

# Chương 4

## Generic và Xử lý ngoại lệ

**Giáo viên**

: ThS. Trần Văn Thọ

**Đơn vị**

: Bộ môn KTHT & MMT

# 1. Generics

- Làm thế nào để một cấu trúc chương trình (hàm, lớp) có thể đáp ứng được với nhiều kiểu dữ liệu khác nhau?  
VD: hàm so sánh, danh sách, ngăn xếp, hàng đợi...
- Giải pháp 1: sử dụng kiểu dữ liệu Object để đại diện cho tất cả các kiểu dữ liệu (trừ kiểu cơ bản)
  - Nhược điểm: không thể kiểm tra hay xác định lớp ban đầu khi sử dụng đối tượng Object
- Giải pháp 2: sử dụng khả năng tham số hóa kiểu dữ liệu cho hàm hoặc lớp
  - Coi kiểu dữ liệu cũng là một loại tham số khi sử dụng hàm hay lớp
    - ✓ Tham số hóa kiểu dữ liệu với hàm: **Generic Methods**
    - ✓ Tham số hóa kiểu dữ liệu với toàn bộ lớp: **Generic Types (Generic Classes)**

# 1.1 Generic Classes

## Khai báo các tham số dành cho kiểu dữ liệu:

- Mỗi tham số đại diện cho một kiểu dữ liệu (trừ kiểu cơ bản) sẽ được sử dụng bên trong lớp -> tham số kiểu
- Cú pháp:
  - `class name<T1, T2, ..., Tn> { ... }`  
T1, T2... là tên các tham số kiểu
  - Quy ước: sử dụng ký tự viết hoa và tuân theo thông lệ chung.  
VD: E- kiểu phần tử, T – kiểu lớp, N – kiểu số...
- Các tham số kiểu được sử dụng ở bất kỳ đâu bên trong lớp để: khai báo thuộc tính, khai báo biến, khai báo tham số hình thức cho hàm, làm kiểu trả về cho hàm
  - **Không thể khởi tạo giá trị cho đối tượng có kiểu là tham số kiểu**

# 1.1 Generic Classes

**VD:**

```
public class Box<T> {
```

Khai báo tham số kiểu

```
    private T t;
```

Sử dụng khai báo thuộc tính

```
    public void set(T t) {
```

```
        this.t = t;
```

Sử dụng khai báo tham số hình thức

```
    }
```

```
    public T get() {
```

```
        return t;
```

Sử dụng làm kiểu trả về cho hàm

```
    }
```

```
}
```

# 1.1 Generic Classes

- **Sử dụng lớp có tham số kiểu dữ liệu:**
  - Khi khai báo đối tượng thuộc lớp có tham số kiểu, cần xác định kiểu dữ liệu cụ thể thực tế mà lớp sử dụng thay cho tham số kiểu
    - Cú pháp:  
tên\_lớp<kiểu dữ liệu cụ thể> đối\_tượng;
    - VD: `Box<Integer> box;`
  - Khi khởi tạo đối tượng, tham số kiểu có thể được ẩn đi (Java 7 trở lên)
    - VD: `box=new Box<>();`
    - Hoặc: `box=new Box<Integer>();`
  - Chuyện gì xảy ra nếu không truyền kiểu dữ liệu khi khai báo?
    - Các tham số kiểu tự động được chuyển thành kiểu Object

# 1.1 Generic Classes

- VD

```
class Box<T> {  
    private T t;  
    public void set(T t) {  
        this.t = t;  
    }  
    public T get() {  
        return t;  
    }  
}  
  
class Cat{  
    public void say{  
        System.out.print("Meo");  
    }  
}
```

```
Box box=new Box();  
box.set(new Cat());  
Box.get().say();
```

```
Box<Cat> box=new Box<>();  
box.set(new Cat());  
box.get().say(); //OK
```

# 1.1 Generic Classes

## Thừa kế lớp có tham số kiểu:

- Lớp có tham số kiểu có thể được thừa kế (extends) hay thực thi (implements) giống như bất kỳ lớp bình thường nào khác
- Lớp dẫn xuất có thể cụ thể hóa tham số kiểu ở lớp cha hoặc mở rộng tham số kiểu ở lớp cha:

- VD cụ thể hóa:

```
class ProBox extends Box<Cat>{...}
```

- VD mở rộng:

```
class DoubleBox<E, T> extends Box<T>{...}
```

## 1.2 Generic Methods

- Hàm có thể có cơ chế tham số kiểu riêng
- Khai báo tham số kiểu cho hàm :

