МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» (Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра технической кибернетики

**Интегральный отчёт по лабораторным работам  
по курсу**

**«Объектно-ориентированное программирование»**

Студент: Борисов Д. С.  
группа 6210  
Преподаватель: Благов А. В.

Самара 2018

Содержание

[Введение 3](#_Toc533113832)

[Лабораторная работа №1 4](#_Toc533113833)

[Лабораторная работа №2 5](#_Toc533113835)

[Лабораторная работа №3 6](#_Toc533113836)

[Лабораторная работа №4 8](#_Toc533113837)

[Лабораторная работа №5 10](#_Toc533113840)

[Лабораторная работа №6 12](#_Toc533113841)

[Лабораторная работа №7 15](#_Toc533113843)

[Лабораторная работа №8 16](#_Toc533113844)

[Заключение 19](#_Toc533113845)

**Введение**

В рамках курса лабораторных работ, состоящего из восьми этапов (по этапу на каждое лабораторное занятие) нужно ознакомиться со структурой исходного кода для Java, а также изучить и апробировать особенности областей видимости и использования пакетов, лексики и базовых конструкций языка, использования интерфейсов и обработки исключительных ситуаций , использования потоков ввода/вывода, в том числе состояния объекта (сериализации /десериализации), создания, перегрузки и переопределения методов (в том числе класса Object), создания и использования конструкций, определенных паттернами проектирования. Необходимый программный функционал должен быть реализован в виде графического приложения со стандартным набором компонент, для расчета ряда результатов должен быть применен многопоточный подход.

**Лабораторная работа №1**

Было осуществлено написание программного кода, состоящего из двух классов, содержащихся в разных пакетах, были изучены особенности областей видимости, использования пакетов и создания jar-архивов.

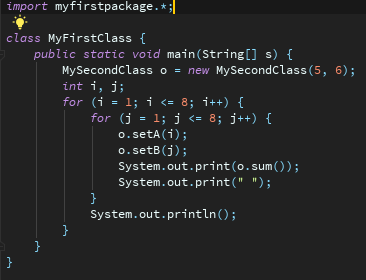


Рисунок 1 - Первый класс

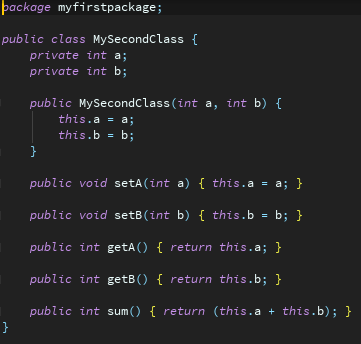


Рисунок 2 - Первое использование пакета

**Лабораторная работа №2**

Был создан набор классов для работы с функциями одной переменной, заданными в табличной форме. В процессе работы был написан класс табулированной точки, имеющий два приватных поля Х и У, конструкторы, методы получения и замены полей, а также класс функции, методы её задания и работы с ней.

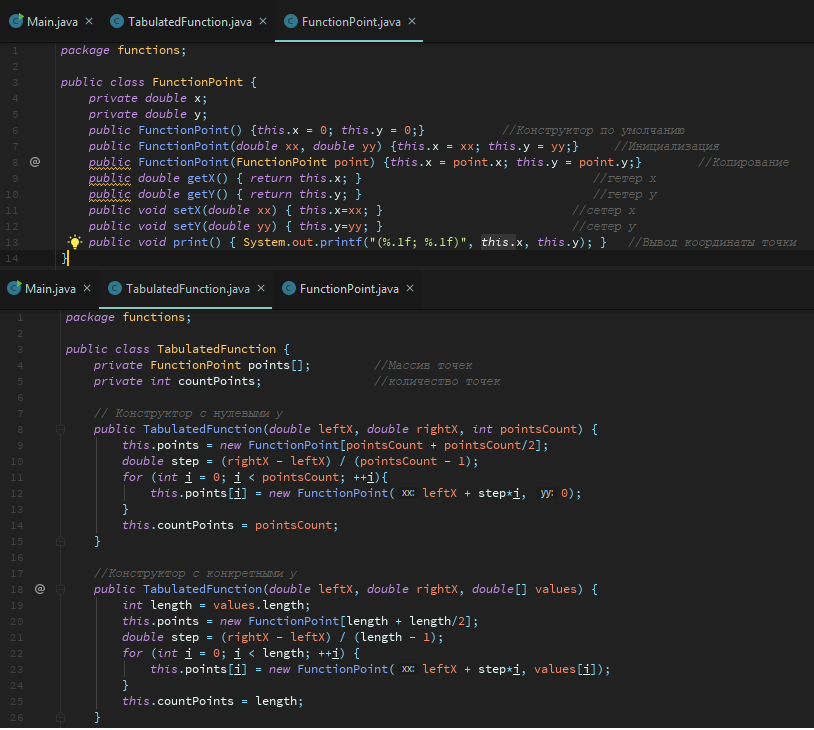


Рисунок 3 - Классы точки и функции

**Лабораторная работа №3**

Созданный в задании 2 пакет для работы с функциями одной переменной, заданными в табличной форме, был дополнен классами исключений, классом функций и базовым интерфейсом.

