THCS 4: LẬP TRÌNH CƠ BẢN VỚI JAVA

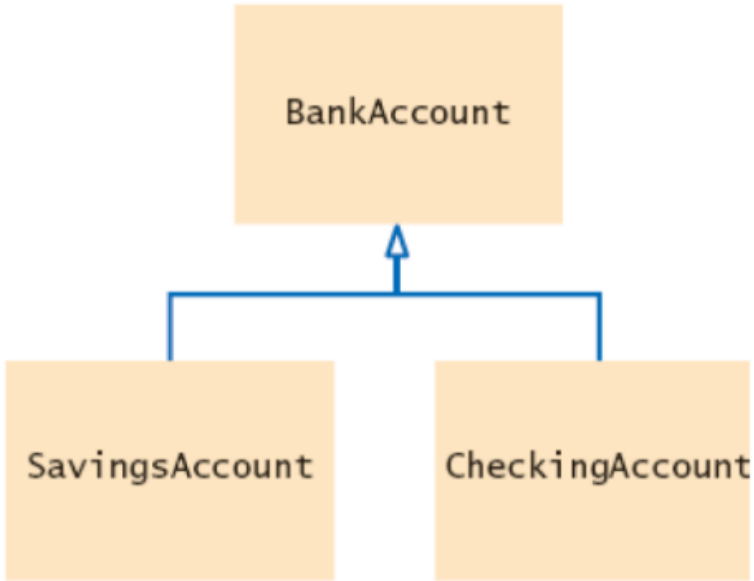
KẾ THỪA - XỬ LÝ NGOẠI LỆ

Nguyễn Thị Tâm
nguyenthitam.hus@gmail.com

Ngày 26 tháng 11 năm 2024

Kế thừa (inheritance)

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



Kế thừa (1)

- 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

Kế thừa (2)

- 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 $\rightarrow$ ghi đè
  - Tài khoản vãng lai cần phương thức deductFees, tài khoản tiết kiệm cần phương thức addInterest $\rightarrow$ tạo mới

Xây dựng lớp kế thừa

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

Tóm tắt về kế thừa

- **class A extends B**: 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 làm 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,...

Kiểm soát quyền 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

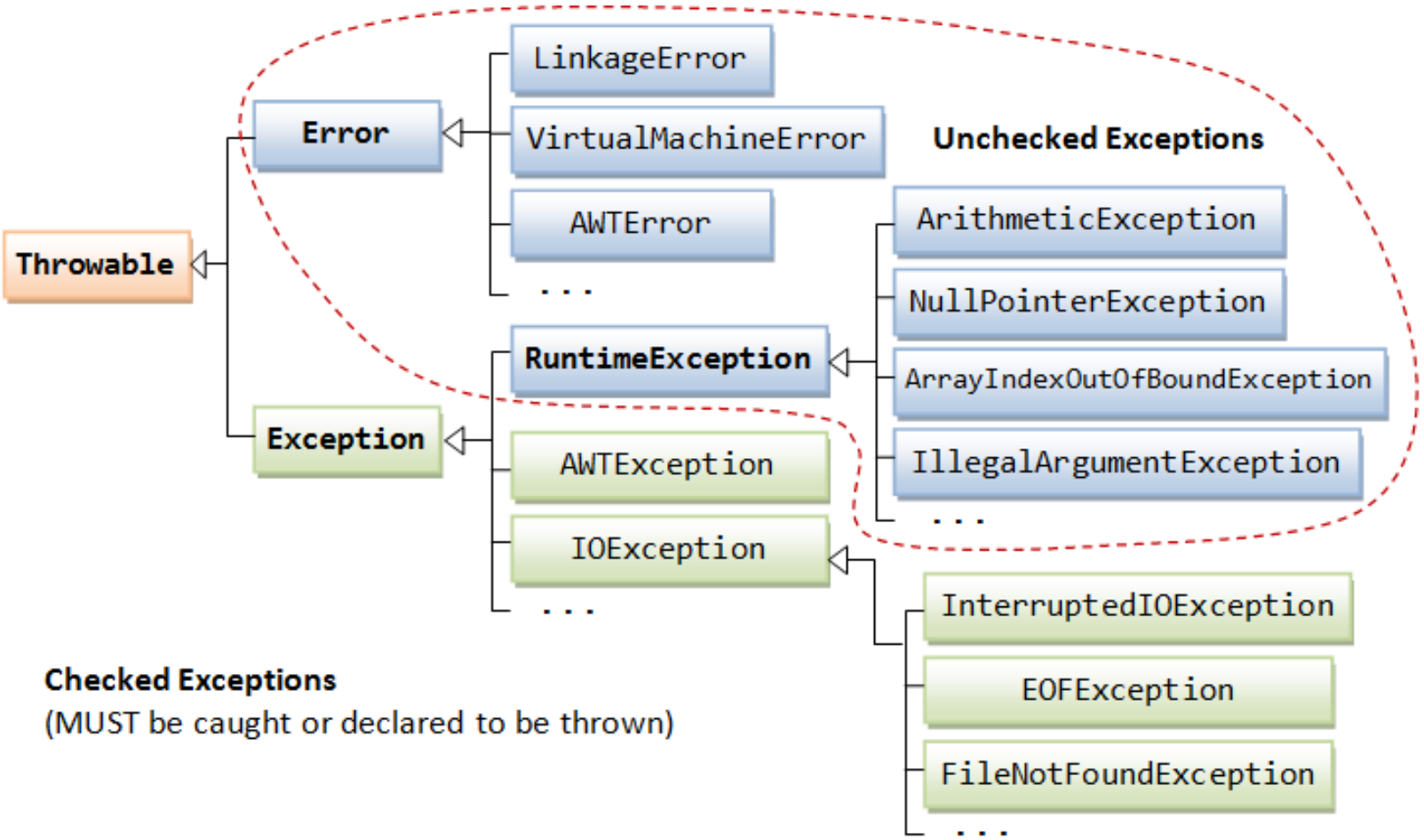
Tóm tắt đặc điểm của lập trình hướng đối tượng

