TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM

LẬP TRÌNH JAVA

NỘI DUNG MÔN HỌC

Chương 1: Giới thiệu về Java

Chương 2: Hướng đối tượng trong Java

Chương 3: Luồng nhập xuất và tập tin

Chương 4: Lập trình Multithread trong Java

Chương 5: Lập trình GUI với AWT & Swing

Chương 6: Lập trình CSDL với JDBC

TÀI LIỆU THAM KHẢO

[1] Cay S. Horstmann, Gary Cornell. Core Java™ 2: Volume I – Fundamentals, Prentice Hall , 2002

[2] H. M. Deitel. Java™ How to Program, Prentice Hall , 2004.

[3] Marty Hall. Core Servlet and Java Server Page. Sun Micro System. Prentice Hall PTR; 1 edition 2000

[4] Subrahmanyam Allamaraju, Andrew Longshaw et al. Professional Java Server Programming J2EE Edition – Wrox 2001

...

https://sites.google.com/site/tinhuynhuit/courses/java-programming

HÌNH THỨC ĐÁNH GIÁ

Báo cáo seminar nhóm: 30%
Bài thu hoạch nhóm: 70% (phát triển từ seminar)

CHƯƠNG 1

TỔNG QUAN VỀ JAVA

NỘI DUNG

- Lịch sử phát triển
- Java Flatforms
- Các dạng chương trình Java
- Đặc điểm của Java
- Máy ảo Java (Java Virtual Machine)
- Viết, dịch, thực thi chương trình HelloWorld
- Môi trường, công cụ: giới thiệu một số IDE phổ biến

Lịch sử phát triển

• 1991: Sun Microsystems phát triển OAK nhằm mục đích viết phần mềm điều khiển (phần mềm nhúng) cho các sản phẩm gia dụng.





• 1995: internet bùng nổ, phát triển mạnh. Sun phát triển OAK và giới thiệu ngôn ngữ lập trình mới tên Java


• **Java** là ngôn ngữ hướng đối tượng tựa C, C++

Lịch sử phát triển
Java Development Kit (JDK)

Môi trường phát triển và thực thi do Sun Microsystems cung cấp (http://java.sun.com)

http://www.oracle.com/technetwork/java/index.html

Bao gồm phần mềm và công cụ giúp compile, debug and execute ứng dụng.

JDK 1.0	- 1996
JDK 1.1	- 1997
JDK 1.2 (Java 2)	- 1998
JDK 1.3	- 2000
Java 1.4	- 2002
Java 5 (1.5)	- 2004
Java 6	- 2006

- **Oracle mua Sun - April 20, 2009 - $7.4 billion**

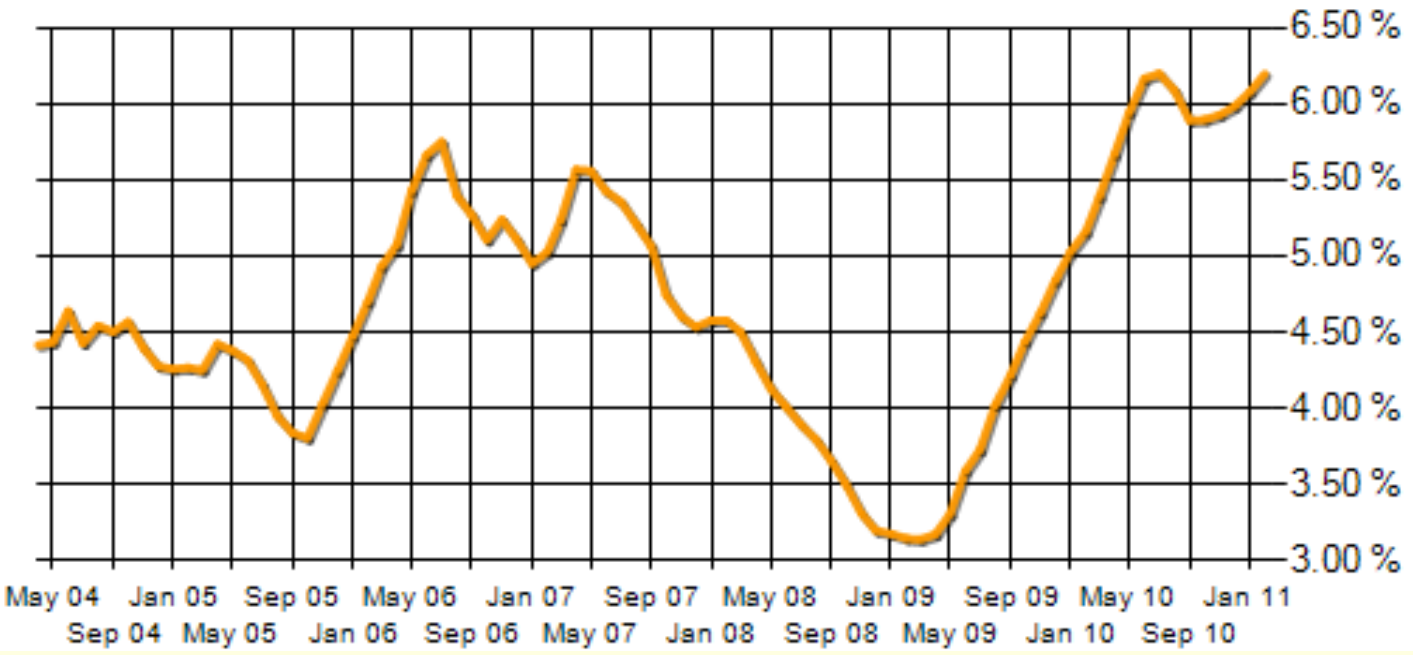
Java Flatforms

- Flatform J2SE (Java 2 Standard Edition)
- Flatform J2EE (Java 2 Enterprise Edition)
- Flatform J2ME(Java 2 Micro Edition)

Một số thống kê liên quan đến Java

Java Developer
Demand Trend

The chart provides the 3-month moving total beginning in 2004 of permanent IT jobs citing **Java Developer** within the UK as a proportion of the total demand within the **Job Titles** category.

