Лабораторная работа №8

Задание на лабораторную работу

Внести изменения в существующий набор типов табулированных функций, позволяющие обрабатывать точки функций по порядку (паттерн «Итератор»), а также выбирать тип объекта табулированной функции при его неявном создании (паттерн «Фабричный метод» и средства рефлексии).

Задание 1 *(1,6 балла)*

Сделайте так, чтобы все объекты типа `TabulatedFunction` можно было использовать в качестве объекта-агрегата в «улучшенном цикле for» (вариант for-each), извлекаемые объекты при этом должны иметь тип `FunctionPoint`.

В интерфейсе `TabulatedFunction` добавьте необходимый родительский тип, используйте при этом параметризованный тип (generic type).

В классах, реализующих интерфейс `TabulatedFunction`, добавьте требующийся метод, возвращающий объект итератора.

Классы итераторов сделайте анонимными. В соответствии с паттерном «Итератор» итераторы должны работать эффективно и использовать знание о внутренней структуре объектов, а не вызывать публичные методы объекта табулированной функции.

Операцию удаления текущего элемента в итераторах реализовывать не нужно. Метод удаления должен всегда выбрасывать исключение `UnsupportedOperationException`.

Метод получения следующего элемента должен выбрасывать исключение `NoSuchElementException`, если следующего элемента нет. Возвращаемый методом объект типа `FunctionPoint` не должен позволять нарушить инкапсуляцию объекта табулированной функции.

В методе `main()` программы проверьте работу итераторов классов табулированных функций, например, следующим образом.

```
--- Пример использования улучшенного цикла for -------------------
        TabulatedFunction f = /* получение или создание объекта */;
        for (FunctionPoint p : f) {
             System.out.println(p);
        }
-------------------------------------------------------------------
```

Задание 2 *(1,7 балла)*

В классе `TabulatedFunctions` ряд методов в ходе своей работы порождает объекты табулированных функций. При этом реальный тип объектов оказывался фиксированным, и изменить его динамически в ходе работы программы было невозможно. Одним из способов решения этой проблемы является применение объектов фабрик с возможностью их замены. Поскольку здесь требуется порождение только одного объекта, воспользуемся паттерном «Фабричный метод».

В пакете `functions` опишите базовый интерфейс фабрик табулированных функций `TabulatedFunctionFactory`. Интерфейс должен объявлять три перегруженных метода `TabulatedFunction createTabulatedFunction()`, параметры которых соответствуют параметрам конструкторов классов табулированных функций.

Для удобства (вообще говоря, такой элемент дизайна необязателен) опишите в классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` классы фабрик `ArrayTabulatedFunctionFactory` и `LinkedListTabulatedFunctionFactory`, реализующие интерфейс фабрики и порождающие объекты соответствующих классов табулированных функций. Сделайте классы фабрик вложенными и публичными.

В классе `TabulatedFunctions` объявите приватное статическое поле типа `TabulatedFunctionFactory` и проинициализируйте его объектом одного из описанных классов фабрик. Также объявите метод `setTabulatedFunctionFactory()`, позволяющий заменить объект фабрики.

Ещё в классе `TabulatedFunctions` опишите три перегруженных метода `TabulatedFunction createTabulatedFunction()`, возвращающих объекты табулированных функций, созданные с помощью текущей фабрики. Параметры методов должны соответствовать параметрам методов фабрики.

В остальных методах класса, где требуется создание объектов табулированных функций, замените явное создание объектов с помощью конструкторов на вызов соответствующего метода `createTabulatedFunction()`.

В методе `main()` проверьте работу фабрик, например, следующим образом.

```
--- Пример проверки работы фабрик -------------------------------
        Function f = new Cos();
        TabulatedFunction tf;
        tf = TabulatedFunctions.tabulate(f, 0, Math.PI, 11);
        System.out.println(tf.getClass());
        TabulatedFunctions.setTabulatedFunctionFactory(new
            LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
        tf = TabulatedFunctions.tabulate(f, 0, Math.PI, 11);
        System.out.println(tf.getClass());
        TabulatedFunctions.setTabulatedFunctionFactory(new
            ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
        tf = TabulatedFunctions.tabulate(f, 0, Math.PI, 11);
        System.out.println(tf.getClass());
-----------------------------------------------------------------------
```

Задание 3 *(1,7 балла)*

Ещё одним инструментом, позволяющим указывать типы порождаемых объектов, является рефлексия.

В классе `TabulatedFunctions` добавьте ещё три перегруженных версии метода `createTabulatedFunction()`. Их параметры должны повторять параметры трёх аналогичных методов, основанных на использовании фабрики, но также эти методы должны получать ссылку типа `Class` на описание класса, объект которого требуется создать. Сделайте так, чтобы в эти методы можно было передать только ссылки на классы, реализующие интерфейс `TabulatedFunction`.

Новые методы создания объектов должны найти в предложенном классе конструктор с соответствующими типами параметров (например, двумя параметрами типа `double` и одним параметром типа `int` для метода, создающего объект табулированной функции по левой и правой границе области определения и количеству точек). С помощью найденного конструктора (в него должны быть переданы фактические параметры) должен быть создан объект табулированной функции. Ссылка на этот объект и должна быть возвращена из метода создания.

Если в ходе выполнения рефлексивных операций возникло исключение (не найден конструктор и т.д.), оно должно быть отловлено (используйте блок `try` с отловом нескольких типов исключений). Вместо него должно быть выброшено исключение `IllegalArgumentException`, причём в его конструктор должно быть передано отловленное исключение из рефлексии. Это позволит в случае возникновения ошибок определить реальную причину ошибки.

В классе `TabulatedFunctions` перегрузите методы, создающие объекты табулированных функций, добавив версии, принимающие также ссылку типа `Class` на описание класса, объект которого требуется создать. Сделайте так, чтобы в эти методы

можно было передать только ссылки на классы, реализующие интерфейс `TabulatedFunction`.

Проверьте в методе `main()` работу методов рефлексивного создания объектов, а также методов класса `TabulatedFunctions`, использующих создание объектов. Это можно сделать, например, следующим образом.

```
--- Пример проверки работы рефлексивных методов ------------------
        TabulatedFunction f;

        f = TabulatedFunctions.createTabulatedFunction(
            ArrayTabulatedFunction.class, 0, 10, 3);
        System.out.println(f.getClass());
        System.out.println(f);

        f = TabulatedFunctions.createTabulatedFunction(
            ArrayTabulatedFunction.class, 0, 10, new double[] {0, 10});
        System.out.println(f.getClass());
        System.out.println(f);

        f = TabulatedFunctions.createTabulatedFunction(
            LinkedListTabulatedFunction.class,
            new FunctionPoint[] {
                new FunctionPoint(0, 0),
                new FunctionPoint(10, 10)
            }
        );
        System.out.println(f.getClass());
        System.out.println(f);

        f = TabulatedFunctions.tabulate(
            LinkedListTabulatedFunction.class, new Sin(), 0, Math.PI, 11);
        System.out.println(f.getClass());
        System.out.println(f);
------------------------------------------------------------------------
```