ПРИМЕНЕНИЕ МЕХАНИЗМА РЕТРОСПЕКЦИИ В JAVA

Бахрамов А.Б.,
студент 2 курса ВГУ имени П.М. Машерова, г. Витебск, Республика Беларусь
Научный руководитель – Ермоченко С.А., канд. физ.-мат. наук

Современные языки программирования обладают широкими возможностями для создания программного обеспечения самой различной направленности. Так, например, язык программирования Java позволяет создавать кроссплатформенные приложения для персональных компьютеров (так называемые настольные приложения), для мобильных устройств, а также серверные web-приложения.

Такой широкий спектр видов разрабатываемых приложений накладывает дополнительные требования к языку программирования. Одним из таких требований является поддержка ускоренной разработки приложений, которая достигается за счёт возможности автоматизации некоторых действий. В языке программирования Java к таким инструментам можно отнести механизм ретроспекции (Reflection).

Актуальность исследования заключается в том, что механизм ретроспекции широко применяется в дополнительных библиотеках классов для языка программирования Java, используемых в реальных приложениях, и применение его программистами помогает повысить скорость разработки программного обеспечения и снизить количество ошибок в исходном коде, что, в конечном итоге, улучшает качество программного обеспечения.

Целью данного исследования является демонстрация возможностей механизма ретроспекции Java по упрощению программного кода на примере простого приложения.

Материал и методы. Основным материалом, на котором будет базироваться данное исследование, является официальная документация по языку программирования Java, а частности раздел, посвящённый Reflection API [1]. В качестве методов исследования будут применяться общенаучные методы анализа и обобщения, общие принципы построения программного обеспечения, принципы объектно-ориентированного проектирования. Последние особенно актуальны при разработке программного обеспечения практической направленности [2].

Результаты и их обсуждение. Для демонстрации возможностей механизма ретроспекции при разработке программного обеспечения практической направленности рассмотрим настольное приложение, реализующее простейшую банковскую систему. Конечно, вопросы безопасности и надёжности хранения данных, а также обеспечения высокой доступности в этой системе не рассматриваются. По сути в программе лишь моделируется аналогичный функционал. Приложение позволяет перевести средства с одного счёта в банке на другой:

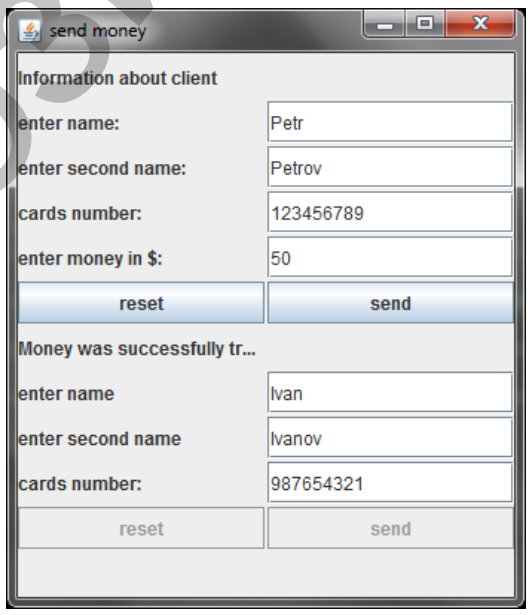

Рисунок 1. Общий вид приложения

Данное приложение архитектурно состоит из трёх частей. Это пользовательский интерфейс, реализация алгоритмов бизнес-логики и подсистема хранения данных. Все эти компоненты реализуются через определённый набор классов, связи между которыми строятся через интерфейсы. Например, интерфейс для подсистемы хранения данных будет выглядеть, как в листинге 1.

Листинг 1. Интерфейс InterfaceWorkUser для работы с подсистемой хранения данных

```
 public interface InterfaceWorkUser {
 public void ReaderInformation(int kartNumber);
 public boolean errorMessage();
 public boolean testInformation(String FirstName, String SecondName,
 int kartNumber, int money);
 public void changedMoney(int money);
}
```

Использование интерфейсов для связей между классами позволяет ослабить зависимости, что делает архитектуру приложения более гибкой и мобильной. Данный подход полностью соответствует принципу инверсии зависимостей. Но для использования таких зависимостей необходимо использовать один из механизмов внедрения зависимостей. Один из них и основан на применении ретроспекции. Основная идея подхода заключается в вынесении зависимостей в отдельный конфигурационный файл (см. листинг 2).

Листинг 2. Конфигурационный файл generator.properties

```
generator = com.Arzuv.ClassWorkUsee
generator.2 = com.Arzuv.RecipientWorkUser
```

А затем при создании объекта класса, от которого зависит некоторый другой класс, применяется ретроспекция (см. листинг 3).

Листинг 3. Применение ретроспекции

```
public class FactoryWorkUser {
 public static InterfaceWorkUser generateDateAboutUser() {
 try {
 String clazz = getFirstClass();
 Class getClass = Class.forName(clazz);
 InterfaceWorkUser iwu = (InterfaceWorkUser)getClass.newInstance();
 System.out.println("everything is working");
 return iwu;
 } catch (Exception e) { /* exception processing */ }
 }

 private static String getFirstClass() throws Exception {
 Properties p = new Properties();
 p.load(new FileReader("generator.properties"));
 return p.getProperty("generator");
 }

 // over analogical methods
}
```

Таким образом, при создании объекта, который будет зависеть от интерфейса InterfaceWorkUser, то в исходном коде приложения явного внедрения этой зависимости нет. Таким образом вносить изменения в программу, например, заменяя одну подсистему хранения на другую, можно только лишь изменением конфигурационного файла без необходимости внесения изменений в само приложение.

Заключение. Разработанное простейшее приложение демонстрирует применение механизма интроспекции в языке программирования Java для связывания объектов классов в соответствии с принципом инверсии зависимостей. Такой подход позволяет гораздо быстрее вносить изменения в приложение, обеспечивая более эффективное применение языка программирования.

1. McCluskey, Glen. Using Java Reflection / Oracle Technology Network [Электронный ресурс]. – Режим доступа: https://www.oracle.com/technetwork/articles/java/javareflection-1536171.html. Дата доступа: 07.09.2018

2. Роберт С. Мартин, Джеймс В. Ньюкирк, Роберт С. Косс. Быстрая разработка программ. Принципы, примеры, практика – Вильямс, 2004