FPT Education

FPT POLYTECHNIC

Lập trình Java 2

Bài 8: Generic

Phần 1

www.poly.edu.vn

Mục tiêu

- Khái niệm Genegics
- Ưu điểm Genegics
- Tạo class generic và method
- Giới hạn kiểu dữ liệu
- Các ký hiệu đại diện
- Generic method
- Generic Interface
- Một số hạn chế

KHÁI NIỆM GENERICS

- Generics là cách thức lập trình tổng quát cho phép một object hoạt động với nhiều kiểu dữ liệu khác nhau.
- Thuật ngữ "Generics" nghĩa là tham số hóa kiểu dữ liệu. Tham số hóa kiểu dữ liệu rất quan trọng vì nó cho phép chúng ta tạo ra và sử dụng một class, interface, method với nhiều kiểu dữ liệu khác nhau.
- Một class, interface hay một method mà thực hiện trên một kiểu tham số xác định thì gọi là generic.

Ví dụ: Sử dụng ArrayList với các kiểu dữ liệu khác nhau

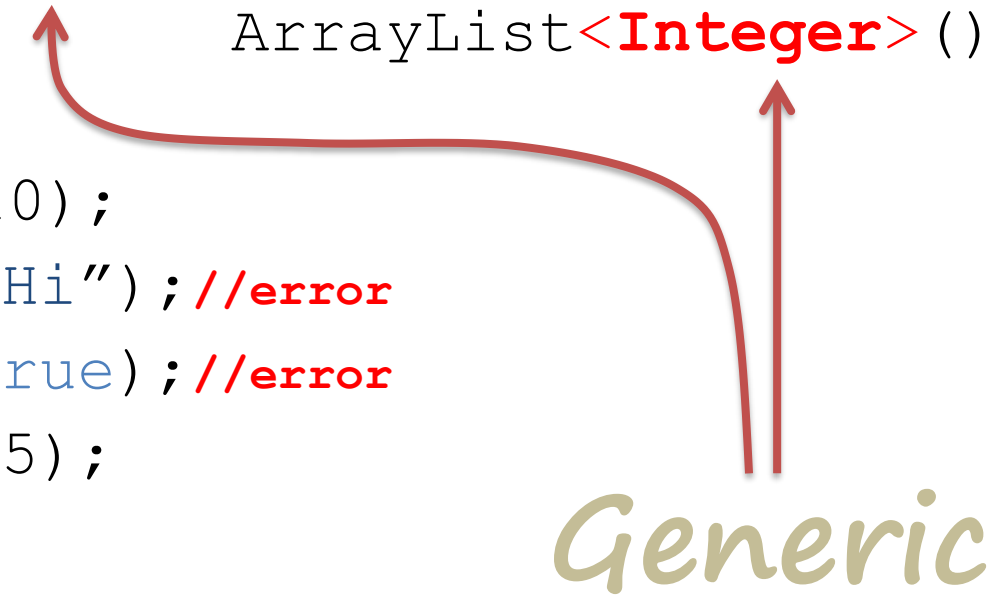
```
ArrayList mylist = new ArrayList();
mylist.add(10);
mylist.add("Hello");
mylist.add(true);
mylist.add(15.75);
```

Lấy ra:

```
int a = (Integer)mylist.get(0);
String str = (String)mylist.get(1);
```

Ví dụ: Sử dụng ArrayList với các kiểu dữ liệu Integer

```
ArrayList<Integer> mylist = new
                        ArrayList<Integer>();

mylist.add(10);
mylist.add("Hi");//error
mylist.add(true);//error
mylist.add(15);
```