- Cú pháp:

`<T1, T2, ...> kiểu_trả_về tên_hàm (danh_sách_tham_số)  
{...}`

- Tham số kiểu của hàm có thể được dùng để đại diện cho kiểu trả về của hàm hoặc kiểu của tham số hàm.
- Tham số kiểu có thể được ẩn đi trong lời gọi hàm



# 1.2 Generic Methods

```
public class Util {  
    public static <T> boolean compare(T p1, T p2) {  
        return p1.equals(p2);  
    }  
}
```

```
Integer a=6;  
Integer b=6;  
System.out.print(Util.<Integer>compare(a,b));
```

```
Integer a=6;  
Integer b=6;  
System.out.print(Util.compare(a,b));
```

## 1.3 Giới hạn tham số kiểu

- Tham số kiểu có thể được giới hạn trong phạm vi dẫn xuất của các lớp cụ thể nào đó cho trước
  - Cho phép quản lý các kiểu mà lớp hay hàm tham số kiểu có thể nhận
- Sử dụng từ khóa **extends** để chỉ rõ tham số kiểu phải là dẫn xuất hoặc thực thi của các lớp cho trước nào đó
  - Có thể liệt kê nhiều lớp cơ sở bằng ký tự &
  - VD :

```
Class A { /* ... */ }  
interface B { /* ... */ }  
interface C { /* ... */ }
```

```
class D <T extends A & B & C> { /* ... */ }
```

## 1.3 Giới hạn tham số kiểu

- VD

```
class Mamal{
    public void run(){
        System.out.print("running...");
    }
}
class Cat extends Mamal{ }
class MamalBox<T extends Mamal>{
    private T t;
    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

```
MamalBox<Cat> boxcat=new MamalBox<>();
boxcat.set(new Cat());
boxcat.get().run();
```

## 2. Xử lý các tình huống bất thường

### **Đặt vấn đề:**

- Các tình huống bất thường có thể xảy ra từ rất nhiều khả năng. Nếu cố gắng đương đầu với nó, chúng ta sẽ làm cho chương trình trở nên cực kỳ phức tạp.
- Giải pháp: cứ viết chương trình như bình thường, sau đó dựa vào cơ chế quản lý ngoại lệ để giải quyết nốt với tất cả các tình huống còn lại.

## 2. Xử lý các tình huống bất thường

### 2 phương pháp xử lý bất thường khác:

- Xử lý trước:
  - Kiểm tra chặt chẽ tham số đầu vào trước khi thực hiện xử lý nhằm tránh các lỗi bất thường có thể xảy ra.
  - Chỉ hiệu quả với các trường hợp dữ liệu đơn giản
  - Trong nhiều trường hợp là rất khó, thậm chí không thể kiểm tra được tính hợp lệ của dữ liệu đầu vào.

## 2. Xử lý các tình huống bất thường

### 2 phương pháp xử lý bất thường:

- Xử lý sau:
  - Cứ để cho phương thức được thực hiện để xem kết quả thế nào. Sử dụng khi không sinh ra lỗi không thể khôi phục.
  - Chuyển một hàm có khả năng sinh lỗi thành 1 Lệnh (thủ tục) còn câu trả lời của Lệnh sẽ được lưu bởi thuộc tính.
  - Trong Java, có thể chuyển một hàm trả về giá trị thành hàm trả về kiểu void và kết quả của hàm sẽ lưu vào tham biến của hàm.