Важно понимать, что представляет собой связный список в объектно-ориентированном программировании. Действительно, в процедурных языках список обычно представляет собой набор записей (или структур) в памяти, некоторые элементы которых являются указателями на соседние элементы списка, а также набор процедур по работе со списком. В Java, во-первых, нет указателей и адресной арифметики, вместо этого будут использоваться ссылки. Во-вторых, вместо записей будут использоваться объекты. В-третьих, процедуры по работе со списком будут заменены на методы класса.

FunctionPointIndexOutOfBoundsException – исключение выхода за границы набора точек при обращении к ним по номеру, наследует от класса IndexOutOfBoundsException.

InappropriateFunctionPointException – исключение, выбрасываемое при попытке добавления или изменения точки функции несоответствующим образом, наследует от класса Exception.

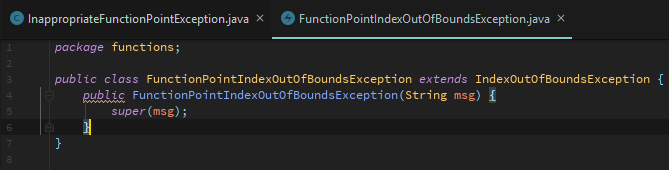


Рисунок 4 - Исключение FunctionPointIndexOutOfBoundsException

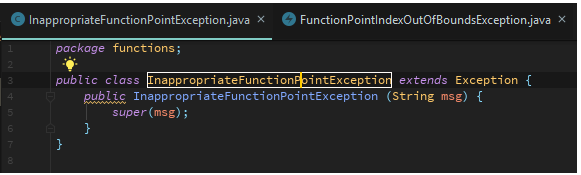


Рисунок 5 - Исключение InappropriateFunctionPointException

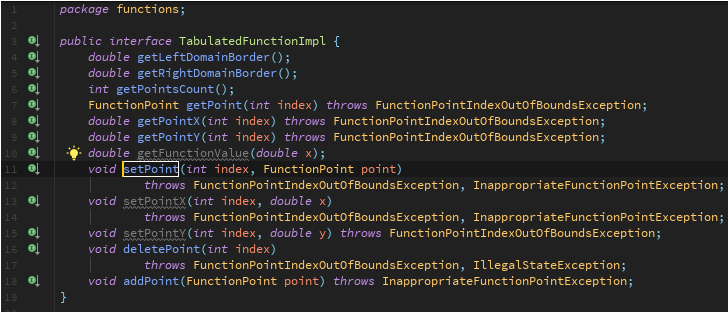


Рисунок 6 - Базовый интерфейс табулированных функций

**Лабораторная работа №4**

В проект были добавлены интерфейсы и классы для аналитически заданных функций, а также методы ввода и вывода табулированных функций, а также добавлена возможность сериализации (преобразование состояния объекта в поток байтов) и десериализации (восстановление состояния объекта из данных потока) объектов. Все классы для работы с функциями реализуют один интерфейс FunctionImpl с тремя методами: получение левой и правой границы области определения, а также значения функции при определённом х. Причём классы Cos, Sin и Tan наследуются от абстрактного класса TrigonometricFunction, который, в свою очередь, реализует интерфейс FunctionImpl. Сделано это для того, чтобы объединить эти три класса, так как у них одинаковая область определения.