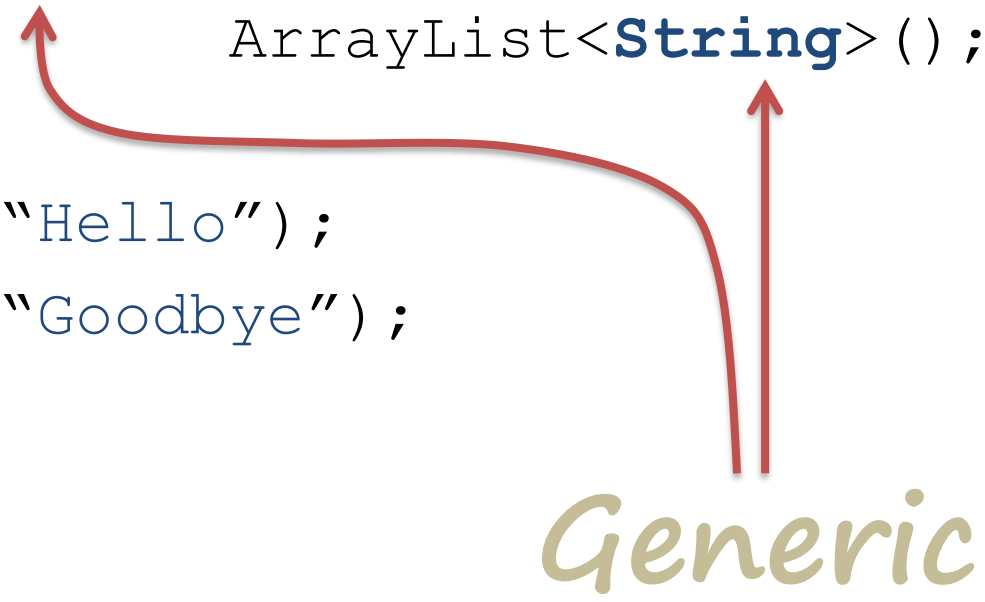
Lấy ra:

```
int a = mylist.get(0);
```

KHÁI NIỆM GENERICS

Ví dụ: Sử dụng ArrayList với các kiểu dữ liệu String

```
ArrayList<String>mylist = new
                          ArrayList<String>();

mylist.add("Hello");
mylist.add("Goodbye");
```



Lấy ra:

```
String str = mylist.get(0);
```

ƯU ĐIỂM CỦA GENERIC

- Kiểm tra kiểu dữ liệu trong thời điểm dịch
  - Trình biên dịch Java áp dụng việc kiểm tra đoạn mã generic để phát hiện các vấn đề như vi phạm an toàn kiểu dữ liệu. Việc sửa lỗi tại thời gian biên dịch dễ dàng hơn nhiều khi sửa chữa lỗi tại thời điểm chạy chương trình.

Compile error !

Ưu điểm của Generic

- Không cần ép kiểu dữ liệu
  - Đoạn code sau đây không dùng generic nên phải ép kiểu:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0); //phải ép kiểu
```

  - Khi dùng generic, không cần ép kiểu:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0); // không ép kiểu
```

ƯU ĐIỂM CỦA GENERIC

- Cho phép người lập trình viên thực hiện các thuật toán tổng quát.
  - Bằng cách sử dụng generics, người lập trình có thể thực hiện các thuật toán tổng quát với các kiểu dữ liệu tùy chọn khác nhau, và nội dung đoạn code trở nên rõ ràng và dễ hiểu.

PROGRAMS

TẠO CLASS GENERIC VÀ METHOD

- Quy ước đặt tên tham số kiểu cho Generics

Ký tự	Ý nghĩa
E	Element – phần tử
K	Key – khóa
V	Value – giá trị
T	Type – kiểu dữ liệu
N	Number – số

TẠO CLASS GENERIC

- Tạo generics class với 1 tham số kiểu:

```
public class GenericsType<T> {
	private T t;
	public T get() {
		return this.t;
	}
	public void set(T t1) {
		this.t = t1;
	}
}
```

SỬ DỤNG CLASS GENERIC

- Tạo generics class với 1 tham số kiểu:

```
public static void main(String args[]) {

    GenericsType<String> type1 = new GenericsType<String>();
    type1.set("Pankaj"); //hợp lệ
    System.out.println(type1);

    GenericsType type2 = new GenericsType(); //không generic
    type2.set("Pankaj"); //hợp lệ
    type2.set(10);       //hợp lệ
    System.out.println(type2);

}
```

TẠO CLASS GENERIC

- Tạo generics class với 2 tham số kiểu:

```
public class Pair<T, S> {
    private T first;
    private S second;

    public Pair(T fi, S se){
        first = fi;
        second = se;
    }
    public T getFirst() { return first; }
    public S getSecond() { return second; }
}
```

SỬ DỤNG CLASS GENERIC

- Tạo generics class với 2 tham số kiểu:

```
public class PairDemo{
  public static void main(String[] args){
    Pair<Integer, String> p1 = new Pair(1,"One");
    Pair<String, String> p2 = new Pair("Hello","World");
    System.out.println(p1.getFirst()+", "+p1.getSecond());
    System.out.println(p2.getFirst()+", "+p2.getSecond());
  }
}
```

