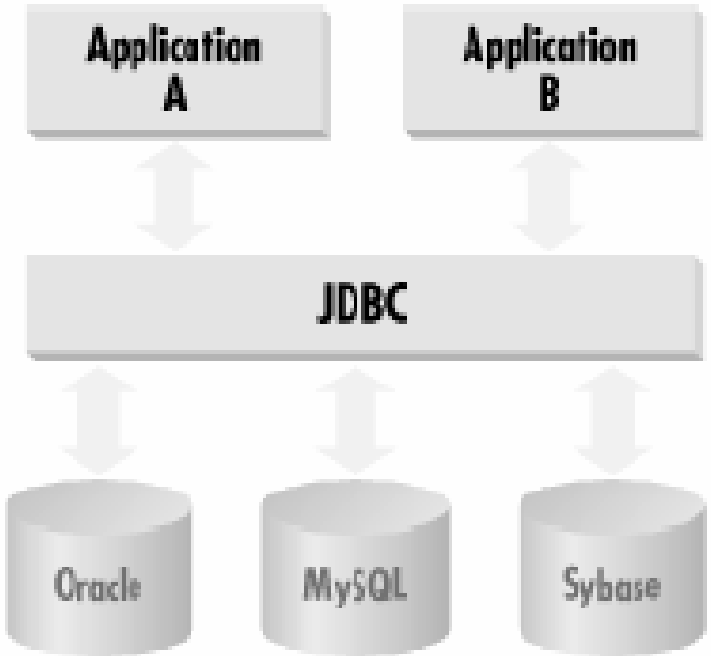
Java Database Connectivity

# Tổng quan

- JDBC cung cấp tập các lớp và interface cho phép chương trình Java có thể nói chuyện được với hệ CSD
- Tập các lớp của JDBC

có thể làm việc được với

mọi hệ csdl.



- Có 3 bước chính để kết nối CSDL.
  - Nạp database drivers
  - Tạo nối kết, Tạo đối tượng Connection
  - Tạo đối tượng Statement để thực thi các lệnh sql..

# Ví dụ

```
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    Connection con=DriverManager.getConnection("jdbc:odbc:ATM");

     Statement stmt = con.createStatement();
    :
    :
    :
    :
```

# Database URL

- Database URL là một chuổi được dùng để kết nối csdl.
- cú pháp :
- jdbc:subprotocol name:other_stuff
- The subprotocol name được dùng tuỳ vào loại driver sử dụng để kết nối csdl.
- ví dụ : subprotocol name là odbc nếu driver là cầu nối jdbcodbc
- Other_stuff cũng phụ thuộc vào loại driver nào được sử dụng. ví dụ nếu driver là cầu nối jdbcodbc thì thành phần này là tên của đối tượng ODBC

# Database Driver

- Bảo đảm ứng dụng java tương tác với mọi csdl dưới một cách thức chuẩn và duy nhất.
- Bảo đảm những yêu cầu từ chương trình sẽ được biểu diễn trong csdl dưới một ngôn ngữ mà csdl hiểu được
- nhận các yêu cầu từ client, chuyển nó nó vào định dạng mà csdl có thể hiểu được và thể hiện trong csdl.
- Nhận các phản hồi, chuyển nó ngược lại định dạng dữ liệu java và thể hiện trong ứng dụng.

# Nạp Driver

- Lớp DriverManager chịu trách nhiệm nạp driver và tạo kết nối đến csdl.

**DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());**

- hoặc

***Class.forName(String);***

  - This returns the object associated with the class with the given string name.
  - Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

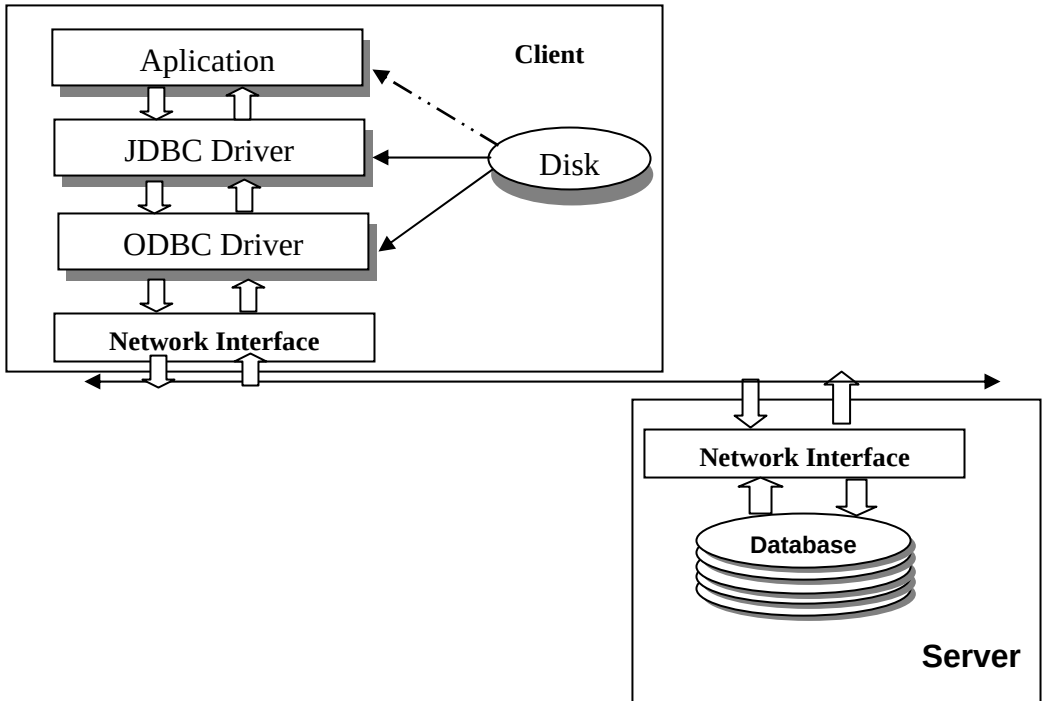
- Equivalent to:
  **new sun.jdbc.odbc.JdbcOdbcDriver();**
  If you have a driver from another vendor, then find out the class name of that driver and load it instead.

# JDBC Driver

- Có 4 loại JDBC Driver
  - Loại 1: JDBC/ODBC
  - Loại 2: Native-API
  - Loại 3: Open Protocol-Net
  - Loại 4: Proprietary-Protocol-Net
- Loại 2,3,4 nói chung được viết bởi nhà cung cấp csdl. hiệu quả hơn loại 1 nhưng thực hiện phức tạp hơn.