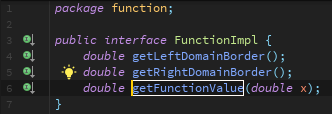


Рисунок 7 - Интерфейс функций

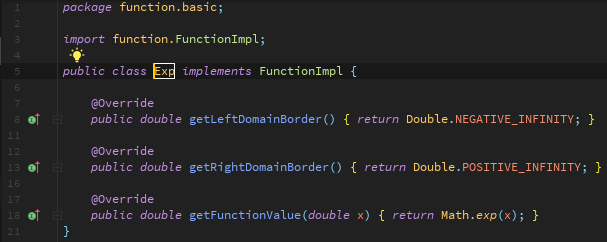


Рисунок 8 - Пример класса функции: экспонента

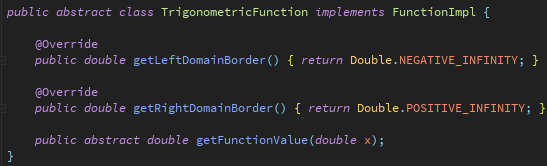


Рисунок 9 - Абстрактный класс TrigonometricFunction

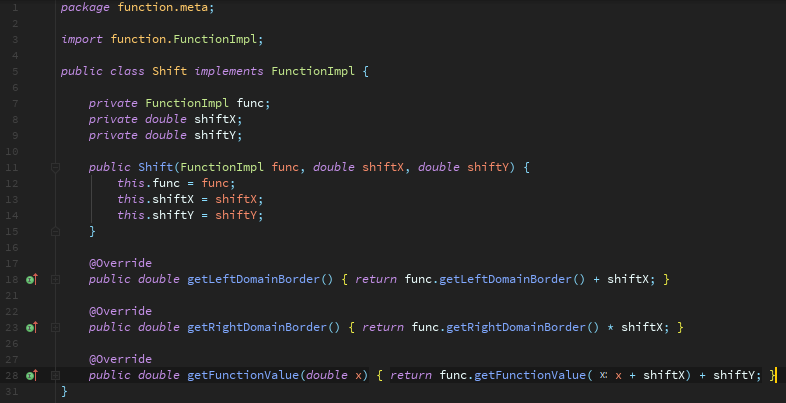


Рисунок 10 - Пример класса для работы с функциями: сдвиг по осям

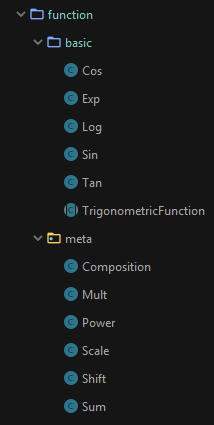


Рисунок 11 - Пакеты с классами для работы с функциями

**Лабораторная работа №5**

Расширены возможности классов, связанных с табулированными функциями за счет переопределения методов, унаследованных из класса Object.

Метод String toString() даёт возможность преобразовывать класс в строку.

Метод boolean equals(Object o) позволяет сравнивать этот класс с объектами.

Метод int hashCode() возвращает индивидуальный код отдельного элемента. Метод Object clone() создаёт копию элемента, а не ссылки.

Ниже приведены переопределённые методы класса Object в классе Array, аналогичным образом эти методы переопределены и в классе List.

