Chapter 6 Java Generic Type

javaseu@163.com

https://wdsseu.github.io/java/

Xiang Zhang

<E>

泛型

GT(Generic Type):
Creating a Type with Parameters

- Example
  - Create a MyArrayList
  - Without GT
  - The element is Object
  - If we add a Person to MyArrayList, what can we get() later?

```
public class MyArrayList {

    // the backbone static array;
    private Object[] backArray;

    public MyArrayList(){
        // ...
    }

    // add an element
    public void add(Object o){
        // add this object to backbone;
    }

    public Object get(int index) {
        return backArray[index];
    }

    public static void main(String[] args) {
        MyArrayList list = new MyArrayList();
        Person p = new Person();
        list.add(p);
        // Can we retrieve p as a Person?
    }
}
```

- Create a MyArrayList
- With GT
- The class is defined as parameterized
- If we add a Person to MyArrayList, what can we get() later?

```java
public class MyArrayList<E> {

    // the backbone static array:
    private E[] backArray;

    public MyArrayList(){
        // ...
    }

    // add an element
    public void add(E o){
        // add this object to backbone;
    }

    public E get(int index) {
        return backArray[index];
    }

    public static void main(String[] args) {
        MyArrayList<Person> list = new MyArrayList<Person>();
        Person p = new Person();
        list.add(p);
        // Can we retrieve p as a Person?
    }
}
```

Significance of Generic Type

- Merit of GT
  - Type-safe
  - Eliminate Type Casting
  - Usually used in defining type of elements in Collections

Creating a type with GT

```
public class MyArrayList<E> {

public interface Comparable<T> {
```

// to create a parameterized class

We call them generic type

// to create a parameterized interface

Using a type with GT

// to instantiate an object with a parameterized class

```
MyArrayList<Person> list = new MyArrayList<Person>();

public class Date implements Comparable<Date> {
```

// to create a class with a parameterized interface

Pay Attention

The parameter of a generic type should be a Class, not primary type.

```
ArrayList<Integer> list1 = new ArrayList<Integer>();
ArrayList<int> list2 = new ArrayList<int>();
```

Bounded Type Parameter 受限参数类型

- Sometimes, parameter type in GT should be limited
- Can **Person** be the parameter type in SortedCollection?

```
// a collection that all elements are sorted
public interface SortedCollection<E> {
    // add an element with type E in a sorted way;
    void add(E e);

    //remove an element in the position index, and re-sorts;
    void remove(int index);

    //get an element with type E which is the nth largest
    E get(int index);
}
```

Bounded Type Parameter

- Bounded Type Parameter

```
interface SortedCollection<E extends Comparable<E>> {
   // add, remove and get methods
}
```

  - The meaning of "extends" is generic (inheritance or implementation)
  - Comparable interface is called upper bound
  - E is called a Bounded Type Parameter

Bounded Type Parameter

Multiple-bounded

one upper bound

```
interface SortedCharSeqCollection<E extends Comparable<E>
                                  & CharSequence> {
    // add, remove and get methods
}
```

another upper bound, should also be an interface

Usage of Generic Type

- Example: Write a Static Method to Sum a List
  - 1. without GT

```
public static double sum(List L){
    // calculate the sum
}
```

  - Short: We do not know the element type, and cannot guarantee each element in List L is able to sum up.

Usage of Generic Type

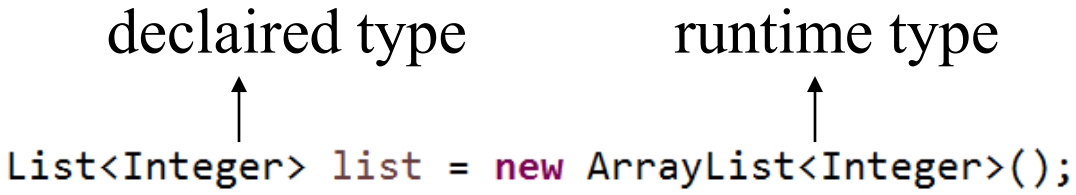

```
public static double sum(List<Number> list){
    double result = 0.0;
    for(int i=0; i<list.size(); i++){
        Number n = list.get(i);
        result += n.doubleValue();
    }
    return result;
}

public static void main(String[] args) {
    List<Integer> list = new ArrayList<Integer>();
    for(int i=0; i<1000; i++){
        list.add(i);
    }
    System.out.println(sum(list));
}
```

Number is the superclass of Integer/Long/ Double.., but why this code is wrong?

