Advanced Programming Second Project

Generic Functions in Java

Introduction

- CLOS' Generic Functions
- CLOS provides Multiple Dispatch
- Java only provides Single Dispatch with Overloading
- Facilitate the use of Multiple Dispatch in Java

My Solution

- Select applicable methods
- Sort applicable methods
- Compute effective method

How do I store methods?

- Every Method is registered through GFMethod Class
- Get the call method signature through reflection
- Convert it to a String
- Store everything in a Map

```
private final TreeMap<String, GFMethod> primary;
private final TreeMap<String, GFMethod> before;
private final TreeMap<String, GFMethod> after;
```

Select Applicable Method

- Get all applicable methods by searching all registered methods.
  - Convert all keys to Class<?>[]
  - Search for before, primary and after methods
  - The applicable criteria is based on the method Class.isAssinableFrom

```
ArrayList<Class<?>[]> primaryMethods = getApplicableMethods(k, primary);
ArrayList<Class<?>[]> beforeMethods = getApplicableMethods(k, before);
ArrayList<Class<?>[]> afterMethods = getApplicableMethods(k, after);
```

Select Applicable Method

```
private ArrayList<Class<?>[]> getApplicableMethods(Class<?>[] args, TreeMap<String, GFMethod> allMethods) {
    Class<?>[][] available = null;
    ArrayList<Class<?>[]> applicable = null;
    available = getAllAvailableMethods(allMethods, args.length);
    applicable = new ArrayList<Class<?>[]>();
    boolean app = true;
    for (Class<?>[] a : available) {
        app = true;
        for (int i = 0; i < args.length; i++) {
            if (!a[i].isAssignableFrom(args[i])) {
                app = false;
                break;
            }
        }
        if (app) {
            applicable.add(a);
        }
    }
    return applicable;
}
```

Sort Applicable Method

- From all applicable methods sort them accordingly.
- For before methods sort them from most generic to most specific
- For after methods sort them from most generic to most specific
- Primary methods? Only most specific one is applied.

```
primaryMethods = orderMethodsSpecificFirst(primaryMethods);
beforeMethods = orderMethodsSpecificFirst(beforeMethods);
afterMethods = orderMethodsSpecificLast(afterMethods);
```

Sort Applicable Method

```
private ArrayList<Class<?>[]> orderMethodsSpecificFirst(ArrayList<Class<?>[]> methods) {
    methods.sort(new Comparator<Class<?>[]>() {
        @Override
        public int compare(Class<?>[] arg0, Class<?>[] arg1) {
            int state = 0;
            for (int i = 0; i < arg0.length; i++) {
                if (arg0[i].equals(arg1[i])) {
                    continue;
                } else if (arg0[i].isAssignableFrom(arg1[i])) {
                    return 1;
                } else {
                    return -1;
                }
            }
            return state;
        }
    });
    return methods;
}
```

Compute Actual Methods

- Apply before methods (and discard return values)
- Apply the most specific primary method and return it's value
- Apply after methods (and discard return values)

```
return computeActualMethod(beforeMethods, primaryMethods, afterMethods, args, k);
```

Compute Actual Methods (before)

```
// Before Methods
if (!bMethods.isEmpty()) {
    for (Class<?>[] bM : bMethods) {
        gM = before.get(ClassesToKey(bM));
        m = getCallMethod(gM);
        m.setAccessible(true);
        try {
            m.invoke(gM, args);
        } catch (IllegalAccessException | IllegalArgumentException | InvocationTargetException e) {
            e.printStackTrace();
            return -1;
        }
    }
}
```

Compute Actual Methods (primary)

```
// Call Primary Method
if (!pMethods.isEmpty()) {
    gM = primary.get(ClassesToKey(pMethods.get(0)));
    m = getCallMethod(gM);
    m.setAccessible(true);
    try {
        ret = m.invoke(gM, args);
    } catch (IllegalAccessException | IllegalArgumentException | InvocationTargetException e) {
        e.printStackTrace();
    }
} else {
    String error = "No methods for generic function " + this.name + "with args " + print(args);
    error = error + " of classes " + printSignature(k);
    throw new IllegalArgumentException(error);
}
```

Compute Actual Methods (after)

```
// After Methods
if (!aMethods.isEmpty()) {
    for (Class<?>[] aM : aMethods) {
        gM = after.get(ClassesToKey(aM));
        m = getCallMethod(gM);
        m.setAccessible(true);
        try {
            m.invoke(gM, args);
        } catch (IllegalAccessException | IllegalArgumentException | InvocationTargetException e) {
            e.printStackTrace();
        }
    }
}
```

Future Work

- Provide mechanisms for (call-next-method)
  - In the GFMethod class provide a call-next-method method that would give the generic function the order to apply the next primary method applicable
- Allow the use of around methods
- Allow the use of different ordering algorithms for applicable methods

Questions?