The New Java Generic Mechanism and Its Application

F. Tian, H.H. Shi, M.T. Yan
College of Computer Information and Engineering
Jiangxi Normal University
Nanchang, China

***Abstract*—In the paper, the core ideas of Java language generic mechanism are described and analyzed in detail, including the generic class, generic method, generic interface, generic wildcard as well as erasure and restrictions on generics, and the Java generic functions are also summarized. An ingenious KLEEN algorithm program is presented and implemented by applying new Java generic mechanism. The paper aims to reveal the generics-based Java component design idea and program refactoring technology for complex algorithmic problems.**

Keywords-java generic mechanism; erasure and restrictions; KLEEN algorithm; refactoring technology

I. INTRODUCTION

The safety and development efficiency of software has attracted a widespread concern in modern software engineering, and software reuse has been encouraged as an effective method. In 1986, D. Mcilroy introduced reusable software components which greatly improved the production efficiency of software[1].In the Adalanguage, users can define type or operation as parameter by themselves. When the procedures are executed, these parameters are assigned. In this way, more general components can be designed, and their reusability is greatly improved.

However software reuse is a bit difficult application due to the low abstraction level of the traditional reusable components. Generic programming can be viewed as an effective way to the development of reusable components for its characteristics of high abstraction and high reusability[2]. Nowadays, many advanced programming languages support generic programming through some language mechanism, such as templates in C++, polymorphic functions in ML, type parameterization in Java, etc. Generic programs reflects a type of programming methodology that focus on the aspects of common properties and variety properties of algorithmic programs, and separates the former, such as algorithm structure, etc., from those algorithmic programs while their running results will not be affected, and therefore improves the readability and reusability of programs[3].

With the growing demand for computer software, generic programming has been increasingly used and its supporting mechanisms have also been optimized and expanded continuously, which provides more effective solutions for complex algorithmic problems.

II. JAVA GENERIC MECHANISMS

A. The JAVA Generic Class

A generic classis defined through parameterizing types rather than defining the type of the variable as Object. The benefit of doing so is to make the type safety inspection generics in compiling stage, thus ensuring type safety. Type parameters (represented by T) of generic class can be used to define a class member variables, methods and the return value of the method type parameters.

B. The Java Generic Method

Method with formal type parameters is called a generic method, whether to have a generic method is independent of the class is generic, only you need placed type parameters or parameter before the method. The use of generic methods to program brought an efficient and concise, easy to call. Java generic method declaration as follows: [access specifies] [static] [final] <type parameter list>[return value type] method name (formal parameter list []).

The type parameter T is used to represent the unknown type, and can also be used extends and the super keyword to specify the range of the T type respectively. The extends statement is used to limit the upper bound of generic parameters, while the super statement is used to limit the lower bound of generic parameters.

C. The Java Generic Interface

Defining a generic interface is similar with a generic class, and the word T represents a type parameter in the brackets. Generic interface is also not directly used to create the object, but can be used to declare an object variable. We can define more than one type parameter. The statement as follows:

Public interface G<T1，T2>{ T1 getT1();T2 getT2();… }

The use of generic interfaces needs to pass the class to implement the interface. If you need to implement the generic interface and retain the interface in a generic class, when they are defined to be retained in the generic declaration generic interface. Implementation of generic interface type must be a generic class, the number of its type parameter must be greater than or equal to the type parameters of generic interface.

D. Erasure and Restrictions on Java Generics

Generics are implemented using an approach called type erasure: The compiler uses the generic type information to compile the code, but erases it afterward. Thus, the generic information is not available at runtime. This approach enables

the generic code to be backward compatible with the legacy code that uses raw types. Because generic types are erased at runtime, there are certain restrictions on how generic types can be used. Here are some of the restrictions[4]:

(1)Cannot use new T(). You cannot create an instance using a generic type parameter. For example, the following statement is wrong: T object=new T();Because the compile time all generic types will become the raw types, then there is no generic T type and it is invalid.

(2) Cannot use new T[].You cannot create an array using a generic type parameter. For example, the following statement is wrong: T[] e=new T[size]; Because if you create a type array and then transition to object, which will lead to an array creation failed.

(3) A generic type parameter of a class is not allowed in a static context. Since all instances of a generic class have the same runtime class, the static variables and methods of a generic class are shared by all its instances. Therefore, it is illegal to refer to a generic type parameter for a generic class in a static method, field.

(4) Replace the type parameter cannot be used basic types. Such as List<int>, because int type as the basic type does not belong to the object and must use the package type.

(5)Exception classes cannot be generic. The generic exception capture will be used in the variation of error.

III. DESIGN AND IMPLEMENTATION OF A KLEEN ALGORITHM

A. *KLEEN Algorithm Ideas*

KLEEN algorithm abstracts three algorithm programs, that is, all pairs shortest pathproblem (Floyd's algorithm), transitive closure problem, and a maximum capacity routing problem, into an abstract algorithm program, where their common structure has been defined as an generic program and variety properties have been defined as generic parameters.

(1) Floyd's Algorithmbasic idea: the all pairs shortest path problem is: given a graph, find the shortest path between every pair of vertices in that graph. The basic notion is that we update an adjacency matrix N times. First, we update the graph to reflect the shortest known paths going through node 1. At step 2, we update the graph to reflect the shortest known path going through node 1 or node 2 (or both). At step 3... And so on[5]. Here is the iterative formula: a[i, j]=min(a[i, j], a[i, k]+a[k, j]).

