

Presumed Function in Typed Programming Languages

Author: Weihan Huang

Affiliation : Master of Computer Science Department, State University of New York at Buffalo, U.S.A.

Master of Physics Department, National Hsing Hua University, Taiwan

Email : weihanh@yahoo.com.tw

DOI: 10.26821/IJSHRE.9.9.2021.9910

ABSTRACT

The concept "Interface" in Java is firstly introduced. And then I show that the use of Interface can solve the problem of invalid function call error by the example of finding the maximum value of a list. Next I propose a new way "Presumed Function" for the same task : finding the maximum value of a list. And lastly I list the advantages of Presumed Function over Interface.

Keywords: Function, Interface, Language,

Presumed, Programming, Typed

1. INTRODUCTION TO INTERFACE

Interface let us be able to make a standard for the behavior of a generic class. Classes that obey this standard implement the behavior functions declared in the Interface.

For example, if we would like to make a standard that

an object's class should have its name string, so that it will be convenient to print the name string for debugging. Then we can create an interface as follows :

```
public interface NameString {  
    public abstract String  
        getNameString(Object object);  
}
```

If we would like the type "number" to obey the standard behavior, then we can make a class NumberNameString in the following

```
.public class NumberNameString implements  
    NameString  
{  
    public String getNameString(Object object) {
```

```
return ((Number)object).value+"";
    }
}
System.out.println("boolean name is "+new
    BooleanNameString().getNameString(new
    Boolean(true));
```

Note that this class "implements NameString" meaning that it follows the standard of Interface NameString, and defines the way of getting string by the implementation in the method getNameString.

Similarly for boolean, we have the following to get nameString of boolean object

```
public class BooleanNameString implements
    NameString {
    public String getNameString(Object object) {
    return ((Boolean)object).booleanValue()+"";
    }
}
```

Hence we can test them by the following program :

```
public class TestNameString {
    public static void main(String[] args) {
    System.out.println("number name is "+new
    NumberNameString().getNameString(new
```

```
}}
The outputs are
    number name is 2
    boolean name is true
```

So this is a short introduction of Interface. The spirit is that since we don't know the details of each class, Interface leaves the implementation to each class!

2. PROBLEM AND THE INTERFACE SOLUTION OF INVALID FUNCTION CALL

2.1 Problem

Suppose we would like to write a function that computes the maximum value of a list.

Say

```
public static Object findMax(List list) {
    for(int i=list.size()-1;i>=1;i--)
```

```
if(list.get(i)>list.get(i-1))) {  
  
    Object temp=list.get(i);  
  
    list.set(i, list.get(i-1));  
  
    list.set(i-1, temp);  
  
    }  
  
}  
  
return list.get(0);  
  
}
```

Here Object is a root type in OOP, so the function is supposed to be able to deal with any sub typed lists.

However, take a look at the condition of if :

```
list.get(i)>list.get(i-1)
```

The types of both sides are all Object class instance,

hence the compiler will check if there is a function

```
Object>Object
```

Unfortunately, there are no such functions defined. So

the compiler will show an Invalid Function Call Error.

Therefore the function call is invalid, we need to

figure out some way to rescue it.

2.2. Interface Solution

Now let's see how to use Interface to solve this problem. We will generalize ">" as an interface

Comparator defined below

```
public interface Comparator {  
  
    public abstract boolean isGreaterThan(  
  
        Object object1, Object object2);  
  
}
```

Hence the function that computes the maximum of a

list is re-written as

```
public static Object findMax(List list, Comparator  
    comparator) {
```

```
    for(int i=list.size()-1;i>=1;i--) {
```

```
        if(comparator.isGreaterThan(  
  
            list.get(i), list.get(i-1))) {
```

```
                Object temp=list.get(i);
```

```
                list.set(i, list.get(i-1));
```

```
                list.set(i-1, temp);
```

```
            }  
  
        }
```

```
    }
```

```
    return list.get(0);
```

```
}
```

i.e. we pass Comparator as an argument, and the comparator tells the compiler that there is an isGreaterThan function defining the behavior of Comparator. Now, if we want to have numbers in list as the first argument, we can have the NumberComparator as the second argument at the same time. So at run time, the list containing Numbers can be compared by NumberComparator as follows :

```
public class NumberComparator
```

```
implements Comparator {
```

```
public boolean isGreaterThan(
```

```
Object object1 , Object object2) {
```

```
return (((Number)object1).value>
```

```
((Number)object2).value);
```

```
}
```

```
}.
```

```
}
```

The Number Comparator coerces the type Object argument to Number. That is, we already know list containing numbers, so it tells the compiler the

object1 and object2 are all number at run time. So this solves the problem of Invalid Function Call Error.