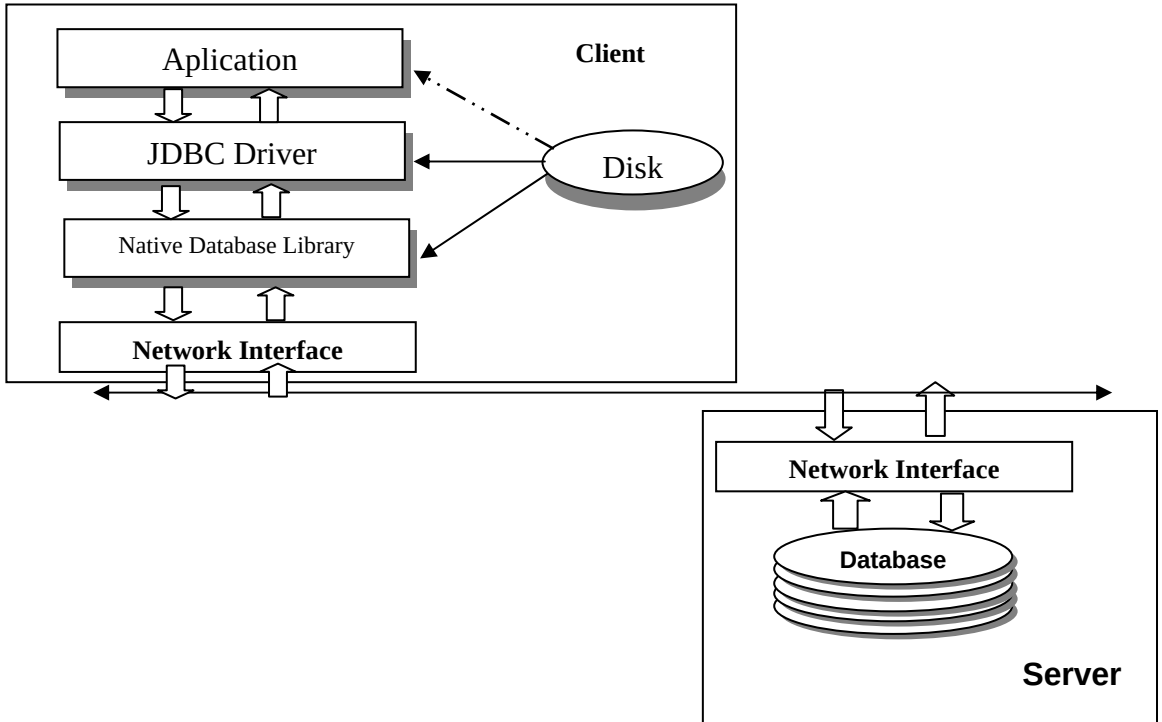
# Loại I JDBC/ODBC

- jdk hỗ trợ cầu nối jdbc-odbc (jdbc-odbc bridge).
- Mềm dẻo nhưng không hiệu quả.