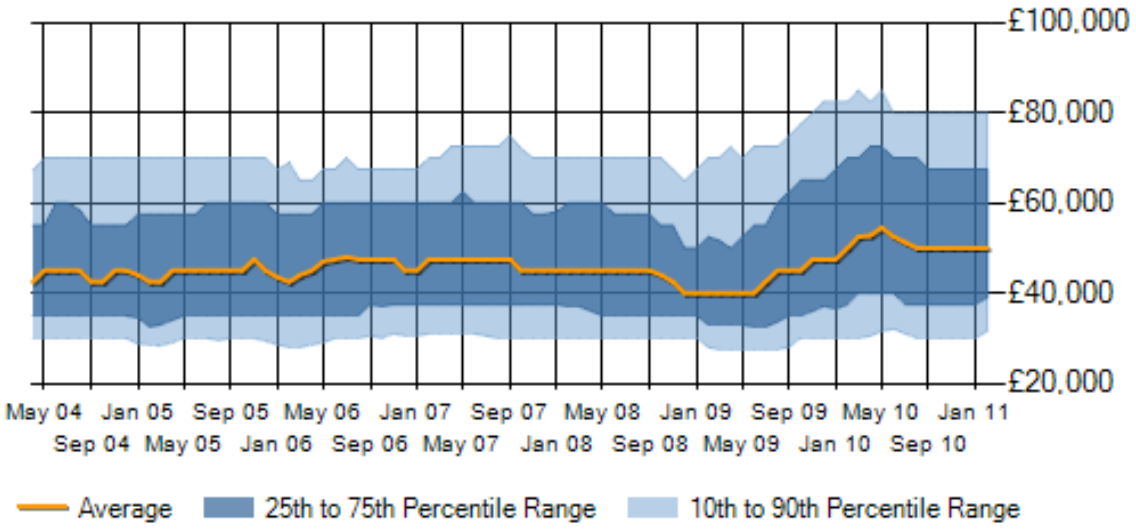


http://www.itjobswatch.co.uk/jobs/uk/java%20developer.do

Một số thống kê liên quan đến Java

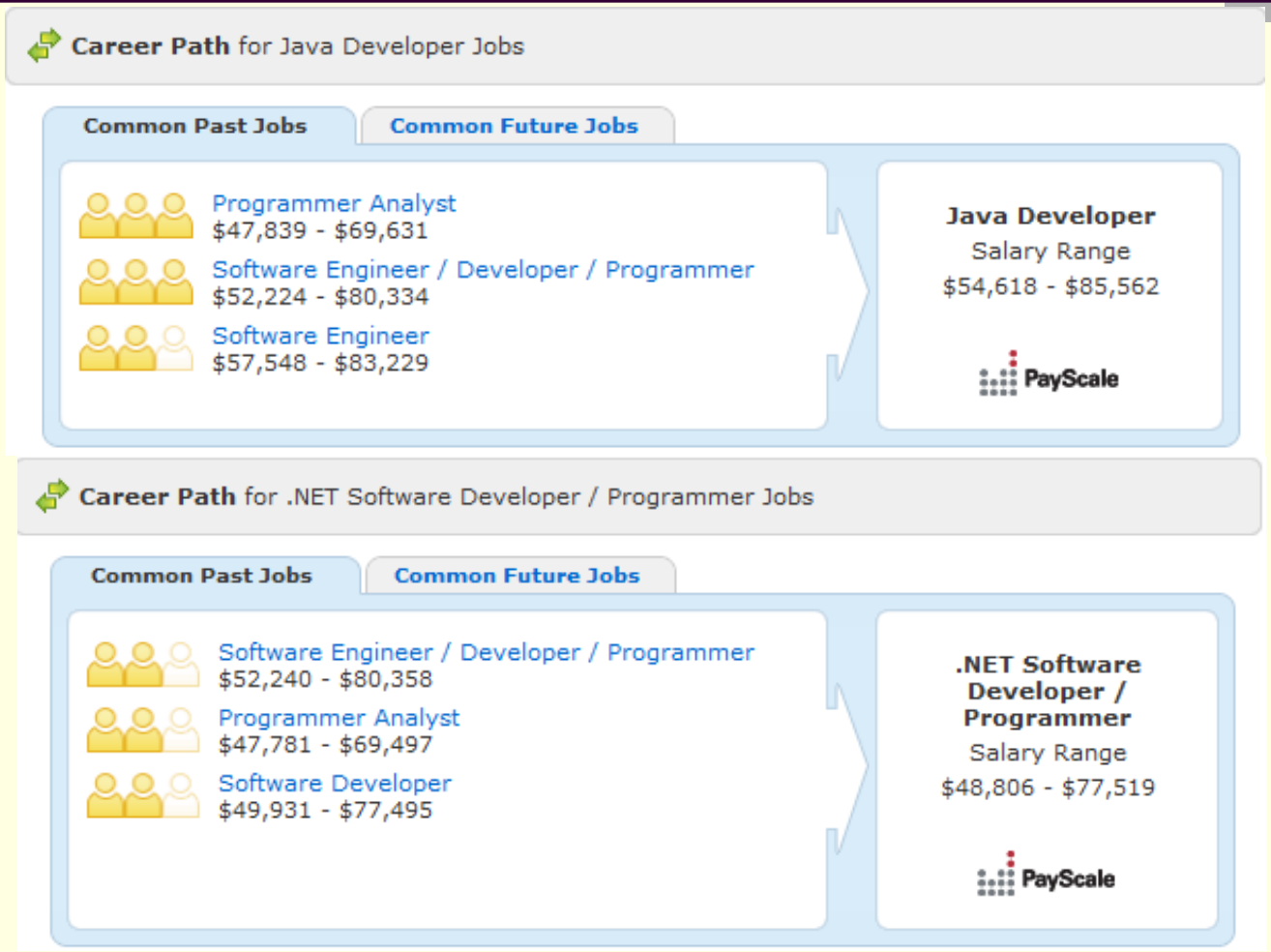
Java Developer
Salary Trend

The chart provides the 3-month moving average for salaries quoted in permanent IT jobs citing **Java Developer** within the UK.