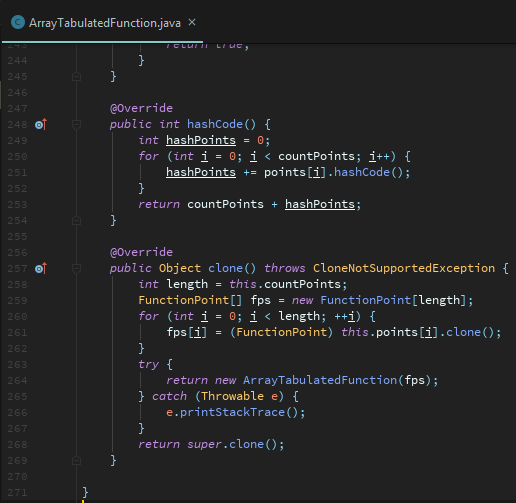


Рисунок 12 - Переопределение методов hashcode и clone

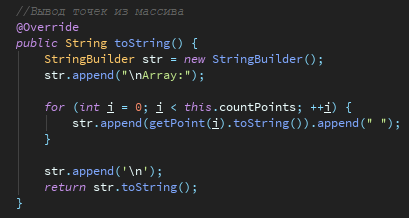


Рисунок 13 - Переопределение метода toString

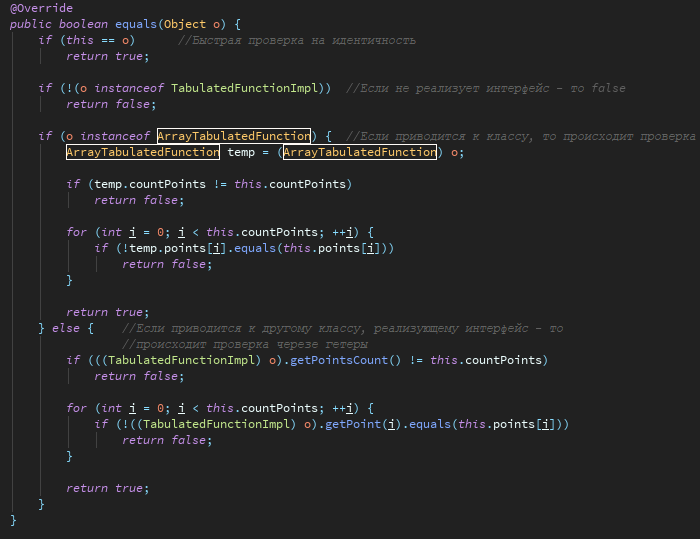


Рисунок 14 - Переопределение метода equals

**Лабораторная работа №6**

Под действующий функционал было создано графическое приложение на платформе SWING. Все окна были созданы с помощью визуального конструктора. Класс ParamTabFunDialog (Рисунок 15) наследуется от класса JDialog, основного класса для создания диалогового окна. При создании этого окна использовался визуальный контруктор. Для работы компонента типа JTable (а именно он будет использоваться для вывода и редактирования табулированной функции) был создан класс FunctionTableModel (Рисунок 17), реализующий TableModel. FunctionTableModel формирует модель для таблицы (Рисунок 18). Для построения табулированной функции путём табулирования обычной функции, указываемой пользователем в виде байт-кода класса функции был создан класс FunctionLoader, расширяющий класс ClassLoader.