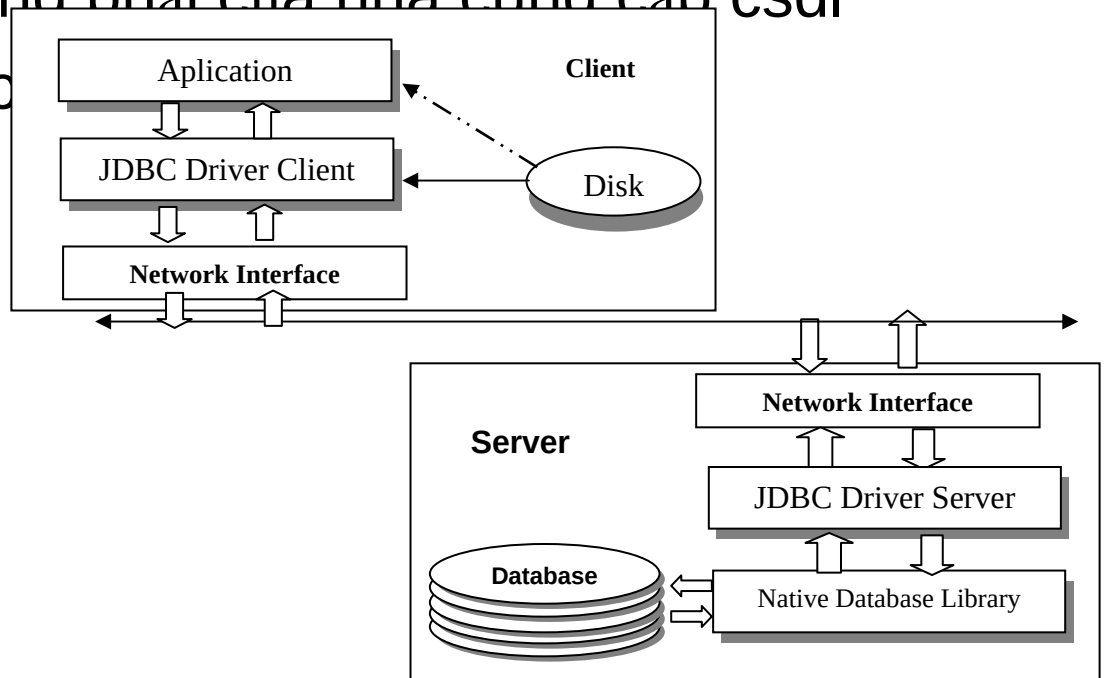
# Loại 2: Native-API

- Tốt hơn loại 1, loại này cho phép JDBC giao tiếp trực tiếp với các driver hay các hàm API của CSDL.



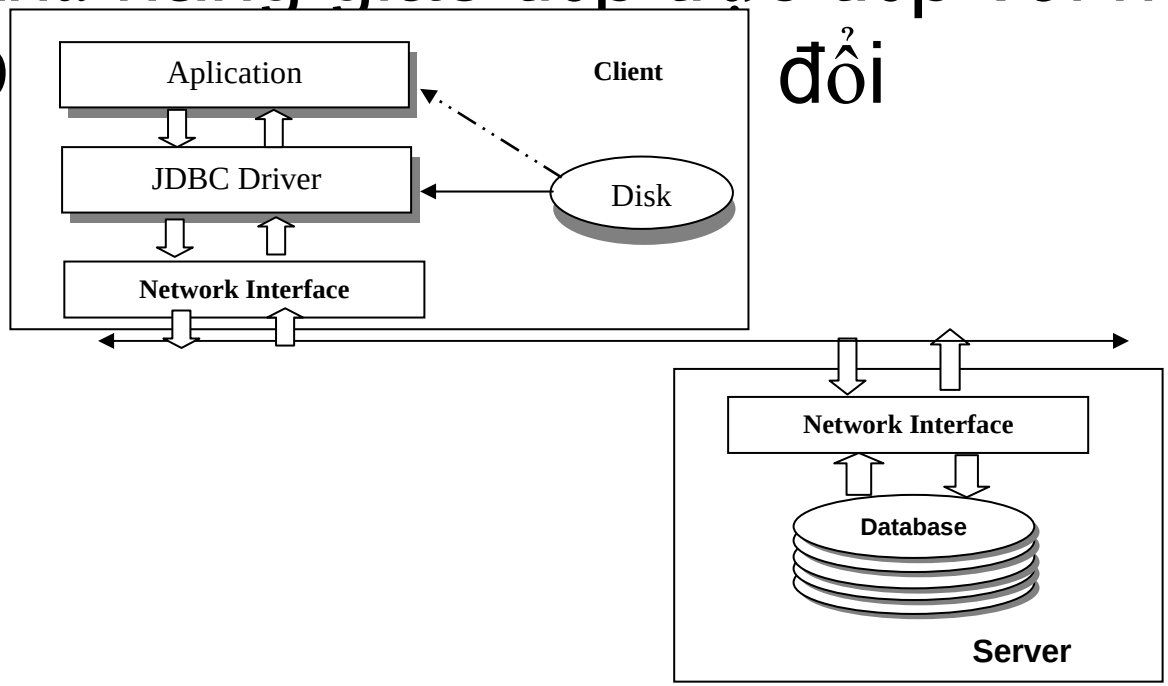
# Loại 3: Open Protocol-Net

- Drivers
  - Có thể chuyển các yêu cầu đến các csdl nằm ở xa.
  - Có thể giao tiếp với nhiều loại CSDL.
  - Không phải của nhà cung cấp csdl
  - Tất c



# Loại 4: Proprietary-Protocol Net

- 100% java
- Có khả năng giao tiếp trực tiếp với hệ CSD đổi



# Gói Java.sql

- Cung cấp tập hợp các lớp và interface dùng để trao đổi với CSDL.
- Các lớp
  - DriverManager
  - Date, Time
  - Timestamp
  - Types
- Các Interfaces