http://www.itjobswatch.co.uk/jobs/uk/java%20developer.do

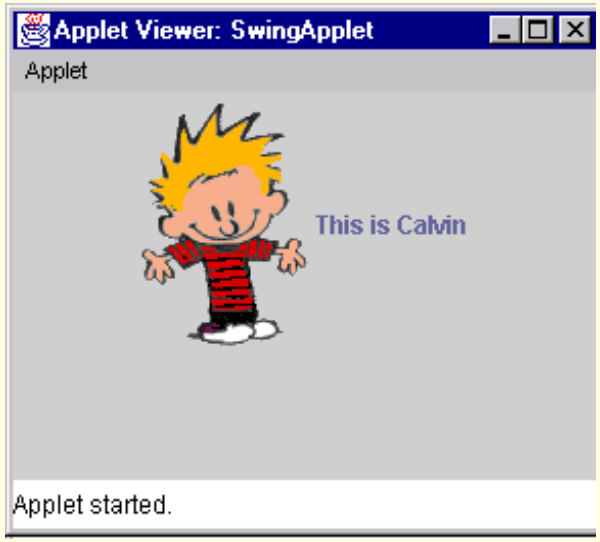
Một số thống kê liên quan đến Java

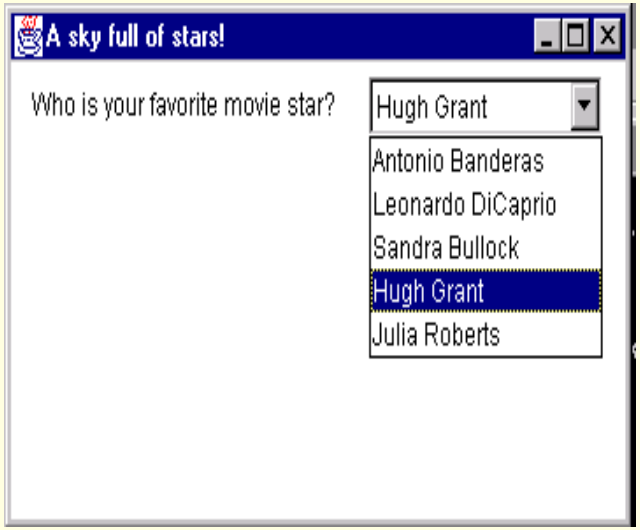


http://www.payscale.com/research/US/Job=Java_Developer/Salary
http://www.payscale.com/research/US/Job=.NET_Software_Developer_%2F_Programmer/Salary