```
Integer StringToInteger(String input) {  
    ...  
    return num;  
}
```

```
void StringToInteger(Integer num, String input) {  
    ...  
}
```

Nguyên tắc: nên bày lỗi hơn là khôi phục khi lỗi đã xảy ra

## 2.1 Xử lý ngoại lệ

### Khái niệm ngoại lệ:

- Là tình huống bất thường xảy ra khi chương trình chạy (*runtime errors*) VD: chia cho không, truy nhập vào phần tử ở bên ngoài mảng, hoặc là cạn kiệt bộ nhớ.
- Các ngoại lệ này tồn tại ở bên ngoài hoạt động bình thường của chương trình và đòi hỏi chương trình phải xử lý ngay.



## 2.1 Xử lý ngoại lệ

- Vai trò xử lý ngoại lệ:

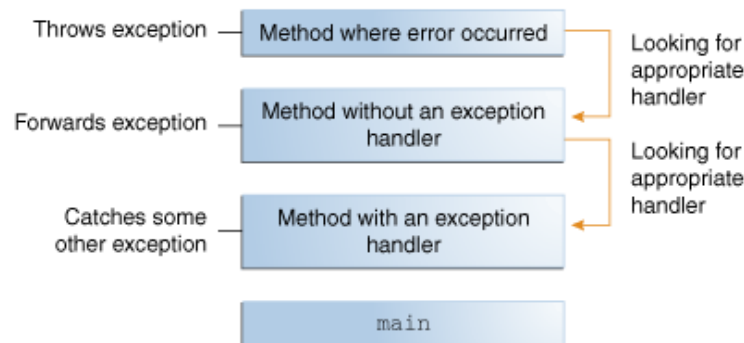
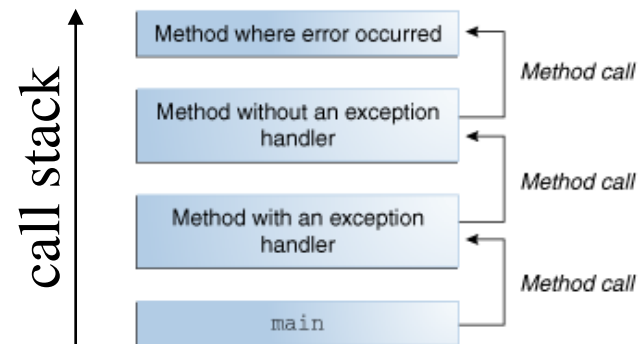
- Trong các trường hợp bình thường, 2 phương pháp xử lý trước và sau là đủ để đối phó với các ngoại lệ.
- Tuy nhiên trong 1 số trường hợp, 2 cách trên là không đủ:
  - Trường hợp không thể khôi phục lại được do chương trình bị dừng ngay lập tức khi xảy ra ngoại lệ
  - Trường hợp mà hoạt động của chương trình cần thực thi cho đến khi hoàn thành bất chấp các lỗi có thể xảy ra, nếu không sẽ gây ra các hậu quả tai hại.
  - Khi chương trình cần sự ổn định, khả năng chịu lỗi tốt

## 2.1 Xử lý ngoại lệ

- Xử lý ngoại lệ: là cơ chế cho phép hai đơn vị chương trình có thể trao đổi thông tin với nhau khi một ngoại lệ xảy ra
- 2 Bước:
  - ✓ Phát sinh ngoại lệ
  - ✓ Bẫy ngoại lệ

## 2.1 Xử lý ngoại lệ

- Khi lỗi xảy ra ở một hàm, một đối tượng ngoại lệ được sinh ra và được hệ thống thực thi xử lý
- Hệ thống thực thi sẽ “tìm kiếm” đoạn code xử lý ngoại lệ (exception handler) theo thứ tự hàm được gọi theo call stack, kể từ hàm phát sinh lỗi.
- Khi một exception handler phù hợp được tìm thấy, ngoại lệ sẽ được trao cho nó để giải quyết.
- Nếu không tìm thấy exception handler nào phù hợp, chương trình sẽ bị dừng

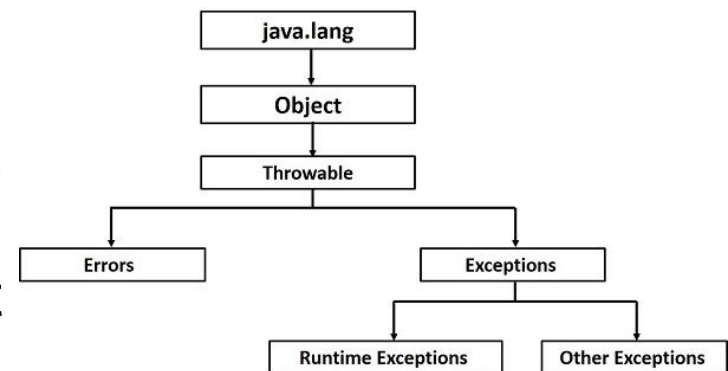


## 2.2 Phát sinh ngoại lệ

- Khi ngoại lệ xảy ra, đoạn chương trình phát hiện ngoại lệ sẽ sinh ra (ném) ra một ngoại lệ.
- Cú pháp ném ra một ngoại lệ:  
`throw đối_tượngngoại_lệ;`
- Đối tượng ngoại lệ có thể thuộc kiểu ngoại lệ có sẵn hoặc tự định nghĩa
- Kiểu ngoại lệ phải là dẫn xuất của lớp **Throwable**
- Các câu lệnh sau throw sẽ bị bỏ qua, chương trình tiếp tục tại phần bắt ngoại lệ (exception handler) hoặc dừng chương trình

## 2.2 Phát sinh ngoại lệ

- Các phương thức của Throwable
  - ✓ `getMessage()`: trả về mô tả lỗi
  - ✓ `toString()`: trả về tên lớp và mô tả lỗi
  - ✓ `printStackTrace()`: in ra kết quả hàm `toString()` dọc theo danh sách hàm trong stack
  - ✓ ...
- Thực tế, người ta hay tạo lớp ngoại lệ thừa kế từ lớp `Exceptions`



## 2.2 Phát sinh ngoại lệ

- Hàm phát sinh ngoại lệ phải được khai báo kiểu ngoại lệ mà nó ném ra bằng từ khóa throws:
  - Cú pháp:  
`Kiểu_tên_hàm (tham_số) throws  
danh_sách_kiểu_ngoại_lệ {...}`
  - Các kiểu ngoại lệ được viết cách nhau bởi dấu phẩy
  - Lưu ý: không khai báo kiểu ngoại lệ mà hàm không ném ra

## 2.2 Phát sinh ngoại lệ

VD:

```
class InsufficientFundsException extends Exception {  
    private double amount;  
    public InsufficientFundsException(double amount) {this.amount = amount;}  
    public double getAmount() {return amount;}  
}  
class Account {  
    double balance;  
    public void checkAccount(String username) throws TimeoutException {  
        // Method implementation ....  
        //When connecting to server takes so long:  
        throw new TimeoutException();  
    }  
    public void withdraw(double amount) throws InsufficientFundsException {  
        if (amount>balance){  
            double needs = amount - balance;  
            throw new InsufficientFundsException(needs);  
        }  
        else{  
            balance-=amount;  
        }  
    }  
}
```

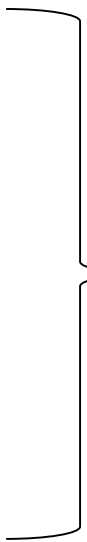
Lớp ngoại lệ có sẵn



## 2.3 Bẫy ngoại lệ

Cấu trúc try...catch dùng để bẫy ngoại lệ:

```
try {  
    // nhóm câu lệnh có khả năng sinh ra ngoại lệ  
    câu lệnh;  
}  
catch (ExceptionType1 e1) {  
    // câu lệnh để xử lý hướng ngoại lệ  
    câu lệnh;  
}  
...  
catch (ExceptionTypeN en) {  
    câu lệnh;  
}  
finally {  
    // The finally block always executes.  
}
```



Exception  
handler



## 2.3 Bẫy ngoại lệ

- Mệnh đề **catch** : được dùng để xử lý các ngoại lệ được sinh ra.
- Mỗi mệnh đề **catch** gồm ba thành phần:
  - Từ khóa **catch**,
  - Tham số : kiểu ngoại lệ
  - Các câu lệnh xử lý ngoại lệ nằm trong cặp ngoặc nhọn.
- Nếu mệnh đề **catch** nào được lựa chọn để xử lý ngoại lệ được sinh ra, nhóm câu lệnh của mệnh đề đó sẽ được thực thi.
- Lệnh trong mệnh đề **finally** luôn luôn được thực hiện, bất kể có lỗi hay không có lỗi ở khối lệnh **try**

## 2.3 Bẫy ngoại lệ

- Tham số của của mệnh đề catch:
  - ✓ catch (T e) dùng để “bắt” các ngoại lệ thuộc kiểu T và các kiểu dẫn xuất từ T
  - ✓ Lưu ý: có thể đặt T là Throwable hoặc Exception để bắt tất cả các ngoại lệ. Tuy vậy, nên bắt các ngoại lệ càng chi tiết càng tốt bằng nhiều mệnh đề **catch**
  - ✓ Tham số e chính là đối tượng ngoại lệ được ném ra bởi phần phát sinh ngoại lệ

## 2.3 Bẫy ngoại lệ - Ví dụ

```
public static void main(String[] args) {  
    try {  
        Account account=new Account();  
        account.balance=100;  
        account.checkAccount("An");  
        account.withdraw(1000);  
    } catch (TimeoutException ex) {  
        ex.printStackTrace();  
    } catch (InsufficientFundsException ex) {  
        System.out.println("Sorry, but you are short $" +  
ex.getAmount());  
        ex.printStackTrace();  
    } finally{  
        System.out.println("OK");  
    }  
  
}
```