Bài 8. Kế thừa. Xử lí ngoại lệ.

THCS 4: Lập trình cơ bản với Java

Đỗ Thanh Hà, Nguyễn Thị Minh Huyền

Bộ môn Tin học
Khoa Toán - Cơ - Tin học
Trường Đại học Khoa học Tự nhiên

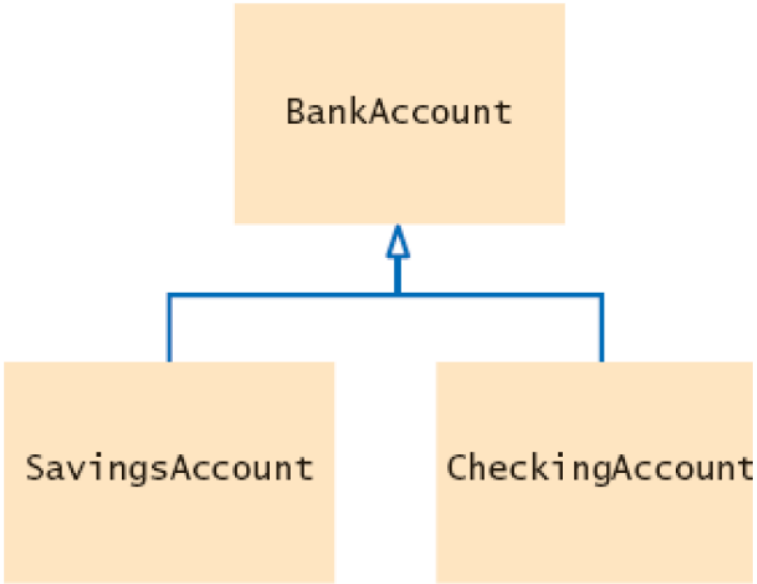
Nội dung

Nội dung

Kế thừa (*inheritance*)

Ví dụ

- **BankAccount**: tài khoản ngân hàng với các thuộc tính và phương thức như: *accountNumber*, *balance*; *getAccountNumber()*, *getBalance()*, *deposit(amount)*, *withdraw(amount)*, *tranfer(amount, amount)*
- **SavingAccount**: tài khoản có lãi
- **CheckingAccount**: tài khoản vãng lai



Kế thừa

- Kế thừa là một trong các đặc trưng chính của lập trình hướng đối tượng
- Ví dụ về kế thừa rất phổ biến: Quản lí nhân sự (lãnh đạo, các lớp nhân viên khác nhau), quản lí xe (xe tải, xe khách, xe hợp đồng, xe cá nhân v.v.), quản lí hàng hoá (các loại hàng hoá khác nhau), quản lí phòng khách sạn (các loại phòng khác nhau) v.v.
- Mục tiêu: Xây dựng lớp mới bằng cách sử dụng lại lớp đã có, mở rộng lớp này với các trường và phương thức mới
- Java: Chỉ có thể kế thừa được từ một lớp cha

Kết thừa (tiếp)

- Lớp được mở rộng (kế thừa): lớp cha (*superclass*)
  - BankAccount
- Lớp mở rộng: lớp con (*subclass*)
  - CheckingAccount: tài khoản vãng lai, không lãi suất, mỗi tháng được miễn phí một số nhỏ giao dịch, còn lại phải trả phí
  - SavingAccount: tài khoản tiết kiệm, có lãi hàng tháng
- Kế thừa phương thức
  - Mọi loại tài khoản đều chung phương thức *getBalance()*, *getAccountNumber()*
  - Mọi tài khoản đều có *deposit* và *withdraw*, tuy nhiên phương thức thực hiện khác nhau $\mapsto$ ghi đè
  - Tài khoản vãng lai cần phương thức *deductFees*, tài khoản tiết kiệm cần phương thức *addInterest* $\mapsto$ tạo mới

Xây dựng lớp kế thừa

- Cú pháp

```
class SubclassName extends SuperclassName{
   methods
   instance fields
}
```

Cây kế thừa

- Mọi lớp đều kế thừa (trực tiếp hoặc gián tiếp) lớp **Object**

Kiểm soát truy cập

	class	package	subclass	world
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Tóm tắt về kế thừa