Các dạng chương trình java

- **Applets:**



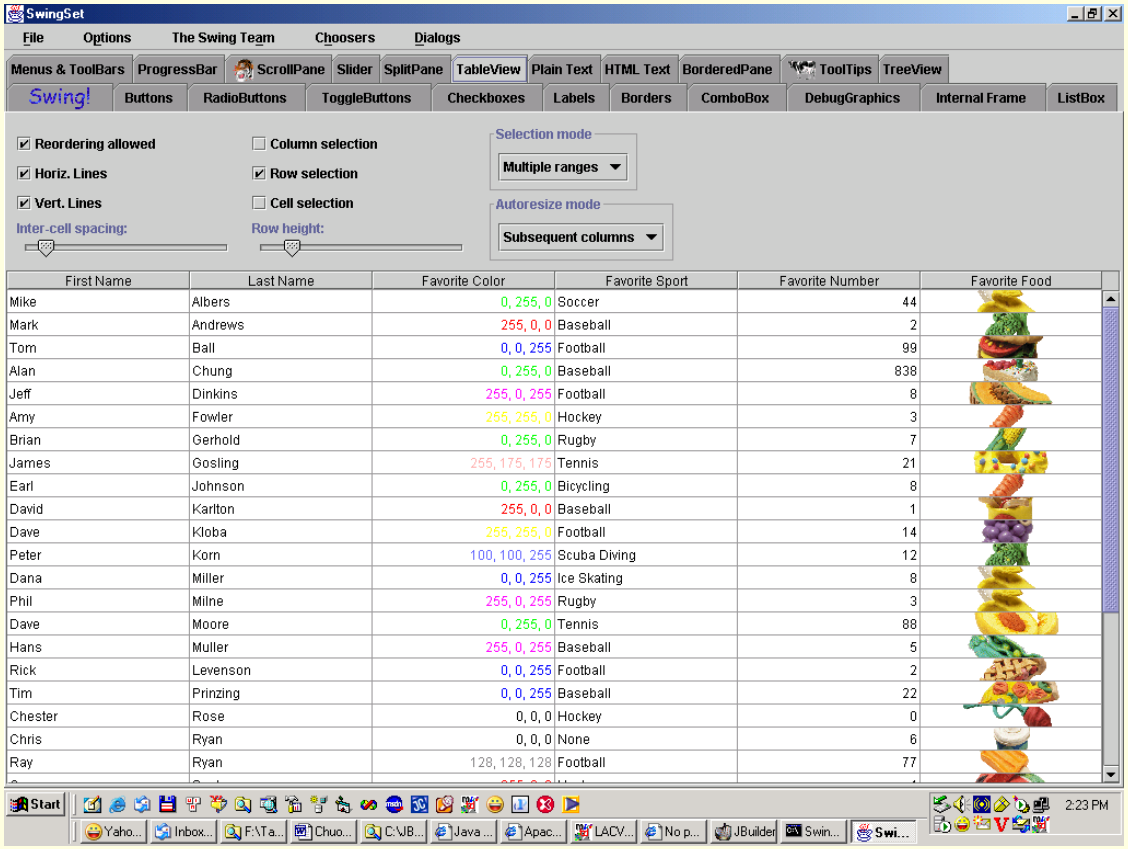


Các dạng chương trình java

- **Console Applications**

Các dạng chương trình java

- Ứng dụng Desktop



Các dạng chương trình java

- Ứng dụng Web

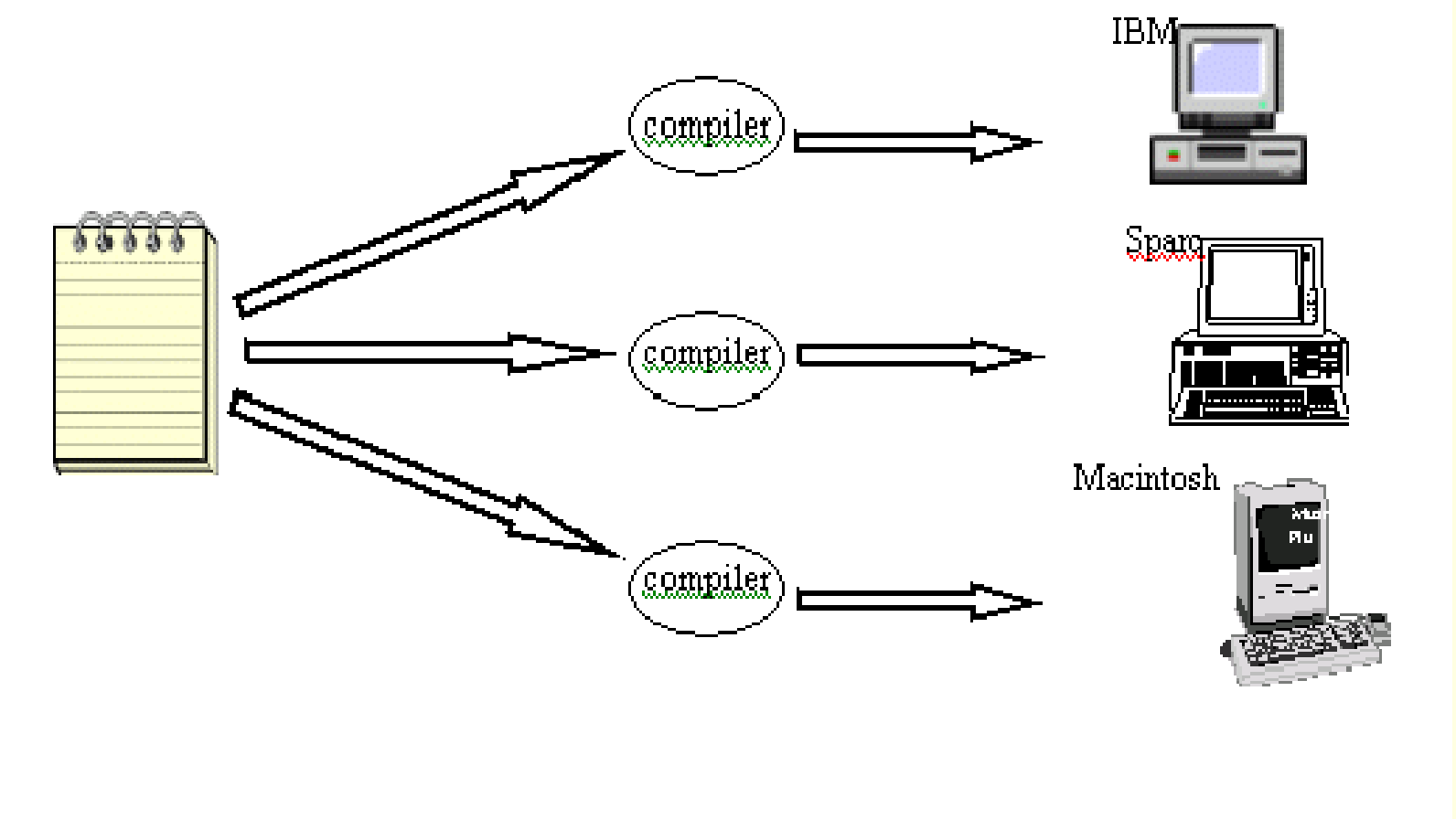
Các dạng chương trình java

- Một dạng phần mềm trên thiết bị di động

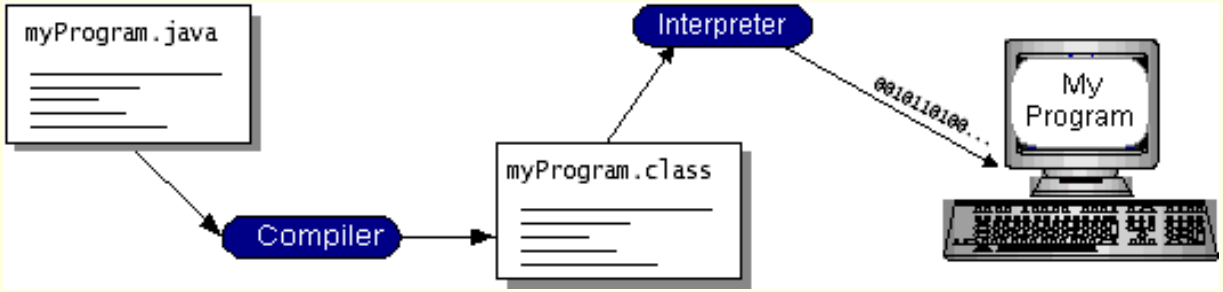
Đặc điểm java

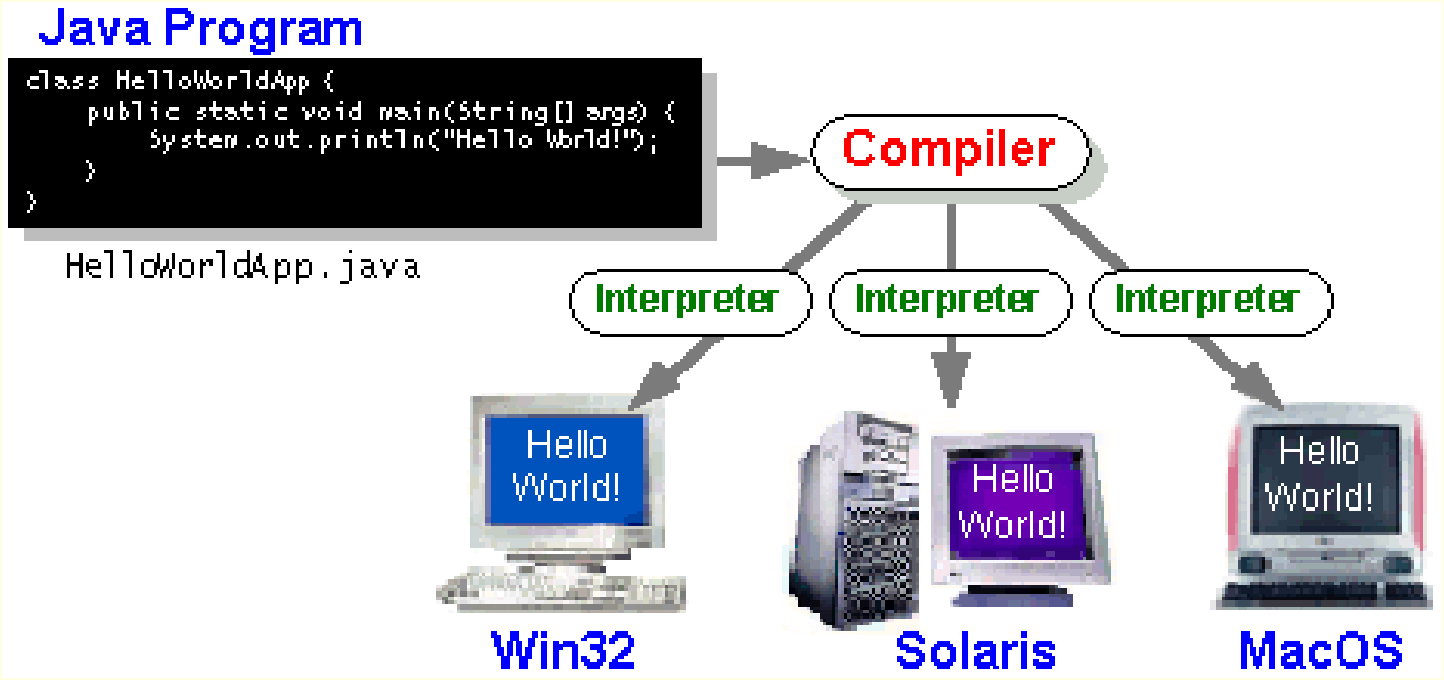
- Tựa C++, hướng đối tượng hoàn toàn.
- Khả chuyển, độc lập nền.
- Thông dịch (vừa biên dịch vừa thông dịch).
- Cơ chế giải phóng bộ nhớ tự động.
- An toàn, bảo mật.

