# METHODS

**GV: Khuất Thuỳ Phương**
**Thực hiện:**
**Võ Văn Minh**
**Phạm Thị Kim Hiền**

# NỘI DUNG CHÍNH

Tìm hiểu thêm về các phương thức trong Java.

# Tham Trị – Tham Chiếu trong Java

- Trong java có 2 loại kiểu dữ liệu:
  – Kiểu tham trị : đối với các kiểu dữ liệu cơ bản: Số nguyên, số thực, kí tự,..=> tạo 1 bản sao của tham số đó và làm việc trên bản sao.
  – Kiểu tham chiếu : Kiểu mảng, Kiểu lớp đối tượng Class, Kiểu Interface,... => làm việc trực tiếp trên bản gốc của tham số.

- Do đó, khi truyền tham trị thì giá trị của biến không thay đổi, ngược lại thì giá trị sẽ thay đổi khi truyền tham chiếu.

# Một số lưu ý

- Object a = new Object("Object A");
  Object b = a;
  b.setNewName("Object B");
  **Khi đó Object a sẽ có name là “Object B”**
- **Để hoán vị giá trị a và b thì a và b phải là Object**

```
private void swap(Object a, Object b) {
        Object temp = a;
        a = b;
        b = temp;
}
```

# Các loại phương thức tham chiếu

- Ví dụ ta có class Car với các phương thức như sau:

```
public static class Car {
    public static Car create( final Supplier< Car > supplier ) {
        return supplier.get();
    }
    public static void collide( final Car car ) {
        System.out.println( "Collided " + car.toString() );
    }
    public void follow( final Car another ) {
        System.out.println( "Following the " + another.toString() );
    }
    public void repair() {
        System.out.println( "Repaired " + this.toString() );
    }
}
```

# Các loại phương thức tham chiếu

- Hàm dựng tham chiếu (hàm dựng này không có đối số)
  Cú pháp: *Class<T>::new*

```
final Car car = Car.create(Car::new);
final List<Car> cars = Arrays.asList(car);
```

- Tham chiếu tới phương thức static (phương thức này chấp nhận chính xác tham số của Car)
  Cú pháp: *Class::staticMethod*

```
cars.forEach(Car::collide);
```

# Các loại phương thức tham chiếu

- Tham chiếu tới phương thức instance của đối tượng tùy ý (phương thức này có thể chấp nhận không có đối số)
  Cú pháp *Class::method*

```
cars.forEach(Car::repair);
```

- Tham chiếu tới phương thức instance của lớp instance cụ thể (phương thức này chấp nhận chính xác tham số của Car)
  Cú pháp *instance::method*

```
final Car police = Car.create(Car::new);
cars.forEach(police::follow);
```

# Các loại phương thức tham chiếu

```
 7⊖     public static void main(String[] args) {
 8
 9         final Car car = Car.create(Car::new);
10         final List<Car> cars = Arrays.asList(car);
11
12         cars.forEach(Car::collide);
13
14         cars.forEach(Car::repair);
15
16         final Car police = Car.create(Car::new);
17         cars.forEach(police::follow);
18
19     }
```

Javadoc | Declaration | Console

```
<terminated> test [Java Application] C:\Program Files\Java\jre1.8.0_102\bin
Collided Car@87aac27
Repaired Car@87aac27
Following the Car@87aac27
```

# Phương thức finalize()

- Dùng để định nghĩa một phương thức mà sẽ được gọi ngay trước khi hủy một đối tượng bởi Garbage Collector.
- Bên trong phương thức finalize(), bạn sẽ xác định những hành động nào phải được thực hiện trước khi một đối tượng bị phá hủy.
- Phương thức finalize() có form chung là:

```
protected void finalize( )
{
        // tai day la phan code ket thuc
}
```