What We Got

- The runtime type of an object is determined by:
  - Its class
  - Its class parameter



- So
  - Passing ArrayList<Number> to List<Number>, OK!
  - Passing ArrayList<Integer> to List<Number>, NO!!
- ArrayList<Integer> is not a subclass of List<Number>
- List<Integer> **is not** a subclass of List<Number>
- What should we do to maintain versatality?

Usage of GT

With GT and Wildcard

```
public static double sum(List<? extends Number> list){
    double result = 0.0;
    for(int i=0; i<list.size(); i++){
        Number n = list.get(i);
        result += n.doubleValue();
    }
    return result;
}

public static void main(String[] args) {
    List<Integer> list = new ArrayList<Integer>();
    for(int i=0; i<1000; i++){
        list.add(i);
    }
    System.out.println(sum(list));
}
```

Remember

1. `public void add(List<Number> list) {}`

≠

2. `public void add(List<? extends Number> list) {}`

≠

3. `public void add(List<?> list) {}`

For 1.2.3, write down all actual parameter that can be passed into add method

Guess


```
public void add(List<Number> list) {}
public void add(List<? extends Number> list) {}
public void add(List<?> list) {}
```

Can we define all these methods in a same class?

Guess

- ArrayList<Number> a = new ArrayList<Integer>();
  // right or not?
- ArrayList<? extends Number> a = new ArrayList<Integer>(); // right or not?

Usage of Generic Type

- Usage of Wildcard
  - Define Upper bound
    - List<? extends Number>
  - Define Lower bound
    - List<? super Integer> // Pls give 5 matching types
  - Multiple-bounded is not allowed
    - List<? extends Number & Serializable> // Compiling error
  - Guess: is List<?> different with List<Object>?

More on <?> and <? extends xx>


```
public static void test(List<? extends Number> list){
    Number firstValue = list.get(0);
}
```

not a type loss, but we will lose some details

```
public static void test(List<?> list){
    Object firstValue = list.get(0);
}
```

there is a type loss

Example

```
public static double sum(List<? extends Number> list){
    double result = 0.0;
    for(int i=0; i<list.size(); i++){
        Number n = list.get(i);
        double value = n.doubleValue();
        result += value;
    }
    return result;
}

public static void main(String[] args){
    List<Integer> list = new ArrayList<Integer>();
    for(int i=0; i<1000; i++){
        list.add(i);
    }
    System.out.println(sum(list));
}
```

Example

```
public static double sum(List<Object> list){
    double result = 0.0;
    for(int i=0; i<list.size(); i++){
        Object n = list.get(i);
        double value = ((Number)n).doubleValue();
        result += value;
    }
    return result;
}

public static void main(String[] args){
    List<Integer> list = new ArrayList<Integer>();
    for(int i=0; i<1000; i++){
        list.add(i);
    }
    System.out.println(sum(list));
}
```

Example

```
public static double sum(List<?> list){
    double result = 0.0;
    for(int i=0; i<list.size(); i++){
        Object n = list.get(i);
        double value = ((Number)n).doubleValue();
        result += value;
    }
    return result;
}

public static void main(String[] args){
    List<Integer> list = new ArrayList<Integer>();
    for(int i=0; i<1000; i++){
        list.add(i);
    }
    System.out.println(sum(list));
}
```

Lab Work

- Creating a SortedList with GT
  - Dynamic Array
  - add(E e) / get(int index) / set(int index, E e) / remove(int index) / toString()
  - After adding an element, your class shold automatically sort the array

Generic Array

You can declare a generic array

```
public class SortedList<E extends Comparable<E>> {

    private E[] array;

    public SortedList(int initCapacity) {
        array = new E[initCapacity];
    }
```

But you cannot initialize a generic array, without a specified E, this array cannot be initialized.

Generic Array


```
public class SortedList<E extends Comparable<E>> {

    private E[] array;

    public SortedList(E[] array) {
        this.array = array;
    }
```

You can assign the generic array with an input array.

Self-study

- Text Processing
  - Basic elements in text processing (chapter 7)
  - String / StringBuffer / StringBuilder // try to benchmark
  - Scanner
  - StringTokenizer
  - Java and IR (Information Retrieval)
    - Tokenizing (分词)
    - Stopwords Removal (去除停用词)
    - Rooting (取词根)
    - Indexing (索引)

Forecast

- AWT and Swing Introduction
- Swing Container（JFrame, JPanel）
- Swing Components
- Layout Manager
- Event and Event-based Programming
- Menu