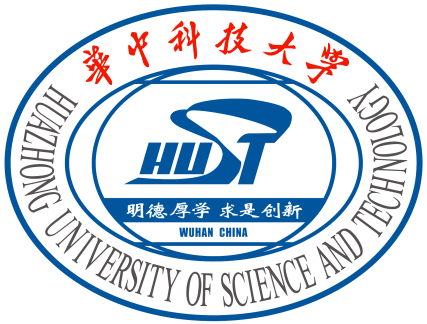
基于Java的面向对象程序设计

陈维亚
weiya_chen@hust.edu.cn

华中科技大学软件学院

第20讲：Java泛型



目录

1. 泛型的定义

❑ 强制类型转换 – 强类型的诅咒

```
Float f = (Float)(new Object());
```

运行时异常： java.lang.ClassCastException

JDK升级过程中，一直致力于将运行时异常转变为编译时错误，降低软件开发和维护的成本。

JDK5引入了泛型**Generics**

从前 （<=JDK 1.4）

```
List list = new ArrayList();
list.add("hello");

Integer i = (Integer)list.get(0);
```

现在

```
List<String> list = new ArrayList<String>();
list.add("hello");

Integer I = list.get(0);
String s = list.get(0);
Object o = list.get(0);
```

1. 泛型的定义

从前 （<=JDK 1.4）

```
List list = new ArrayList();
list.add("hello");

Integer i = (Integer)list.get(0);
```

现在

```
List<String> list = new ArrayList<String>();
list.add("hello");

Integer I = list.get(0);
String s = list.get(0);
Object o = list.get(0);
```

List是使用String类作为其类型参数的一个泛型接口。

不再需要强制转换类型。

不仅仅是类型声明的位置发生变化。

1. 泛型的定义

❑ 泛型类

```
public class OldBag {
    private Object content;
    public OldBag(Object content){
        this.content = content;
    }

    public Object get(){
        return this.content;
    }

    public void set(Object content){
        this.content = content;
    }

    public static void main(String[] args){
        OldBag bag = new OldBag("mybook");
        Object content = (Integer)bag.get();
    }
}
```

1. 泛型的定义

- 泛型类

```
public class Bag<T> {
    private T content;
    public Bag(T content){
        this.content = content;
    }

    public T get(){
        return this.content;
    }

    public void set(T content){
        this.content = content;
    }

    public static void main(String[] args){
        Bag<String> bag = new Bag<String>("mybook");
        Integer content1 = (Integer)bag.get();
        String content2 = (String)bag.get();
    }
}
```

1. 泛型的定义

- 泛型接口

```
public interface List<E>{
    void add(E x);
    Iterator<E> iterator();
}

public interface Iterator<E> {
    E next();
    boolean hasNext();
}
```

1. 泛型的定义

❑ 泛型方法

```
public <E> void printArray(E[] inputArray) {
    // 输出数组元素
    for (E element : inputArray){
        System.out.printf("%s ", element);
    }
    System.out.println();
}
```

所有泛型方法声明都有一个类型参数声明部分（由尖括号分隔），该类型参数声明部分在**方法返回类型之前**。

泛型方法体的声明和其他方法一样。注意类型参数**只能代表引用型类型**，不能是基本类型。

练习 1

写一个排序方法，能够对整型数组排序

```
void bubbleSort(int[] a) {
    int n = a.length;
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-1; j++){
            if(a[j] > a[j+1]){
                int temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    for(int e : a)
        System.out.println(e);
}
```

练习 1

写一个排序方法，能够对整型数组排序

写一个排序方法，能够对浮点数数组排序

```
void bubbleSort(float[] a) {
    int n = a.length;
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-1; j++){
            if(a[j] > a[j+1]){
                float temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    for(int e : a)
        System.out.println(e);
}
```

练习 1

写一个排序方法，能够对整型数组排序

写一个排序方法，能够对浮点数数组排序

写一个排序方法，能够对任何基本数据类型的数组进行排序，该如何实现?

```
<E> void bubbleSort(E[] a) {
    int n = a.length;
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-1; j++){
            if(a[j] > a[j+1]){
                E temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    for(E e : a)
        System.out.println(e);
}
```

如何确保E是可比较大小的?

2. 泛型的分析

❑ 泛型的好处

减少了cast带来的运行时异常

使算法和框架更为通用，减少了冗余代码

2. 泛型的分析

❑ 泛型的潜在问题

```
List<String> ls = new ArrayList<String>();  // 1
List<Object> lo = ls;                       // 2
```

```
lo.add(new Object()); // 3
String s = ls.get(0); // 4: Attempts to assign an Object
to a String!
```

潜在问题一

如果Foo是Bar的子类型（子类或接口实现类），G是一个泛型类，那么G<Foo>并不是G<Bar>的子类型。

2. 泛型的分析

❑ 泛型的潜在问题

JDK 5 以前:

```
void printCollection(Collection c) {
    Iterator i = c.iterator();
    for (k = 0; k < c.size(); k++) {
        System.out.println(i.next());
    }
}
```

JDK 5 以后:

```
void printCollection(Collection<Object> c) {
    for (Object e : c) {
        System.out.println(e);
    }
}
```

2. 泛型的分析

❑ 泛型的潜在问题

JDK 5 以后:

```
void printCollection(Collection<Object> c) {
    for (Object e : c) {
        System.out.println(e);
    }
}
```

如果我们想以各种不同的Collection作为该函数的参数，该怎么办?