(2)Transitive closure algorithm basic idea: Transitive closure algorithm represents a directed graph from the adjacency matrix, seeking the path of all nodes are reachable, this matrix is to pass the required closure matrix. Directed graph vertex i to vertex j has the path starts from the vertex i go through the other points (or without other points) to reach the vertex j, the vertex i to j if there is a path, then R [i, j] is set to true, otherwise it is set to false. Suppose R(k) represents the path contains the first k vertices as intermediate vertices, each of the matrix R(k) for its front matrix R(k-1) is on the path for the addition of a vertex[6].The iterative formula: R[i, j]=or (R[i, j], R[i, k] and R[k, j]).

(3)The maximum capacity of the routing algorithm thought: In a directed graph in all the way from the vertex vi to vertex vj ($i \neq j$), the largest capacity is called the maximum capacity of the routing from a vertex vi to the vertex vj in a directed path. Suppose C (k) for the middle after the capacitance values of k vertices, each of the C (k) is for C (k-1) is to increase the capacity value of a vertex on the path. Similarly, the iterative formula: C[i, j]=max(C[i, j], C[i, k] min C[k, j]).

B. *The Generic Design of KLEEN Algorithm*

With the introduction of generics and generic mechanism of constant change in the Java language, the Java language features become more powerful. Generic mechanism provides a powerful method to deal with the problem of complex algorithms component. As can be seen from the above algorithm thought, KLEEN algorithm contains three sub-algorithms common operating structure, they can be packaged into a generic algorithm components so that simplifying the procedures and improving the efficiency of the program design.

Java language generic mechanism has a generalization procedure and increase the reusability of program, which combined with abstract classes、 inheritance、covering and other features, this design can effectively design a KLEEN algorithm components. The main Kleen program as follow：

```
publicclassKleen {
        abstractclass kleene<T>{
                publicabstract T BigMulti(T a,T b);
                publicabstract T BigPlus(T a,T b);
                publicvoid genericrun(int n,T[][] c){
                for(k=1;k<=n;k++)
                        for(i=1;i<=n;i++)
                                for(j=1;j<=n;j++)
                        c[i][j] =
BigMulti(c[i][j],(BigPlus(c[i][k],c[k][j])));
                        System.out.println(c[i][j] + ",");}
                ……
        class floyd<T>extends kleene<T>{
                public T BigMulti(T a, T b) {return
Math.min(a, b);}
                public T BigPlus(T a, T b) {return a+b;}}
        class close_set<T>extends kleene<T>{
                public T BigMulti(T a, T b) {return a || b;   }
                public T BigPlus(T a, T b) {return a &&
b;   }}
        class capcity<T>extends kleene<T>{
                public T BigMulti(T a, T b) {return
Math.max(a, b);}
                public T BigPlus(T a, T b) {return
Math.min(a, b);}}
        publicvoid maincall(){
                ...
                floyd<Intger> floydproc = new
floyd<Intger>();
                floydproc.genericrun(num,c1);……
        publicstaticvoid main(String[] args) {
                Kleen aco = new Kleen ();
                aco.maincall();}}
```

In the above description of the Java program, the program by creating a generic abstract class to complete the Kleen algorithm common function package. According to the development of the concept of algorithm components[7],at first, the definition of the two abstract methods in abstract classes to describe the component which can provide service operation, and use a common approach to design and implement component functionality itself, and by the generic type parameter passed to implement the call to component interfaces. And then through the subclass inheritance of abstract classes cover the implementation of service operations in the component. Finally through the generic type instantiation of type abstraction class complete transmission parameters, thus realizing the external calls to the component interface.

IV. CONCLUSIONS

As an extension of object-oriented programming techniques, generic programming has attracted more and more concern. The Generic mechanisms can improve reliability and productivity of software components by means of the safety features and high reusability of generic programs.

We presented the new Java generic mechanisms and its programming methodology in the paper, and employed the mechanisms and component development method to implement an ingenious KLEEN algorithm program, from which we could see that the Java generic mechanisms provide a solution to improve the reusability and abstract level of algorithm component substantially. It can be concluded that the generic mechanisms will be more widely used in modern software engineering to guarantee reliability and reusability of software components in the future.

ACKNOWLEDGEMENTS

This work was supported in part by NSFC (No.61363013), NSF of Jiangxi Province (No. 20142BAB217026).

REFERENCE

[1] M.D.Mcilroy. Mass produced software components. In Software Engineering. P.Naur and B.Randell,Eds. Germisch, Germany:NATO Sci.Committee,138-155,Jan.1969.

[2] YLLi, G.Novak. Generation of geometric programs specified by diagrams. In: Denney E, Schultz U, eds. Proc. of the 10th Int'lConf. on Generative Programming and Component Engineering (GPCE 2011). New York: ACM Press. 63-72, 2011.

[3] Wen-shengXu, Jin-yunXue. An extension to generic programming and Its Implementation in Java. Computer Engineering & Science.29(10):89-93, 2007.

[4] Y.Daniel Liang. Introduction to Java programming,9th Edition. Armstrong Atlantic State University.2012.

[5] L. Ridi, J. Torrini, E. Vicario. Developing a Scheduler with Difference-Bound Matrices and the Floyd-Warshall Algorithm. Software, IEEE, 29(1):76-83, 2011.

[6] S.T.Chakradhar, V.D.Agrawal, S.G. Rothweiler. A transitive closure algorithm for test generation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 12(7): 1015 – 1028, 1993.

[7] Pan-pan WU， YiYAN. Encapsulation of algorithm module based on component technology in ladder diagram programming. Journal of Mechanical & Electrical Engineering, 30(6):764-768, 2013.