Chapter 6
Java Generic Type

Wang Yang
wyang AT njnet.edu.cn


Outline

- Last Chapter Review
- Significance of Generic Type
- Definition of Generic Type
- Usage of Generic Type
- Forbidden of Generic Type
- Reference

Last Chapter Review

Collection

- Array & Arrays
- initialize : fill / copyOf
- search & sort : binarySearch / sort
- misc : toString / equal


Collection

- Collection<E>
- List<E>
  - ArrayList<E> / LinkedList<E>
- Map<K,V>
  - HashMap<K,V>
- Collections
- Performance Benchmark

Significance of Generic Type

Sometimes

- We want a data structure class for all classes
  - Collection<E>, Map<K,V>
- We want a function work all classes
  - Arrays.sort(T[] a, Comparator<? super T> c)

If we don't have Generic...

- We want a Collection, so we do it like

```
public class Collection {
    Object [] array;
    int size = 0;

    public Collection(int capacity){
        array = new Object[capacity];
    }

    public void add(Object o){
        array[size++] = o;
    }

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

If we don't have Generic...

- we use it like...

```
Collection students = new Collection(100);
students.add(new Student("张三", 7111101));
students.add(new Student("李四", 7111102));
student s = students.get(0);
```

If we don't have Generic...

- we use it like...

```
Collection students = new Collection(100);
students.add(new Student("张三", 7111101));
students.add(new Student("李四", 7111102));
student s = students.get(0);
```

Wrong, you can't assign an object to a student

If we don't have Generic...

```
Collection students = new Collection(100);
students.add(new Student("张三", 7111101));
students.add(new Student("李四", 7111102));
Student s = (Student) students.get(0);
```

If we don't have Generic...

```
Collection students = new Collection(100);
students.add(new Student("张三", 7111101));
students.add(new Student("李四", 7111102));
Student s = (Student) students.get(0);
```

```
students.add(new Cat("小白"));
students.add(new Dog("袜子"));
```

Compiler would allow it, because they are all derived from Object

If we don't have Generic...

```
Collection students = new Collection(100);
students.add(new Student("张三", 7111101));
students.add(new Student("李四", 7111102));
```

```
students.add(new Cat("小白"));
students.add(new Dog("袜子"));

Student s = (Student) students.get(3);
```

Compiler would allow it

But JVM would found it

```
java.lang.ClassCastException: Dog cannot be cast to Student
```


If we don't have Generic...

- So If we don't have Generic
  - We still can work
  - but it is inconvenient : do the cast
  - and it is error-prone : mistake between different Classes

we want Generic

- We want an mechanism that help us do it
  - correctly : check before the class casting
  - concisely : do the class casting automatically
- It is Java Generic.
  - write code that is safer and easier to read
  - especially useful for general data structures


Definition of Generic Type

Generic Type

- What is Generic Type
  - a generic class or interface that is parameterized over types.

type parameter

```
public class Pair<T> {
    T first;
    T second;
    Constructor has not type
    public Pair(T value1, T value2){
        first = value1;
        second = value2;
    }

}
```

Generic Type

- What T can do
  - class variables,
  - parameters,
  - return type
  - local variables

```
T first;
T second;

public Pair(T value1, T value2){
    first = value1;
    second = value2;
}

public T getFirst(){
    return first;
}

public T getSecond(){
    return second;
}

public void swap(){
    T temp;
    temp = first;first = second; second = temp;
}
```

Generic Type

- A generic class can have multi types parameter

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

    public Pair(T value1, S value2){
        first = value1;
        second = value2;
    }
```


Generic Type

- Type Parameter Naming Conventions
  - single, uppercase letters
  - E - Element (used extensively by the Java Collections Framework)
  - K - Key
  - N - Number
  - T - Type
  - V - Value
  - S,U,V etc. - 2nd, 3rd, 4th types

Generic Type

- Instantiate
  - use the real class type to instantiate the generic type
  - Pair<String> pair
  - Pair<String, Integer> pair
  - Pair<String, Pair<String, Integer>> pair

```
Pair<String> p = new Pair<String>("Hello", "World");
```

Generic Method

- Generic Method
  - methods that introduce their own type parameters

```
public class Util {
    static <T> boolean compare(Pair<T> p1, Pair<T> p2){
    if(p1.first.equals(p2.first) &&
            p1.second.equals(p2.second)){
        return true;
    } else {
        return false;
    }
}

static <T> Pair<T> getMax(Pair<T> p1, Pair<T> p2){
```

Internal in Java Generics

- Java Generics technique : Erasure
  - Replace all type parameters in generic types with their bounds or Object
  - Insert type casts if necessary to preserve type safety.
  - do all the check in Compiler