Chương trình truyền thống



Dịch và thực thi chương trình java





Java Virtual Machine

- Là phần mềm dựa trên cơ sở máy tính ảo.
- Có thể xem như 1 hệ điều hành thu nhỏ.
- Cung cấp môi trường thực thi cho chương trình java (độc lập nền)
- Hình thành 1 lớp trừu tượng:

  Phần cứng máy tính bên dưới

  Hệ điều hành

  Mã đã biên dịch

- Chương trình java chỉ chạy khi có JVM
- JVM đọc và thực thi từng câu lệnh java
- …

Viết và thực thi chương trình Hello World

• Dùng Notepad soạn thảo đoạn lệnh bên dưới và lưu lại với tên HelloWorld.java

```
import java.io.*;
class HelloWorld
{
        public static void main(String args[])
        {
                System.out.print("Hello Class");
        }
}
```

Khai báo thư viện java.io

Định nghĩa lớp tên "**HelloWorld**"

Bắt đầu đoạn lệnh

Phương thức main

Xuất ra Console thông báo

Kết thúc đoạn lệnh

Viết và thực thi chương trình Hello World (tt)

• **Biên dịch**: dùng chương trình **javac**

C:\> javac HelloWorld.java

Biên dịch thành công tạo ra tập tin có đuôi .class (HelloWorld.class)

• **Thông dịch (thực thi):** dùng chương trình **java**

C:\> java HelloWorld

Lưu ý: Phải khai báo đường dẫn chỉ đến thư mục cài đặt java, và thư mục chứa các class cần thực thi

Ví dụ: *C:\> set path=C:\jdk1.5\bin*

C:\> set classpath = D:\ThucHanhJava\BT1

Môi trường, công cụ

- Môi trường phát triển và thực thi của Sun – JDK 1.5
- **IDE (Integrated Development Enviroment)**
  - ✓ Jcreator Pro 3.5
  - ✓ NetBean 5.5
  - ✓ Eclipse 3.2
  - ✓ Jbuilder 9.0
  - ✓ …

CĂN BẢN VỀ NGÔN NGỮ JAVA

NỘI DUNG

- Biến & Hằng
- Kiểu dữ liệu (kiểu cơ sở, kiểu tham chiếu)
- Toán tử, biểu thức
- Các cấu trúc điều khiển (chọn, rẽ nhánh, lặp)
- Lớp bao kiểu cơ sở
- Phương thức và cách sử dụng
- Một số ví dụ minh họa

Biến

• Biến là một vùng nhớ lưu các giá trị của chương trình

• Mỗi biến gắn với 1 kiểu dữ liệu và 1 định danh duy nhất là tên biến

• Tên biến phân biệt chữ hoa và chữ thường. Tên biến bắt đầu bằng 1 dấu _, $, hay 1 ký tự, không được bắt đầu bằng 1 ký số.

Khai báo

<kiểu dữ liệu> <tên biến>;

<kiểu dữ liệu> <tên biến> = <giá trị>;

Gán giá trị

<tên biến> = <giá trị>;

Phân loại biến

- Biến trong Java có 2 loại: instance varible và local variable.
- Đối với instance variable, có thể được sử dụng mà không cần khởi tạo giá trị (được tự động gán giá trị mặc định).
- Đối với local variable, Java bắt buộc phải khởi tạo giá trị trước khi sử dụng. Nếu không sẽ tạo ra lỗi khi biên dịch khi sử dụng.

Hằng

- Là một giá trị bất biến trong chương trình
- Tên đặt theo qui ước như tên biến
- Được khai báo dùng từ khóa **final,** và thường dùng tiếp vĩ ngữ đối với các hằng số (l, L, d, D, f, F)
- Ví dụ:

  final int x = 10; // khai báo hằng số nguyên x = 10

  final long y = 20L; // khai báo hằng số long y = 20

- Hằng ký tự: đặt giữa cặp nháy đơn ‘’
- Hằng chuỗi: là một dãy ký tự đặt giữa cặp nháy đôi “”