Collection<Integer>
Collection<String>
…

于是就有了**Collection<?>** → collection of unknown

3. 通配符

❑ Wildcards

Collection<?> → collection of unknown

```
void printCollection(Collection<?> c) {
    for (Object e : c) {
        System.out.println(e);
    }
}
```

```
Collection<String> c = new ArrayList<String>();
c.add("hello");
printCollection(c);
```

```
Collection<?> c = new ArrayList<String>();
c.add("hello");
printCollection(c);
```

3. 通配符

- 举个例子

```
public abstract class Shape {
    public abstract void draw(Canvas c);
}

public class Circle extends Shape {
    private int x, y, radius;
    public void draw(Canvas c) { ... }
}

public class Rectangle extends Shape {
    private int x, y, width, height;
    public void draw(Canvas c) { ... }
}
```

```
public class Canvas {
    public void draw(Shape s) {
        s.draw(this);
    }
}
```

3. 通配符

- 举个例子

```
public abstract class Shape {
    public abstract void draw(Canvas c);
}

public class Circle extends Shape {
    private int x, y, radius;
    public void draw(Canvas c) { ... }
}

public class Rectangle extends Shape {
    private int x, y, width, height;
    public void draw(Canvas c) { ... }
}
```

潜在问题二

drawAll只能接受List<Shape>，不能够接受Shape子类的数组。

```
public class Canvas {
    public void drawAll(List<Shape> shapes) {
        for(Shape s: shapes)
            s.draw(this);
    }
}
```

3. 通配符

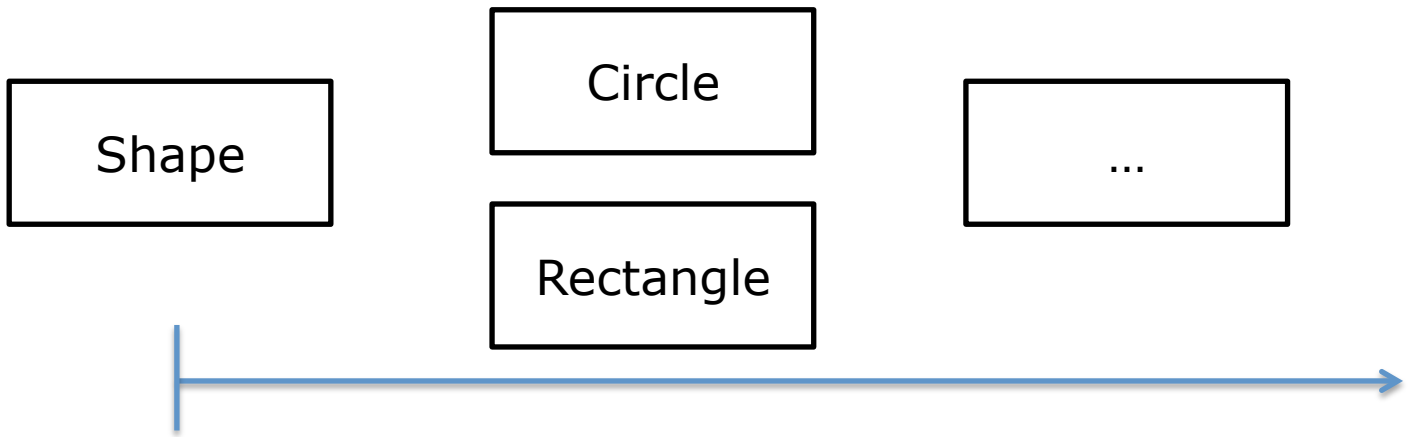
❑ Bounded Wildcards

解决方法是引入有界通配符

潜在问题二

drawAll只能接受List<Shape>，不能够接受Shape子类的数组。

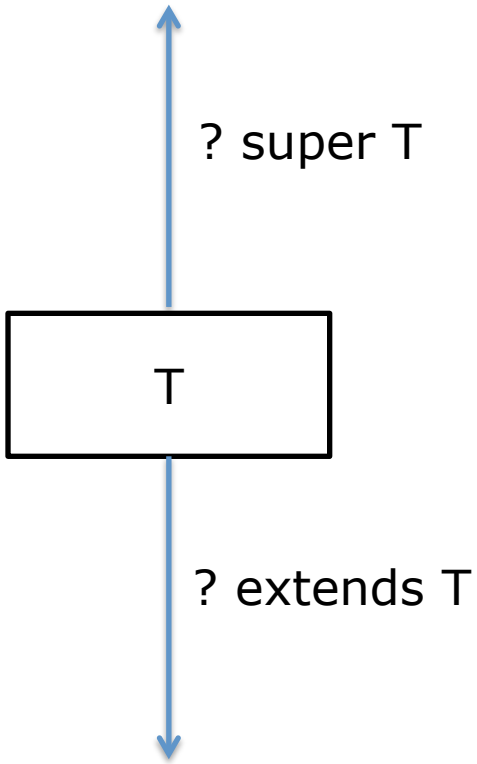
```
public class Canvas {
    public void drawAll(List<? extends Shape> shapes) {
        for(Shape s: shapes)
            s.draw(this);
    }
}
```



3. 通配符

❑ Bounded Wildcards

有界通配符




```
<? extends T> 上界为T

<? super T>   下界为T
```

3. 通配符

❑ Bounded Wildcards

```
<E> void bubbleSort(E[] a) {
    int n = a.length;
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-1; j++){
            if(a[j] > a[j+1]){
                E temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    for(E e : a)
        System.out.println(e);
}
```

如何确保E是可比较大小的?

```
<E extends Comparable<E>> void
bubbleSort(E[] a) {
    int n = a.length;
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-1; j++){
            if(a[j] > a[j+1]){
                E temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    for(E e : a)
        System.out.println(e);
}
```

下节预告

Java 集合框架