- 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 => 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)

Ngoại lệ

- Là sự kiện xảy ra ngoài mong đợi
- Lớp Exception: cho phép báo lỗi
- Khi ngoại lệ xảy ra, phương thức kết thúc ngay, chương trình chuyển điều khiển cho bộ xử lý ngoại lệ

Phân cấp ngoại lệ



Phân loại ngoại lệ

- Checked Exceptions:
  - Ngoại lệ xảy ra trong lúc compile time, nó cũng có thể được gọi là **compile time exceptions**. Loại exception này không thể bỏ qua trong quá trình compile, bắt buộc phải xử lý
  - Ví dụ: IOException, FileNotFoundException, NoSuchFieldException,...
- Unchecked Exceptions:
  - Ngoại lệ xảy ra tại thời điểm thực thi chương trình, được gọi là **runtime exceptions**.
  - Các lớp kế thừa từ RuntimeException được gọi là unchecked exception.
  - Ví dụ: NullPointerException, NumberFormatException, ArrayIndexOutOfBoundsException, DivideByZeroException

Ví dụ

Checked Exception

```
import java.io.FileReader;
public class ExceptionExample1{
    public static void main (String []args){
        FileReader reder = new FileReader ("data.txt");
        // Unhandled exception type FileNotFoundException
    }
}
```

Unchecked Exception

```
import java.io.FileReader;
public class ExceptionExample1{
    public static void main (String []args){
       String str = null;
       System.out.println(str.length());
    }
}
```

public class Exception

- Exception là một lớp
- Có thể tự tạo ra 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);
    }
}
```

Xử lý ngoại lệ

Khối lệnh try - catch

- Khối lệnh

```
try{
    // code co the nem ra ngoai le
}catch(ExceptionClass e){
    // code xu ly ngoai le
}
```

- Không có ngoại lệ xử lý

```
public class TestTryCatch1 {
    public static void main(String args[]) {
        int data = 50 / 0;
        // rest of the code
    }
}
```

Xử lý ngoại lệ

Khối lệnh try - catch

- Giải quyết bằng xử lý ngoại lệ

```
public class TestTryCatch2 {
    public static void main(String args[]) {
        try {
            int data = 50 / 0;
        } catch (ArithmeticException e) {
            System.out.println(e);
        }
        // rest of the code
    }
}
```

Xử lý ngoại lệ

Nhiều khối lệnh try - catch

- Cú pháp

```
try{
    // code co the nem ra ngoai le
}catch(ExceptionClass e1){
    // code xu ly ngoai le
}catch(ExceptionClass e2){
    // code xu ly ngoai le
}
```

- Quy tắc
  - Tại một thời điểm chỉ xảy ra một ngoại lệ và tại một thời điểm chỉ có một khối catch được thực thi
  - Các khối catch phải được sắp xếp từ cụ thể nhất tức chung nhất. Ví dụ, phải khai báo khối lệnh catch để xử lý lỗi **ArithmeticException** trước khi khai báo catch để xử lý lỗi **Exception**

Xử lý ngoại lệ

Nhiều khối lệnh try - catch

```
public class TestTryCatch3 {
   public static void main(String args[]) {
       try {
           int a[] = new int[5];
           a[5] = 30 / 0;
       } catch (ArithmeticException e) {
           System.out.println("task1 is completed");
       } catch (ArrayIndexOutOfBoundsException e) {
           System.out.println("task 2 completed");
       } catch (Exception e) {
           System.out.println("common task completed");
       }
       // rest of the code
   }
}
```

Mệnh đề finally

- Khối lệnh **finally** trong java được sử dụng để thực thi các lệnh quan trọng như đóng kết nối, đóng các stream,...
- Khối lệnh **finally** trong java luôn được thực thi cho dù có ngoại lệ xảy ra hay không hoặc gặp lệnh **return** trong khối **try**.
- Khối lệnh **finally** trong java được khai báo sau khối lệnh **try** hoặc sau khối lệnh **catch**.

Mệnh đề finally

- Cú pháp

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

Từ khoá throw

- Được sử dụng để ném ra một ngoại lệ cụ thể
- Cú pháp