3. USING PRESUMED FUNCTION TO SOLVE THE INVALID FUNCTION CALL ERROR

Here I propose a second way to solve the problem of Invalid Function Call Error. Recall the place where compiler shows the Invalid Function Call Error :

```
list.get(i)>list.get(i-1)
```

in function findMax

```
public static Object findMax(List list) {
```

```
for(int i=list.size()-1;i>=1;i--) {
```

```
if(list.get(i)>list.get(i-1))) {
```

```
Object temp=list.get(i);
```

```
list.set(i, list.get(i-1));
```

```
list.set(i-1, temp);
```

```
}
```

```
}
```

```
return list.get(0);
```

```
}
```

The compiler shows the error because there is no

function Object>Object defined. Here I propose a "Presumed Function" to solve the problem. All we need to do in Presumed Function is to write down a warning sign saying that we presume there is a function > that will compare two objects.

```
public static Object findMax(List list) {  
  
    for(int i=list.size()-1;i>=1;i--) {  
  
        if(list.get(i)>list.get(i-1)) {    !!! Presumed  
  
            Function          :  
  
            Object>Object !!!  
  
            Object temp=list.get(i);  
  
                list.set(i, list.get(i-1));  
  
            list.set(i-1, temp);  
  
        }  
  
    }  
  
    return list.get(0);  
  
}
```

So at compile time, when the compiler sees the warning message at the same line of the function call, it will not show Invalid Function Call Error. And at run time, programmers must guarantee the run time

object is comparable and has a function defined. For example,

```
Number>Number  
  
at run time, the two sides are numbers, and numbers have the function Number>Number defined.
```

This is as same as the solution of Interface. At run time, programmer must guarantee that the type coercion is applicable. They must be numbers for the

```
public class NumberComparator implements  
    Comparator {  
  
    public boolean isGreaterThan(Object object1 ,  
        Object object2) {  
  
        return (((Number)object1).value>  
  
            ((Number)object2).value);  
  
    }  
  
}
```

4. ADVANTAGES OF PRESUMED FUNCTION

4.1. The Presumed Function Object>Object deals with

all classes that have `>` defined, while the classes implementing the interface `Comparator` can use type coercion for only one class use. For example, `NumberComparator` can only do with `Number`, `StringComparator` can only do with `String`. Therefore Presumed Function `Object>Object` can do with much more cases than Interface `Comparator`.

4.2 It is more natural to think that the elements in a list for finding its maximum have the ability of comparing when we write the find-maximum procedure. Thus the code is more intuitive to illustrate the algorithm. In human's thinking, we will simply write `"list.get(i)>list.get(i-1)"` with the assumption that they are comparable, so also natural language programming[1] can use the warning line "presumed function : `Object>Object`" to mimic this assumption

4.3. The code `"if(list.get(i) >list.get(i-1))"` is shorter and neater than introducing a comparator

```
"if(comparator.isGreaterThan  
  
    (list.get(i),list.get(i-1))".
```

4.4. The procedure find Max now only need take one

argument, the list, and it does not need to take a second argument, the comparator. And therefore also we don't need to introduce a new interface "comparator".

5. CONCLUSION

I have introduced the concept of "Interface" in Java[2] and then use it to solve the problem of Invalid Function Call Error. A new method "Presumed Function" is proposed to solve the same problem. Lastly I give the advantages of Presumed Function over Interface.

6. REFERENCES

- [1] Natural Language Programming
https://en.wikipedia.org/wiki/Natural-language_programming
- [2] Java
[https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))