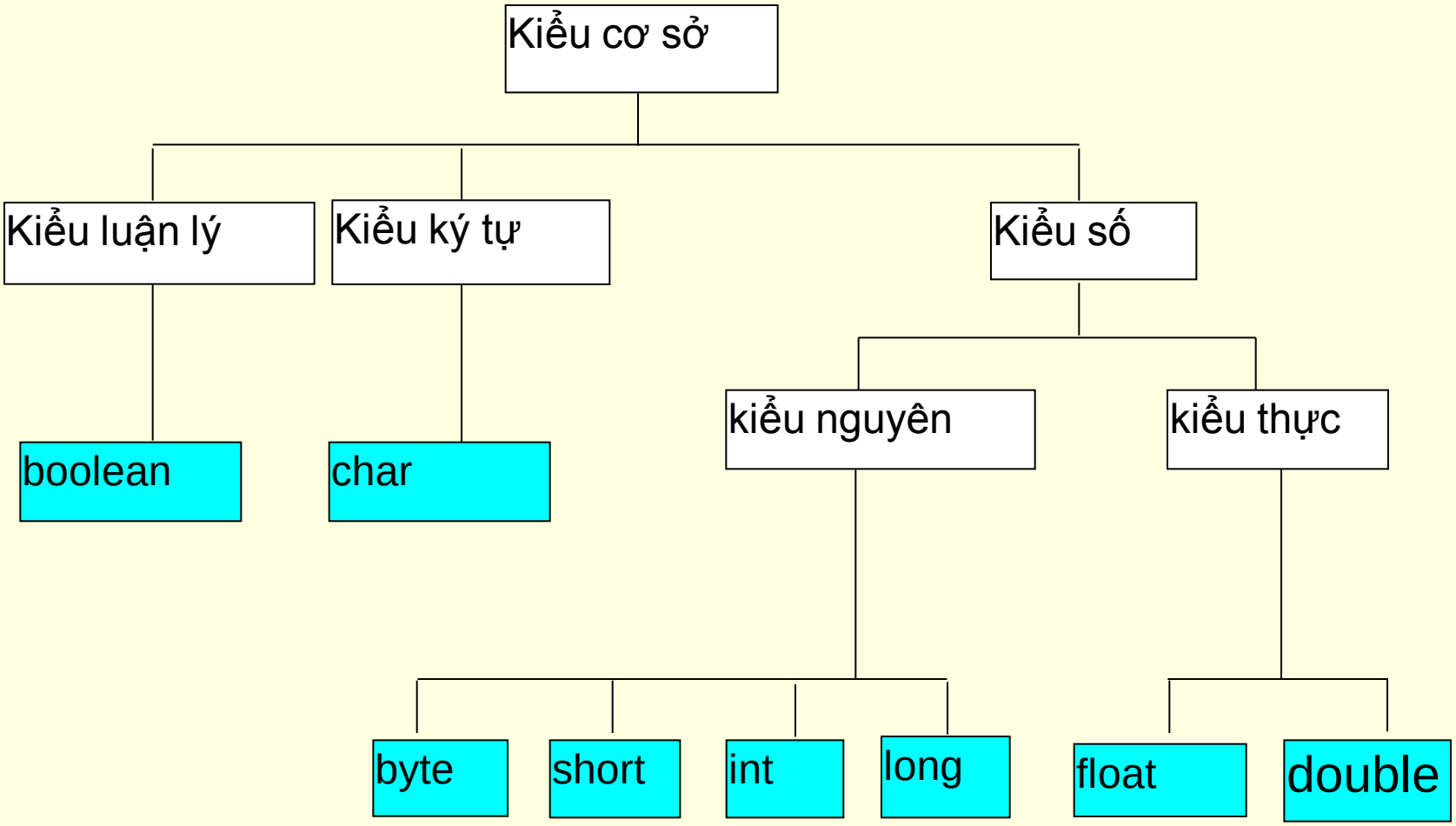
Hằng ký tự đặc biệt

Ký tự	Ý nghĩa
\b	Xóa lùi (BackSpace)
\t	Tab
\n	Xuống hàng
\r	Dấu enter
\”	Nháy kép
\’	Nháy đơn
\\	\
\f	Đẩy trang
\uxxxx	Ký tự unicode

Kiểu dữ liệu

- Kiểu dữ liệu cơ sở (primitive data type)
- Kiểu dữ liệu tham chiếu (reference data type)

Kiểu dữ liệu cơ sở



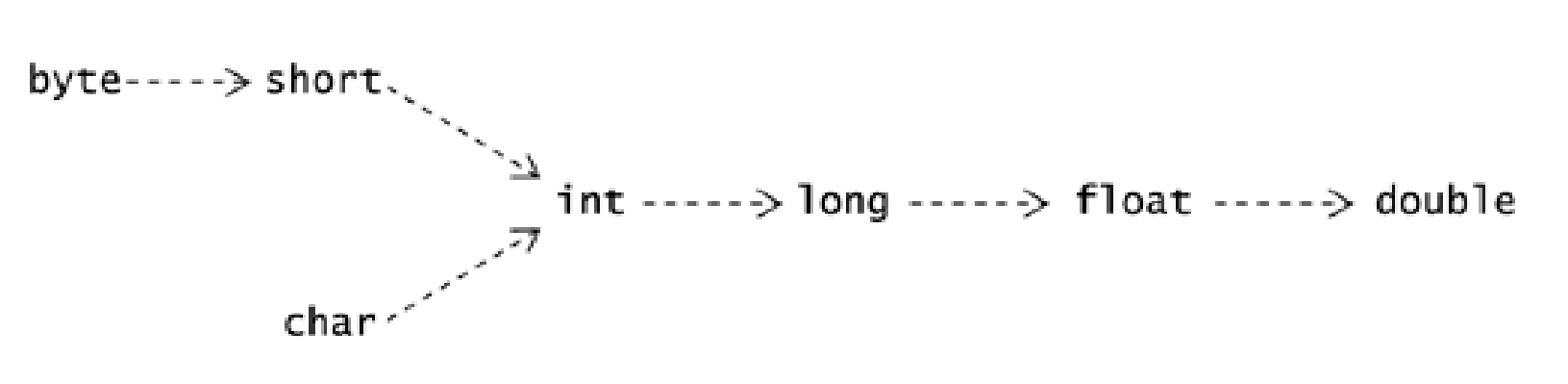
Kiểu dữ liệu cơ sở (tt)

Kiểu	Kích thước (bits)	Giá trị	Giá trị mặc định
boolean	[*Note:* The representation of a boolean is specific to the Java Virtual Machine on each computer platform.]	true và false	**false**
char	16	'\u0000' to '\uFFFF' (0 to 65535)	**null**
byte	8	–128 to +127 (-2^7 to $2^7 - 1$)	**0**
short	16	–32,768 to +32,767 (-2^{15} to $2^{15} - 1$)	**0**
int	32	–2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)	**0**
long	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (–263 to 263 – 1)	**0l**
float	32	1.40129846432481707e–45 to 3.4028234663852886E+38	**0.0f**
double	64	4.94065645841246544e–324 to 1.7976931348623157E+308	**0.0d**

Kiểu dữ liệu cơ sở (tt)

- **Chuyển đổi kiểu dữ liệu**: khi có sự không tương thích về kiểu dữ liệu (gán, tính toán biểu thức, truyền đối số gọi phương thức)
  - ✓ Chuyển kiểu hẹp (lớn → nhỏ): ***cần ép kiểu***

    <tên biến 2> = (kiểu dữ liệu) <tên biến 1>;
  - ✓ Chuyển kiểu rộng (nhỏ → lớn): ***tự động chuyển***



Kiểu dữ liệu cơ sở (tt)

- **Lưu ý**

1. *Không thể chuyển đổi giữa kiểu boolean với int và ngược lại.*
2. ***Nếu** 1 toán hạng kiểu **double** thì*

   *"Toán hạng kia chuyển thành **double**"*

   ***Nếu** 1 toán hạng kiểu **float** thì*

   *"Toán hạng kia chuyển thành **float**"*

   ***Nếu** 1 toán hạng kiểu **long** thì*

   *"Toán hạng kia chuyển thành **long**"*

   ***Ngược lại** "Tất cả chuyển thành **int** để tính toán"*

Kiểu dữ liệu cơ sở (tt)

- **Ví dụ minh họa**

1. byte x = 5;

2. byte y = 10;

3. byte z = x + y;

// Dòng lệnh thứ 3 báo lỗi chuyển kiểu cần sửa lại

// byte z = (byte) (x + y);

Kiểu dữ liệu tham chiếu

- **Kiểu mảng**

  ✓ Mảng là tập hợp các phần tử có cùng tên và cùng kiểu dữ liệu.