```
throw exceptions;
```

- Ví dụ

```
throw new IOexception ("File is not found");
```

Từ khoá throw

Ví dụ

```
public class TestThrow1 {
    static void validateAge(int age) {
        if (age < 18)
            throw new ArithmeticException("not valid");
        else
            System.out.println("valid");
    }

    public static void main(String args[]) {
        validateAge(13);
        // rest of the code
    }
}
```

Từ khoá throw

Ví dụ ném ngoại lệ và có xử lý

```
public class TestThrow2 {
    static void validateAge(int age) {
        try {
            if (age < 18)
                throw new ArithmeticException("not valid");
            else
                System.out.println("valid");
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }

    public static void main(String args[]) {
        validateAge(16);
       // rest of the code
    }
}
```

Từ khoá throws

- Được sử dụng để khai báo một ngoại lệ.
- Cú pháp

```
typeMethod name_method(args) throws exception_class_name{
...
}
```

- Lợi ích của **throws**
  - Ngoại lệ được ném ra ngoài và được xử lý ở một hàm khác
  - Cung cấp thông tin cho lời gọi phương thức về các ngoại lệ
- Quy tắc
  - Nếu đang gọi một phương thức throws một ngoại lệ, phải bắt hoặc throws ngoại lệ đó

Từ khoá throws

Trường hợp 1: Đã bắt ngoại lệ

Trường hợp này, ngoại lệ đã được xử lý bằng khối lệnh try - catch: code sẽ được thực thi dù ngoại lệ có xuất hiện trong chương trình hay không

---

```
import.java.io.IOException;
public class TestThrows2{
    static void method () throws IOException{
        throw new IOException ("IO Exception");
    }
    public static void main (String []args){
        try{
            method();
        }catch(Exception e){
            System.out.println("Done");
        }
        System.out.println("Continue");
    }
}
// Done
//Continue
```

Từ khoá throws

Trường hợp 2: Khai báo throws ngoại lệ

- Nếu ngoại lệ không xảy ra, code sẽ thực hiện tốt
- Nếu ngoại lệ xảy ra, một ngoại lệ sẽ được ném ra tại runtime vì throws không xử lý ngoại lệ đó

```
import.java.io.IOException;
public class TestThrows2{
    static void method () throws IOException{
        System.out.println("Ok");
    }
    public static void main (String []args) throws IOException{
        method();
        System.out.println("Continue");
    }
}
```

Ví dụ DivideByZeroWithExceptionHandling.java (1)

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class DivideByZeroWithExceptionHandling {
    public static int quotient( int numerator, int denominator ) throws
         ArithmeticException{
        return numerator / denominator;
    }
    public static void main( String args[] ) {
        Scanner scanner = new Scanner( System.in );
        boolean continueLoop = true;
        do {
         try {
              System.out.print( "Nhap so bi chia nguyen: " );
              int numerator = scanner.nextInt();
              System.out.print( "Nhap so chia nguyen: " );
              int denominator = scanner.nextInt();
```

Ví dụ DivideByZeroWithExceptionHandling.java (2)

```
            int result = quotient( numerator, denominator );
            System.out.printf( "\nKet qua: %d / %d = %d\n", numerator,
                denominator, result );
            continueLoop = false;
         }
      catch(InputMismatchException inputMismatchException) {
            System.err.printf( "\nNgoai le: %s\n",
               inputMismatchException );
            scanner.nextLine();
            System.out.println(
                "Du lieu khong hop le. Xin moi nhap lai so nguyen.\n" );
           }
      catch (ArithmeticException arithmeticException ) {
            System.err.printf( "\nNgoai le: %s\n", arithmeticException );
             System.out.println(
                "So chia phai khac 0. Xin moi nhap lai.\n" );
           }} while ( continueLoop );
   }
}
```

Ví dụ UsingException.java (1)

```
    public class UsingExceptions {
        public static void throwException() throws Exception {
            try {
               System.out.println("Method throwException" );
               throw new Exception();
            }catch (RuntimeException runtimeException){
               System.err.println(
                  "Exception handled in method throwException" );
            } // end catch
            finally{
               System.err.println( "Finally is always executed" );
            } // end finally
        } // end method throwException
} // end
```

Ví dụ UsingException.java (2)

```
public class UsingExceptions {
    public static void main( String args[] ) {
        try {
           throwException();
        } catch ( Exception exception ){
           System.err.println( "Exception handled in main" );
        } // end catch
    } // end main
```