FPT Education

FPT POLYTECHNIC

LẬP TRÌNH JAVA 2

BÀI 8: GENERIC

PHẦN 2

www.poly.edu.vn

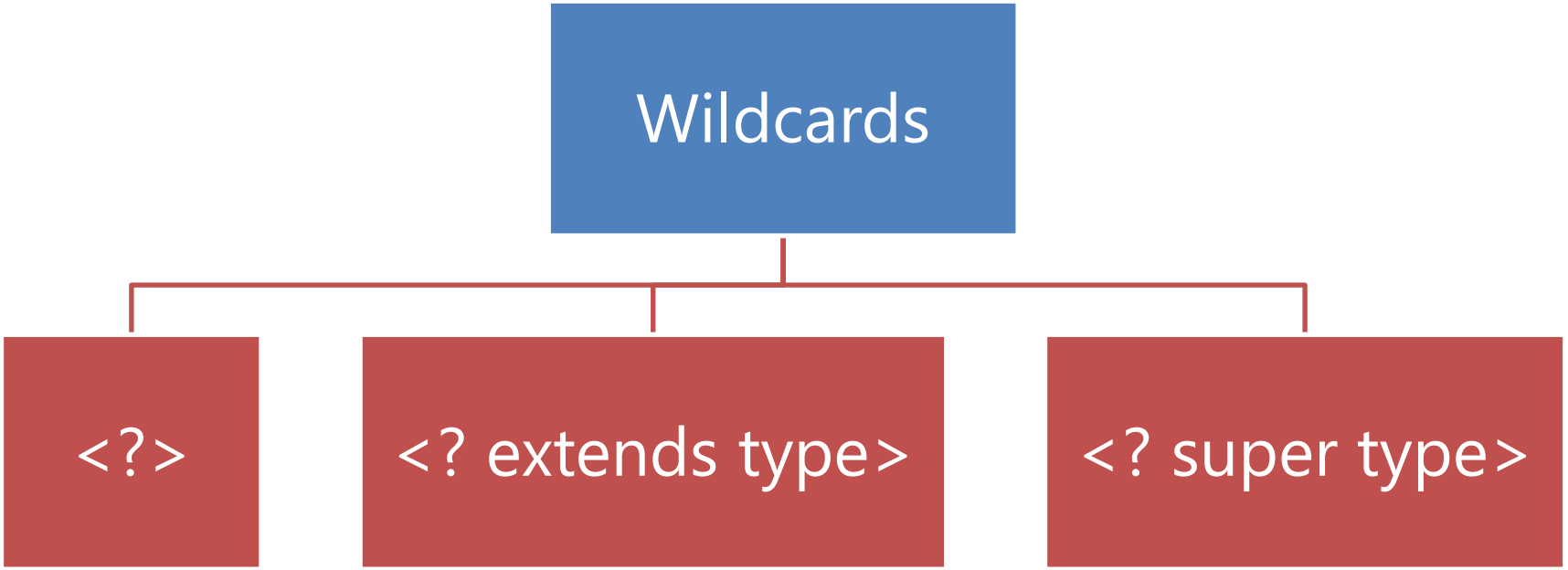
GIỚI HẠN KIỂU DỮ LIỆU

```
public class example<T extends Number> {
    private T number;

    example(T n) {
        number = n;
    }
    double reciprocal() {  //Lấy nghịch đảo
        return 1/number.doubleValue();
    }
    double fraction() {      //Lấy phần thập phân
        return number.doubleValue() - number.intValue();
    }
```

GIỚI HẠN KIỂU DỮ LIỆU

```
example<Integer> e1 = new example<Integer>(5);
System.out.println("Reciprocal:"+e1.reciprocal());
System.out.println("Fraction:"+e1.fraction());

example<Double> e2 = new example<Double>(7.5);
System.out.println("Reciprocal:"+e2.reciprocal());
System.out.println("Fraction:"+e2.fraction());

example<Float> e3 = new example<Float>(7.25f);
System.out.println("Reciprocal:"+e3.reciprocal());
System.out.println("Fraction:"+e3.fraction());
```

CÁC KÝ HIỆU ĐẠI DIỆN GENERIC



CÁC KÝ HIỆU ĐẠI DIỆN GENERICS

- Xét ví dụ sau:

```
boolean testequals(example<T> e){
    if (number.doubleValue()==e.number.doubleValue()){
        return true;
    }
    return false;
}
public static void main(String[] args) {
    example<Integer> e1 = new example<Integer>(5);
    example<Double> e2 = new example<Double>(5.0);

    System.out.println("e1==e2 ? "+e1.testequals(e2));
```