Usage of Generic Type

Bounded Type Parameters

- If we want specify some ability of T

```
public class Pair<T> {
    T first ;
    T second ;
```

```
public Pair(T value1, T value2){
    first = value1;
    second = value2;
}

public int sum(){
    Number n1 = (Number)first;
    Number n2 = (Number)second;
    return (n1.intValue() + n2.intValue());
}
```

Bounded Type Parameters

- If we Instantiate a class without the ability

```
public static void main(String args[]){
    Pair<String> p1 = new Pair<String>("1","2");
    p1.sum();
```

```
ClassCastException: java.lang.String cannot be cast to java.lang.Number
```

Bounded Type Parameters

- We want to tell Compiler about it
  - Bounded Type Parameters

```
public class Pair<T extends Number> {
    T first ;
    T second ;
```

```
Pair<String> p1 = new Pair<String>("1","2");
```

Compiler find it : Bound Mismatch

Wildcard

- Sometimes the Java Generic looks mad...

```
static int sum(List<Number> list){
    int sum = 0;
    for(Number n : list){
        sum += n.intValue();
    }
    return sum;
}

public static void main(String args[]){
    List<Integer> l = new ArrayList<Integer>();
    sum(l);
}
```

The Compiler won't let you do it!!

Wildcard

- Unfortunately, List<Number> has nothing to do with List<Integer>
- We want another way tell compiler we want List<subclass of Number>
- Java introduce wildcard : **?**

Wildcard

```
static int sum(List<? extends Number> list){
    int sum = 0;
    for(Number n : list){
        sum += n.intValue();
    }
    return sum;
}

public static void main(String args[]){
    List<Integer> l = new ArrayList<Integer>();
    sum(l);
}
```


Wildcard

- Upper Bound Wildcard
- Lower Bound Wildcard
- Unbounded Wildcard

Wildcard

- Upper Bound Wildcard : ? extends
  - all instance are subclass of upper bound
  - Scenario : the variable are in variable
  - the function read the elements from the variable

```
static int sum(List<? extends Number> list){
     int sum = 0;
     for(Number n : list){
          sum += n.intValue();
     }
     return sum;
}
```

Wildcard

- Lower Bound Wildcard : ? super
  - all instance are superclass of lower bound
  - Scenario : the variable are out variable
  - the function add the elements to the variable

```
static void addNumbers(List<? super Integer> list) {
    for (int i = 1; i <= 10; i++) {
        list.add(i);
    }
}
```

Wildcard

- Unbounded Wildcard : ?
  - all instance can be any class
  - Scenario : limited actions, only Object function are safe

```
static void printList(List<?> list) {
    for (Object elem: list)
        System.out.print(elem + " ");
    System.out.println();
}
```

Forbidden of Generic

Cannot : Primitive Types

- Cannot Instantiate Generic Types with Primitive Types

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

    public static void main(String args[]){

        Pair<int, char> p = new Pair<int, char>();
        Pair<Integer, Character> p = new Pair<Integer, Character> p();

    }
```

wrong

Right

Cannot : Create Instance

- Cannot Create Instances of Type Parameters

```
public class Pair<T, S> {
    T first = new T();
    T second = new S();
```

Wrong

Cannot : Declare Static

- Cannot Declare Static Fields Whose Types are Type Parameters

```
public class Collection<T> {
    Object [] array;
    static T internal;
    ......
    ......
public static void main(String[] args){
    Collection<String> stringCol = new Collection<String>();
    Collection<Integer> stringCol = new Collection<Integer>();
    Collection<Person> stringCol = new Collection<Person>();
```

What's the internal's type: String ? Integer ?

It's a disaster!!

Cannot : Arrays

- Cannot Create Arrays of Parameterized Types

```
List<Integer>[] arrayOfLists = new List<Integer>[2];  // compile-time error

arrayofLists[0] =  new ArrayList<Integer>();
arrayofLists[1] =  new ArrayList<String>();
```

Cannot : Exception

- Cannot Create, Catch, or Throw Objects of Parameterized Types

```
// Extends Throwable indirectly
class MathException<T> extends Exception { /* ... */ }    // compile-time error

// Extends Throwable directly
class QueueFullException<T> extends Throwable { /* ... */ // compile-time error
```

Cannot : Overload

- Cannot Overload a Method Where the Formal Parameter Types are used

```
public class Example {
     public void print(Set<String> strSet) { }
     public void print(Set<Integer> intSet) { }
}
```

Reference


- Java Tutorial Lesson : generics http://docs.oracle.com/javase/tutorial/java/generics/restrictions.html
- Thinking in Java : Generics : page 440