  ✓ Mỗi phần tử được truy xuất thông qua chỉ số

  - **Khai báo mảng**

    <kiểu dữ liệu>[] <tên mảng>; // mảng 1 chiều

    <kiểu dữ liệu> <tên mảng>[]; // mảng 1 chiều

    <kiểu dữ liệu>[][] <tên mảng>; // mảng 2 chiều

    <kiểu dữ liệu> <tên mảng>[][]; // mảng 2 chiều

Kiểu dữ liệu tham chiếu (tt)

- **Khởi tạo**

  int arrInt[] = {1, 2, 3};

  char arrChar[] = {'a', 'b', 'c'};

  String arrString[] = {"ABC", "EFG", "GHI"};

- **Cấp phát & truy cập mảng**

  *int arrInt = **new** int[100];*

  int arrInt[100]; // Khai báo này trong Java sẽ bị báo lỗi.

  Chỉ số mảng ***n*** phần tử: từ ***0*** đến ***n-1***

Kiểu dữ liệu tham chiếu (tt)

- **Kiểu đối tượng**

  Khai báo đối tượng

  <Kiểu đối tượng> <biến ĐT>;

  Khởi tạo đối tượng

  <Kiểu đối tượng> <biến ĐT> = ***new*** *<Kiểu đối tượng>*;

  Truy xuất thành phần đối tượng

  <biến ĐT>.<thuộc tính>

  <biến ĐT>.<phương thức>

Toán tử, biểu thức

- **Toán tử số học**

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia nguyên
%	Chia dư
++	Tăng 1
--	Giảm 1

Toán tử, biểu thức (tt)

- **Phép toán trên bit**

Toán tử	Ý nghĩa
&	AND
|	OR
^	XOR
<<	Dịch trái
>>	Dịch phải
~	Bù bit

Toán tử, biểu thức (tt)

- **Toán tử quan hệ & logic**

Toán tử	Ý nghĩa
= =	So sánh bằng
!=	So sánh khác
>	So sánh lớn hơn
<	So sánh nhỏ hơn
>=	So sánh lớn hơn hay bằng
<=	So sánh nhỏ hơn hay bằng
||	OR (biểu thức logic)
&&	AND (biểu thức logic)
!	NOT (biểu thức logic)

Toán tử, biểu thức (tt)

- **Toán tử gán**

Toán tử	Ví dụ	Ý nghĩa
=	a = b	gán a = b
+=	a += 5	a = a + 5
-=	b -= 10	b = b – 10
*=	c *= 3	c = c * 3
/=	d /= 2	d = d/2
%=	e %= 4	e = e % 4

Toán tử, biểu thức (tt)

- **Toán tử điều kiện**

Cú pháp: *<điều kiện> ? <biểu thức 1> : < biểu thức 2>*

Ví dụ:

```
int x = 10;
int y = 20;
int Z = (x<y) ? 30 : 40;
// Kết quả z = 30 do biểu thức (x < y) là đúng.
```

Cấu trúc điều khiển

- **Cấu trúc *if … else***

Dạng 1:

```
if (<điều_kiện>) {
        <khối_lệnh>;
}
```

Dạng 2:

```
if (<điều_kiện>)  {
        <khối _lệnh1>;
}
else {
        <khối _lệnh2>;
}
```

Cấu trúc điều khiển

- **Cấu trúc *switch ... case***

```
switch (<biến>) {
        case <giátrị_1>:
                <khối_lệnh_1>;
                break;
        ….
        case <giátrị_n>:
                <khối_lệnh_n>;
                break;
        default:
                <khối lệnh default>;
}
```

Cấu trúc điều khiển

- **Cấu trúc lặp**
  - **Dạng 1:** *while (<điều_kiện_lặp>) {*

    <khối _lệnh>;

    }
  - **Dạng 2:** *do {*

    <khối_lệnh>;

    } while (điều_kiện);
  - **Dạng 3:** *for (khởi_tạo_biến_đếm;đk_lặp;tăng_biến) {*

    <khối _lệnh>;

    }

Cấu trúc điều khiển

- **Cấu trúc lệnh nhảy jump:** dùng kết hợp nhãn (label) với từ khóa ***break*** và ***continue*** để thay thế cho lệnh ***goto*** (trong C).

Ví dụ:

```
label:
for (…) {
    for (…) {
        if (<biểu thức điều kiện>)
            break label;
        else
            continue label;
    }
}
```

Lớp bao kiểu dữ liệu cơ sở

Data type	Wrapper Class (java.lang.*)	Ghi chú
boolean	Boolean	- Gói (package): chứa nhóm nhiều class. - Ngoài các Wrapper Class, gói java.lang còn cung cấp các lớp nền tảng cho việc thiết kế ngôn ngữ java như: String, Math, …
byte	Byte	
short	Short	
char	Character	
int	Integer	
long	Long	
float	Float	
double	Double	

QUẢN LÝ EXCEPTIONS

Quản lý Exception

- Giới thiệu về Exception
- Kiểm soát Exception
- Ví dụ minh họa
- Thư viện phân cấp các lớp Exception

Giới thiệu về Exception

Ví dụ 1:

```
...
int x = 10;
int y = 0;
float z = x/y;
System.out.print("Ket qua la:" + z);
...
```

Dòng lệnh thứ 3 có lỗi chia cho 0, vì vậy đoạn chương trình kết thúc và dòng lệnh thứ 4 xuất kết quả ra màn hình không thực hiện được.

Giới thiệu về Exception

Ví dụ 2:

```
…
void docfile(String filename)
{       …
        FileInputStream fin = new FileInputStream(filename);
        …
}
```

Dòng lệnh trên có khả năng xảy ra lỗi đọc file (chẳng hạn khi file không có trên đĩa)

Giới thiệu về Exception

- **Exception**
  - ✓ Dấu hiệu của lỗi trong khi thực hiện chương trình
  - ✓ *ví dụ*: lỗi chia cho **0,** đọc file không có trên đĩa, …
- **Quản lý Exception (Expcetion handling)**
  - ✓ Kiểm soát được lỗi từ những thành phần chương trình
  - ✓ Quản lý Exception theo 1 cách thống nhất trong những project lớn
  - ✓ Hạn chế, bỏ bớt những đoạn source code kiểm tra lỗi trong chương trình.

Kiểm soát Exception

Ví dụ 1:

…

```
try {
        int x = 10;
        int y = 0;
        float z = x/y;
        System.out.print("Ket qua la:" + z);
}
catch(ArithmeticException e) {
            System.out.println(“Loi tinh toan so hoc”)
}
```

…

Kiểm soát Exception

Ví dụ 2:

```
…
void docfile(String  filename) throws IOException  {
        …
        FileInputStream fin = new FileInputStream(filename);
        …
}
```

Kiểm soát Exception

Hoặc

…

```
void docfile(String  filename) {  …
    try {
            …
        FileInputStream fin = new FileInputStream(filename);
            …
    }
    catch (IOException e) {
            System.out.println("Loi doc file");
    }
}
```

Kiểm soát Exception

• Khi có lỗi phương thức sẽ ném ra một exception

• Việc kiểm soát exception giúp chương trình kiểm soát được những trường hợp ngoại lệ và xử lý lỗi.