CÁC KÝ HIỆU ĐẠI DIỆN GENERICS

- Ký tự đại diện <?>

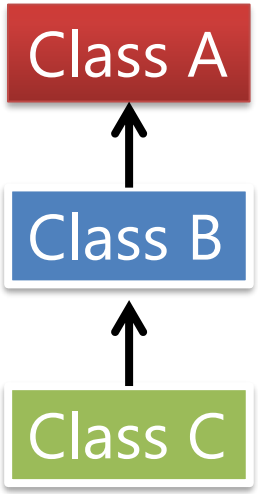
```
boolean testequals(example<?> e){
    if (number.doubleValue()==e.number.doubleValue()){
        return true;
    }
    return false;
}
public static void main(String[] args) {
    example<Integer> e1 = new example<Integer>(5);
    example<Double> e2 = new example<Double>(5.0);

    System.out.println("e1==e2 ? "+e1.testequals(e2));
```

CÁC KÝ HIỆU ĐẠI DIỆN GENERICS

- Ký tự đại diện <? extends type>

```
public void processEle(List<? extends A> elements){
    ...
}
```

Trong đó :




```
List<A> listA = new ArrayList<A>();
processEle(listA);

List<B> listB = new ArrayList<B>();
processEle(listB);

List<C> listC = new ArrayList<C>();
processEle(listC);
```

CÁC KÝ HIỆU ĐẠI DIỆN GENERICS

- Ký tự đại diện <? super type>

```
public void processEle(List<? super A> elements){
     for(A obj : elements){
       System.out.println(obj.getValue());
      }
}
```

```
List<A> listA = new ArrayList<A>();
processEle(listA);

List<Object> listO = new ArrayList<Object>();
processEle(listO);
```

GENERIC CONSTRUCTORS

```
public class Summation {
    private int sum;
    <T extends Number> Summation(T size) {
        sum = 0;
        for (int i = 0; i <= size.intValue(); i++) {
            sum += i;
        }
    }
    int getSum() {return sum;}
    public static void main(String[] args) {
        Summation obj = new Summation(4.0);
        System.out.println("Sum of 1->4 is " + obj.getSum());
    }
}
```

GENERIC INTERFACES

```
interface Containment<T>{
    boolean contains(T obj);
}
public class MyClass<T> implements Containment<T>{
    T[] arrayRef;
    public MyClass(T[] arr){
        arrayRef = arr;
    }
    public boolean contains(T obj) {
        for (T x: arrayRef)
            if (x.equals(obj)) return true;
        return false;
    }
}
```

GENERIC INTERFACES

```
public static void main(String[] args) {
    Integer x[]={1,2,3};
    MyClass<Integer> ob = new MyClass<Integer>(x);
    Integer a = 20;
        if (ob.contains(a)){
            System.out.println(a+" in array");
        }else
            System.out.println(a+" NOT in array");
    }
```

MỘT SỐ HẠN CHẾ CỦA GENERIC

- Không thể khởi tạo generic với dữ liệu kiểu nguyên thủy

```
Pair<int, char> p = new Pair<>(8, 'a');  //error
Pair<Integer, Character> p = new Pair<>(8, 'a');
```

- Không thể tạo instance cho kiểu dữ liệu

```
class Gen<T>{
      T obj;
      Gen(){
            obj= new T(); //Illegal
      }
}
```

MỘT SỐ HẠN CHẾ CỦA GENERIC

- Không thể là static trong class

```
class Gen<T>{
        static T obj;          //Kiểu T không thể là static
        static T getObj(){  //Phương thức không thể static
                return obj;
        }
}
```

MỘT SỐ HẠN CHẾ CỦA GENERIC

Không thể tạo mảng

```
Gen<Integer> gens[]=new Gen<Integer>[10]; //error

Gen<?> gens[] = new Gen<?>[10];                //ok
Gens[0] = new Gen<Integer>(25);
Gens[1] = new Gen<String>("Hello");
```

Giới hạn về ngoại lệ generic

Một class generic **không thể kế thừa** class Throwable. Do đó ta không thể tạo class ngoại lệ là generic.

TỔNG KẾT NỘI DUNG BÀI HỌC

- Khái niệm Genegic
- Ưu điểm Genegic
- Tạo class generic và method
- Giới hạn kiểu dữ liệu
- Các ký hiệu đại diện
- Generic method
- Generic Interface
- Một số hạn chế