  - Driver
  - Connection
  - DatabaseMetaData
  - Statement
  - PreparedStatement
  - CallableStatement
  - ResultSet
  - ResultSetMetaData

# Đối tượng Statement

- Đối tượng Connection chứa liên kết trực tiếp đến csdl.
- Sử dụng đối tượng Connection để tạo đối tượng Statement.
  - Statement s = con.createStatement();
- Đối tượng này có nhiệm vụ gửi các câu lệnh sql đến csdl.
- **executeQuery(String)** or **executeUpdate(String)** method
- Cùng một đối tượng Statement có thể sử dụng cho nhiều câu lệnh sql khác nhau.

- Có 3 phương thức thực thi
  - executeQuery()
  - executeUpdate()
  - execute()
- The executeQuery()
  - Nhận câu lệnh SQL (select) làm đối số, trả lại đối tượng ResultSet
- ResultSet rs = s.executeQuery(“SELECT * FROM Books”);

- Phương thức executeUpdate()
  - Nhận các câu lệnh sql dạng cập nhật
  - Trả lại số nguyên biểu thị số hàng được cập nhật.
  - UPDATE, INSERT, or DELETE.
- Phương thức execute()
  - Được áp dụng cho trường hợp không rõ loại sql nào được thựưc hiện.
  - Được áp dụng cho trường hợp câu lệnh sql đwocj tạo ra tự động bởi chương trình.

# ResultSet

- Chứa một hoặc nhiều hàng dữ liệu từ việc thực hiện câu lệnh truy vấn.
- Có thể lấy dữ liệu từng hàng dữ liệu một trong ResultSet.
- Sử dụng phương thức next() để di chuyển đến hàng  dữ liệu tiếp theo trong ResultSet.
- Hàm next() trả lại true chỉ rằng hàng chứa dữ liệu, trả lại false hàng cuối không chứa dữ liệu.
- Thực hiện

```
while (rs.next()){
   // examine a row from the results
   }
```

- Để lấy dữ liệu ở các cột trên mỗi hàng của ResultSet, ta dùng các phương thức.
  - get type(int | String)
    - Đối số là chỉ số cột tính từ 1.
    - Áp dụng cho các cột có kiểu dữ liệu là int, float, Date.....
  - Ví dụ :
    - String isbn = rs.getString(1); // Column 1
    - float price = rs.getDouble(“Price”);

# ResultSet Metadata

- Đối tượng này cho biết thông tin về ResultSet

- ***ResultSet rs = stmt.executeQuery(SQLString);***
  ***ResultSetMetaData rsmd = rs.getMetaData();***
  ***int numberOfColumns = rsmd.getColumnCount();***
- ***getColumnName(int column)***

# Prepared Statements

- To execute a Statement object many times, it will reduce execution time to use PreparedStatement object
- PreparedStatement object
  - unlike a Statement object, it is given an SQL statement when it is created.
  - The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled.
  - As a result, the PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled.
  - This means that when the PreparedStatement is

# Các đối tượng Statement khác

- Prepared Statements
- Callable Statement

# Database Metadata

- Đối tượng này cho biết thông tin về csdl.

# Chương trình mẫu

```
import java.sql.*;
class JDBCDemo1 {
    public static void main(String[] args) {

  try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     Connection con=DriverManager.getConnection("jdbc:odbc:Accserver");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(args[0]);
    ResultSetMetaData rsmd = rs.getMetaData();
    int numberOfColumns = rsmd.getColumnCount();
    for(int j=1; j<=numberOfColumns;j++) {
            System.out.println(rsmd.getColumnLabel(j));
            }
    while(rs.next()) {
            for(int i=1; i<=numberOfColumns;i++){
            System.out.println(rs.getObject(i));
                        }}
            rs.close();
            stmt.close();
    } catch(Exception e){  System.out.println("Error " + e);
```