• Những lỗi không kiểm soát được sẽ có những ảnh hưởng bất lợi trong chương trình.

• Dùng từ khóa **throws** để chỉ định những loại exception mà phương thức có thể ném ra.

<tiền tố> <tên phương thức>(<đối số>) **throws** <các exceptions>

Kiểm soát Exception

• Đoạn code có thể sinh ra lỗi cần đặt trong khối lệnh bắt đầu bằng **try**.

• Đoạn code để kiểm tra, xử lý trong trường hợp có lỗi xảy ra đặt trong khối lệnh **catch**.

```
try {
        // Đoạn mã có thể sinh ra lỗi …
}
catch (<Kiểu Exception>){
        // Đoạn mã kiểm soát lỗi

}
```

Kiểm soát Exception

• Khối lệnh đặt trong **finally** luôn được thực thi cho dù có Exception hay không.

• Thường dùng để giải phóng tài nguyên

```
try {
        // Đoạn mã có thể sinh ra lỗi …
}
Catch (<Kiểu Exception>) {
        // Đoạn mã kiểm soát lỗi
}
finally {
        // Đoạn mã luôn luôn được thực thi
}
```

Kiểm soát Exception

```
try {
        // Khối lệnh trước dòng lệnh sinh ra lỗi
        // Dòng lệnh sinh ra lỗi (Exception)
        …
}
catch (<Kiểu Exception>){
        // Đoạn mã kiểm soát lỗi
}
finally { …
}
```

Khối lệnh sau dòng lệnh sinh ra lỗi sẽ bị bỏ qua và không thực hiện khi có exception

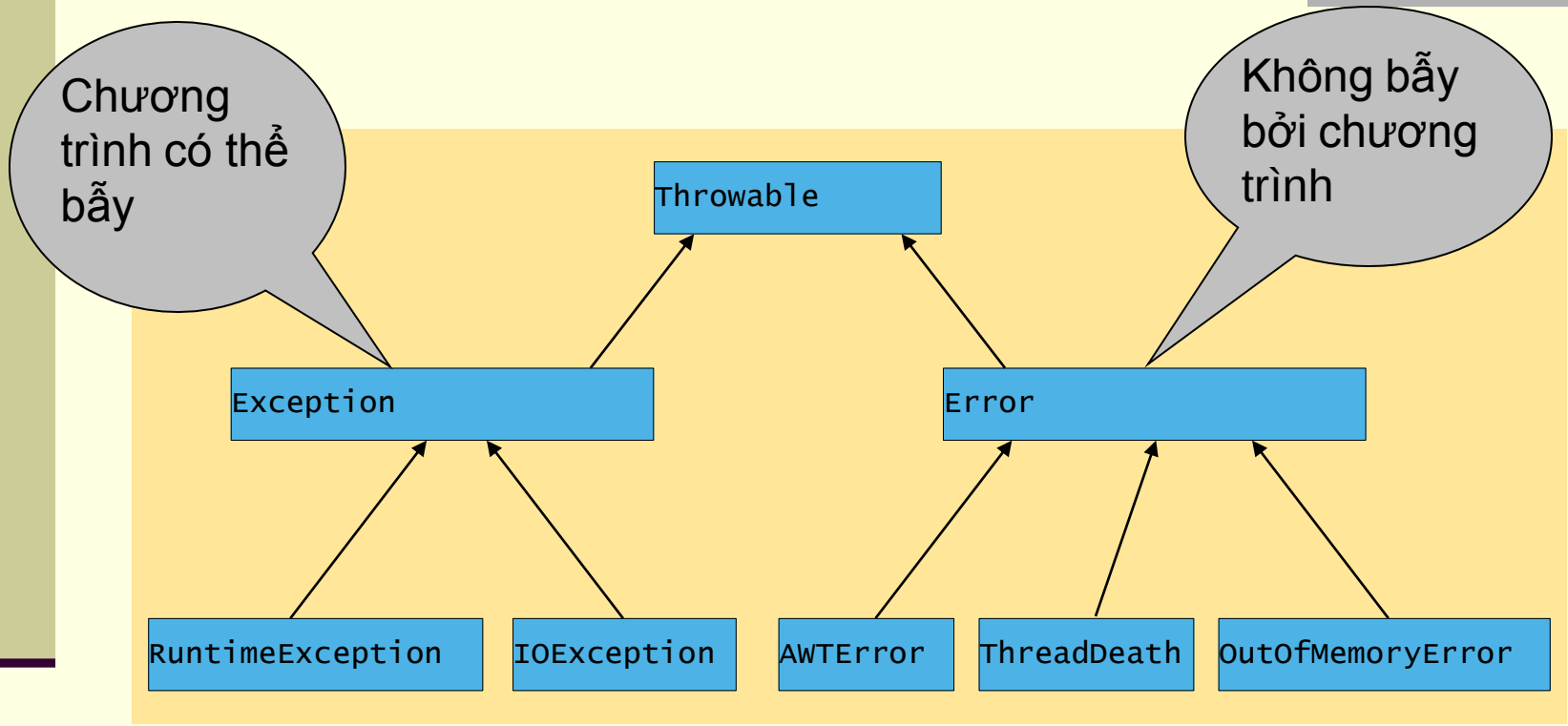
Ví dụ kiểm soát Exception: chia cho 0

```
import java.io.*;
public class MainClass {
public static void main(String[] args) {
    try {
            int num_1, num_2;
            BufferedReader in = new BufferedReader(new
            InputStreamReader(System.in));
            System.out.print("\n Nhap so thu 1:");
            num_1 = Integer.parseInt(in.readLine());
            System.out.print("\n Nhap so thu 2:");
            num_2 = Integer.parseInt(in.readLine());
            float rs = num_1/num_2;
            System.out.print("\n Ket qua:" + rs);
        }
```

Ví dụ kiểm soát Exception: chia cho 0

```
        catch (ArithmeticException e) {
                System.out.print("Loi chia cho 0");
        }
        catch (IOException e) {
                System.out.print("Loi xuat nhap");
        }
        catch(Exception e) {
                System.out.print("Loi khac");
        }
        System.out.print(“Kiem soat duoc loi hay Khong co loi");
    }
}
```

Thư viện các lớp Throwable



• Có thể định nghĩa các exception mới bằng cách dẫn xuất (**extends**) từ những lớp Exception đang có.