# Phương thức mặc định cho Interface

- Java 8 cho phép bạn thêm một method không trìu tượng vào interface bằng cách sử dụng từ khóa default. Các method này được hiểu như **các phương thức mở rộng**.

```
public interface Formula {

    // Khai báo một method trìu tượng.
    double calculate(int a);

    // Khai báo một method không trìu tượng.
    // Sử dụng từ khóa default.
    // (Hàm tính căn bậc 2 của một số)
    default double sqrt(int a) {
        return Math.sqrt(a);
    }

}
```

# Có 4 kiểu của java access modifiers:

- **Public:** phương thức có thể truy cập được từ bên ngoài lớp khai báo.
- **Protected:** có thể truy cập được từ lớp khai báo và
- những lớp dẫn xuất từ nó như qua tính kế thừa.
- **Private:** chỉ được truy cập bên trong bản thân lớp khai báo.
- **(Mặc định):** Phạm vi truy cập của là trong nội bộ package.

| Access Modifier | Truy cập bên trong class? | Truy cập bên trong package? | Truy cập bên ngoài package bởi class con? | Truy cập bên ngoài class và không thuộc class con? |
|---|---|---|---|---|
| private | Y | N | N | N |
| Mặc định | Y | Y | N | N |
| protected | Y | Y | Y | N |
| public | Y | Y | Y | Y |

# Một số lưu ý của access modifiers:

- Với Interface khi bạn khai báo một trường (Field) hoặc một phương thức (Method) ta luôn phải khai báo là public hoặc để mặc định, nhưng Java luôn hiểu đó là public.
- **protected access modifier** chỉ áp dụng cho field, method và constructor. Nó không thể áp dụng cho class (class, interface, enum, annotation).
- Ta có thể ghi đè một method của class cha với một method cùng tên cùng tham số tại class con, tuy nhiên bạn không được phép làm giảm phạm vi truy cập của method này (chiều tăng dần dần phạm vi truy cập là **Private** -> **Mặc định** -> **Protected -> Public**

# Các **non-access modifiers**

- **Final:** Phương thức không thể được thừa kế hoặc ghi đè. Ví dụ:

```
public class Test{
    public final void changeName(){
       // Phan than phuong thuc
    }
}
```

- **Static:** phương thức có thể được gọi mà không cần đến đối tượng. Nó chỉ được sử dụng đối với các dữ liệu và các phương thức tĩnh khác.

# Các **non-access modifiers**

- **Abstract:** phương thức không có code và nó sẽ được bổ sung ở các lớp con (subclass). Loại phương thức này được sử dụng trong các lớp kế thừa.

  Ví dụ:

```
public abstract class SuperClass{
    abstract void m(); //phuong thuc abstract
}

class SubClass extends SuperClass{
    // trien khai phuong thuc abstract
     void m(){
   .........
     }
}
```

# Các **non-access modifiers**

- **Native:** Chỉ ra rằng phần thân của phương thức được viết trên các ngôn ngữ khác Java ví dụ C, hoặc C++.
- **Synchronized:** Sử dụng với phương thức trong quá trình thực thi threads. Nó cho phép chỉ một thread được truy cập vào khối mã tại một thời điểm.
- **Volatile**: Được sử dụng với các biến để thông báo rằng giá trị của biến có thể được thay đổi vài lần khi thực thi chương trình và giá trị của nó không được đặt vào thanh ghi.

# Lỗi biên dịch

- Các phương thức thể hiện trong (a) là logic đúng, nhưng nó có một lỗi biên dịch vì trình biên dịch Java nghĩ rằng phương thức này có thể không trả về một giá trị.

Ví dụ:

```
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else if (n < 0)
        return –1;
}
```

(a)

should be

```
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else
        return –1;
}
```

(b)

# *Thank you for your time and attention!*

Sources:

- https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html
- http://o7planning.org/vi/10319/access-modifier-trong-java
- https://voer.edu.vn/c/phuong-thuc-trong-mot-lop-method/95eeacbe/d75e2109