- **class A extends B...** $\mapsto$ A là lớp con của B
- A có tất cả các trường và phương thức của B
- A có thể có các trường và phương thức riêng
- A có thể cài đặt lại một phương thức của lớp cha (ghi đè)
- Gọi phương thức của lớp cha

```
super.method(args)
```

- Gọi hàm dựng của lớp cha

```
super(args)
```

- A chỉ có thể kế thừa từ một lớp cha
- Khi gọi 1 phương thức của 1 đối tượng A, Java tìm cài đặt của phương thức đó trong A, nếu không sẽ tìm tới lớp cha của A, ...

Tóm tắt đặc điểm lập trình hướng đối tượng

- Trừu tượng dữ liệu (*abstraction*): chương trình chỉ quan tâm đến các đặc tính của đối tượng (gồm có gì, làm gì), bỏ qua các tiểu tiết (làm như thế nào).
- Đóng gói dữ liệu (*encapsulation*): Cơ chế đảm bảo môi trường bên ngoài chỉ có thể tác động vào đối tượng thông qua các dịch vụ (phương thức) cho bởi người viết mã chương trình $\Rightarrow$ tính toàn vẹn dữ liệu.
- Tính đa hình (*polymorphism*): cùng một thông báo, mỗi đối tượng "phản ứng" khác nhau.
- Kế thừa (*inheritance*): Cho phép đối tượng này lấy lại (kế thừa) những đặc tính sẵn có của đối tượng khác. (Lưu ý, một số ngôn ngữ cho phép kế thừa bội. Trong Java điều này cũng có thể thực hiện được thông qua cơ chế interface)

Nội dung

Ngoại lệ

- Là sự kiện xảy ra ngoài mong đợi
  - null.someMethod();
  - (new int[1])[1] = 0;
  - int i = "string";
  - *DivideByZeroNoExceptionHandling.java*
- Lớp Exception: cho phép báo lỗi
  - Tạo đối tượng Exception
  - Bổ sung 1 số thông tin về ngoại lệ
  - Ném (throw) đối tượng Exception
- Khi ngoại lệ xảy ra, phương thức kết thúc ngay, chương trình chuyển điều khiển cho bộ xử lí ngoại lệ
- *DivideByZeroWithExceptionHandling.java*

public class Exception

- Exception là một lớp
- Có thể tự tạo lớp ngoại lệ bằng cách kế thừa từ lớp Exception

```
public class MyException extends Exception{...}
```

- Viết phương thức ném ngoại lệ
  - Dùng **throws** sau khai báo tên phương thức để cảnh báo rằng phương thức này có thể ném ngoại lệ
  - Dùng **throw** trong phương thức khi cần ném ngoại lệ

```
public Object get(int index) throws
  ArrayOutOfBoundsException{
    if(index <0 || index > size())
      throw new ArrayOutOfBoundsException(""
          + index);
  }
}
```

Ném và xử lí ngoại lệ

- Ném ngoại lệ
  - Cú pháp: *throw exceptionObject;*
- Xử lí ngoại lệ
  - Bắt ngoại lệ (*catch*)
  - Tiếp tục ném ngoại lệ (*throws*)

Bắt ngoại lệ

- Khối lệnh

```
try {
.....
} catch(ExceptionClass error){
  ......
}
```

- Ví dụ
- *DivideByZeroWithExceptionHandling.java*

```
try {
   get(-1);
} catch(ArrayOutOfBoundsException error){
   System.out.println("Error!");
}
```

Tiếp tục ném ngoại lệ

- Ví dụ

```
void doBad() throws ArrayOutOfBoundsException {
  get(-1);
}
```

Mệnh đề finally

- Dùng cho đoạn mã cần được thực hiện trong mọi tình huống có lỗi cũng như không có lỗi
- Cú pháp:

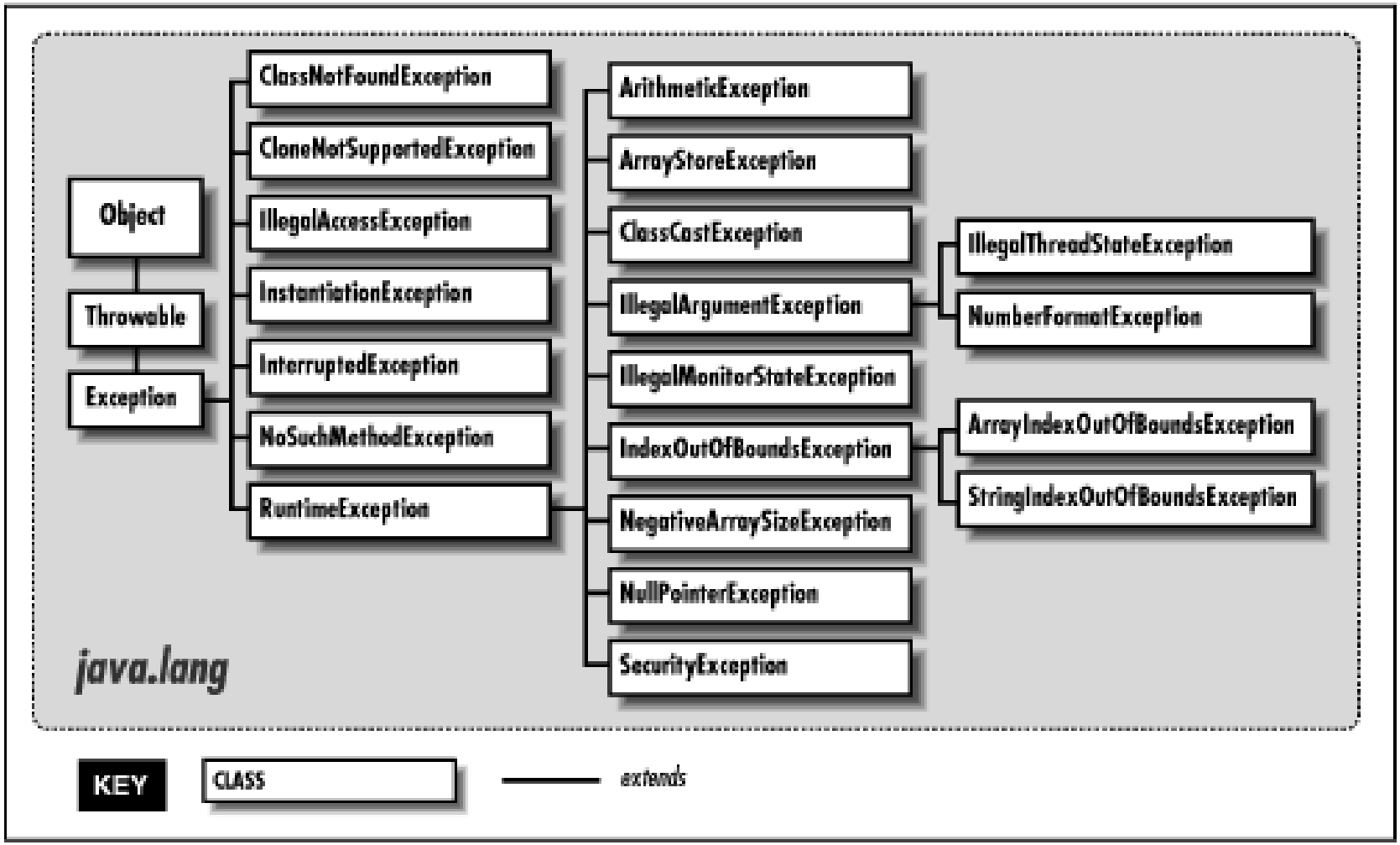
```
try {
  ...
} finally{
  ...
}
```

- Ví dụ: *UsingExceptions.java*

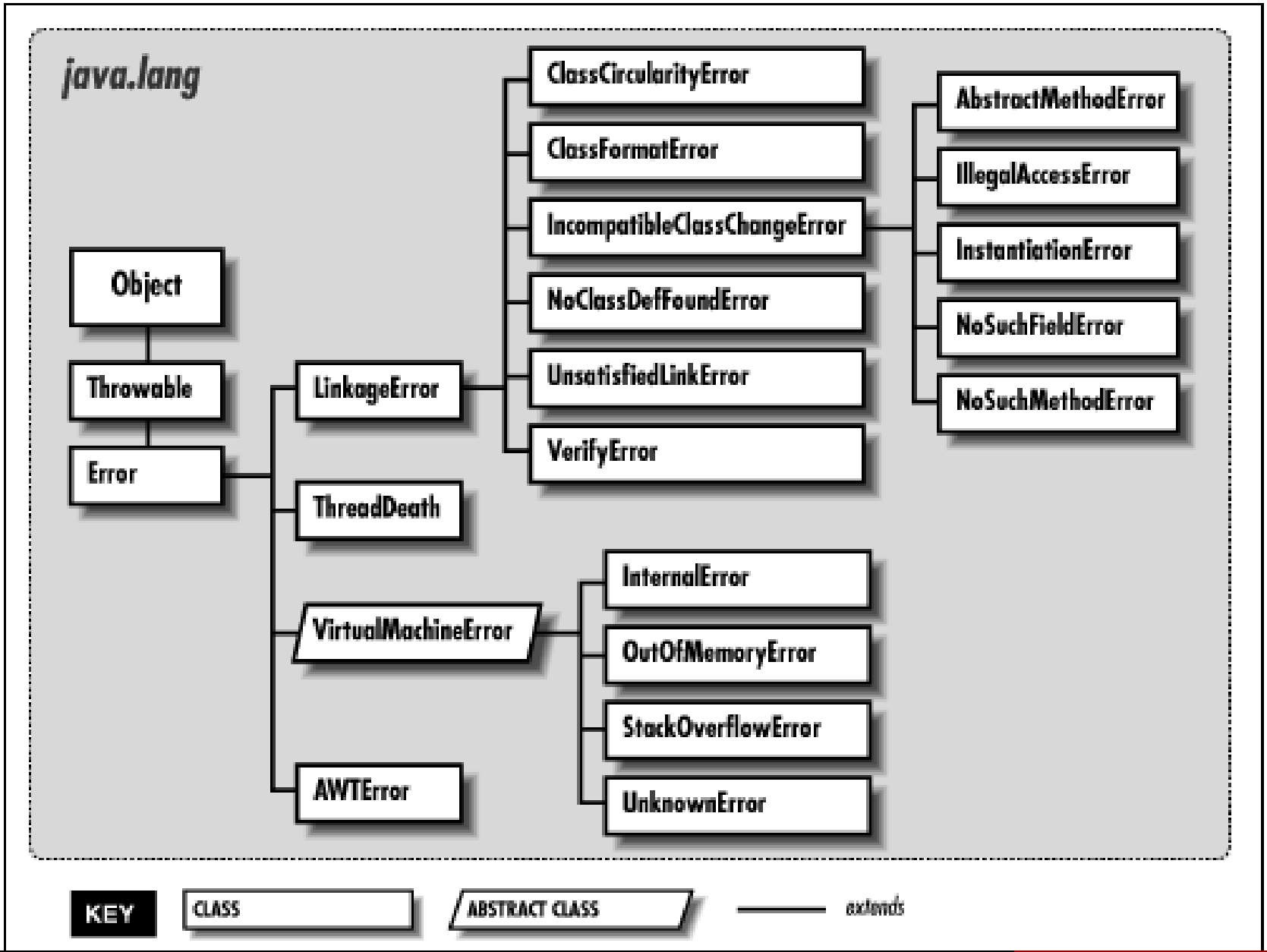
Phân cấp các lớp ngoại lệ

- Lỗi: error
- Ngoại lệ không kiểm soát: Runtime Exception
- Ngoại lệ có kiểm soát: bắt buộc phải xử lí
- Throwable: getMessage, getStackTrace (*UsingExceptions2.java*)

Cây kế thừa các lớp ngoại lệ - Exception



Cây kế thừa các lớp lỗi - Error



Bài tập

- Sửa chương trình *Addition.java* để xử lí lỗi khi các dữ liệu nhập vào không hợp lệ