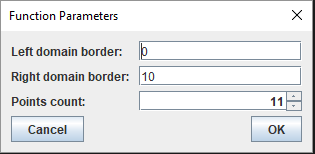


Рисунок 15 - Окно задания параметров табулированной функции

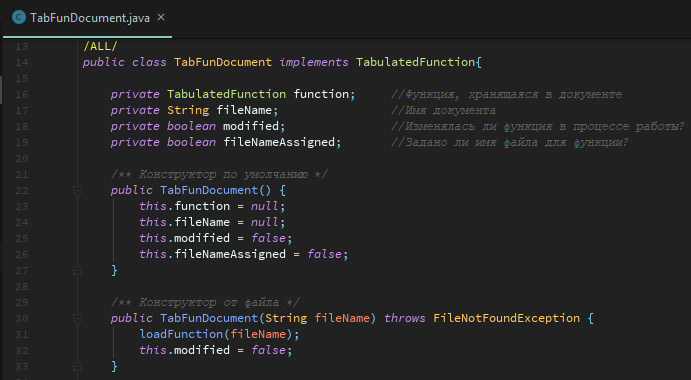


Рисунок 16 - Класс документа с табулированной функцией

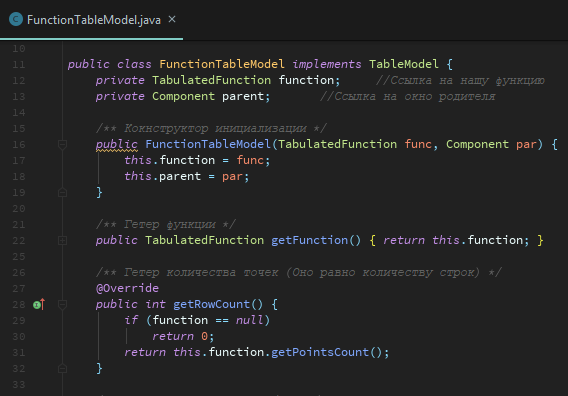


Рисунок 17 - Класс модели для таблицы

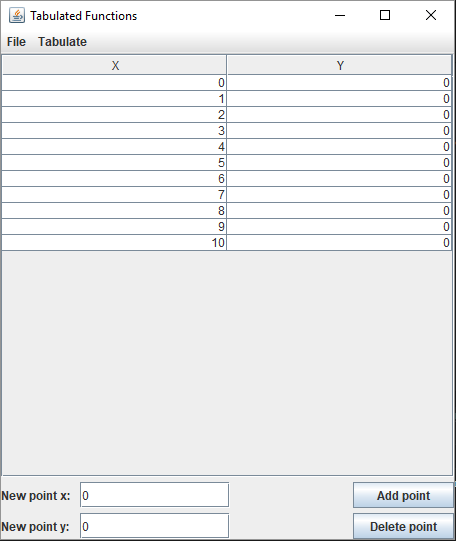


Рисунок 18 - Класс основного окна программы

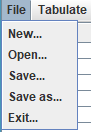


Рисунок 19 - Основное меню программы: вкладка File



Рисунок 20 - Основное меню программы: вкладка Tabulate

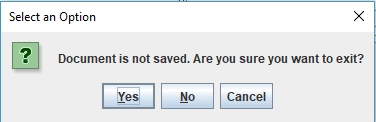


Рисунок 21 - Окно выхода из программы с просьбой сохранить результат

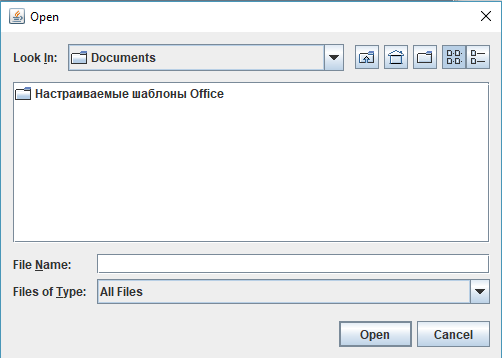


Рисунок 22 - Окно выбора файла

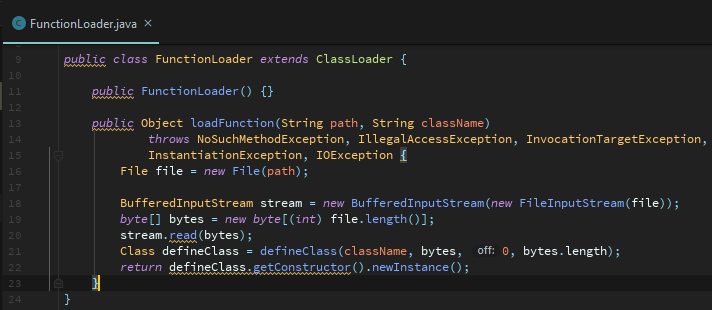


Рисунок 23 - Класс загрузчика классов из указанного файла

**Лабораторная работа №7**

В приложение добавлен функционал многопоточных вычислений. В одном потоке вычислений формируется набор заданий для интегрирования (Рисунок 23), а во втором потоке – вычисляющее значения интегралов функций (Рисунок 24).

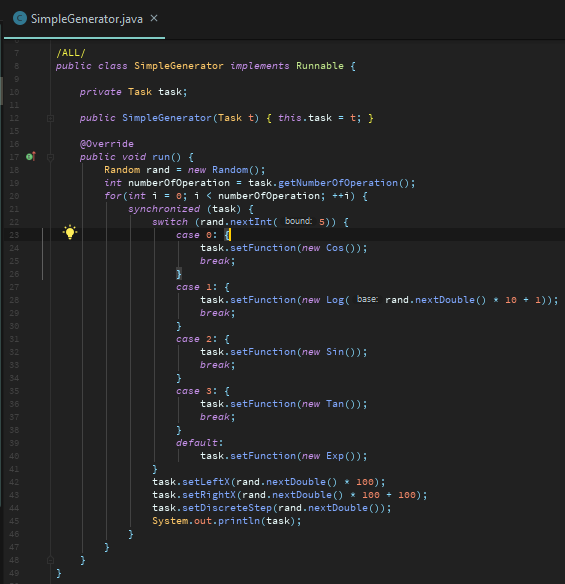


Рисунок 24 - Формирование набора заданий для интегрирования

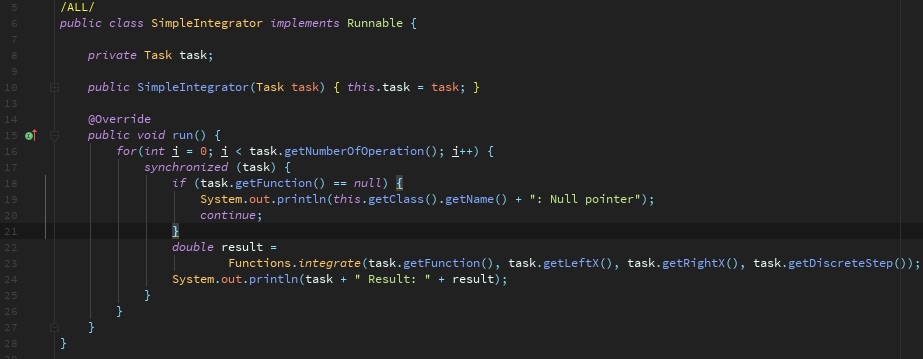


Рисунок 25 - Вычисление интеграла функции

**Лабораторная работа №8**

Внесены 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).

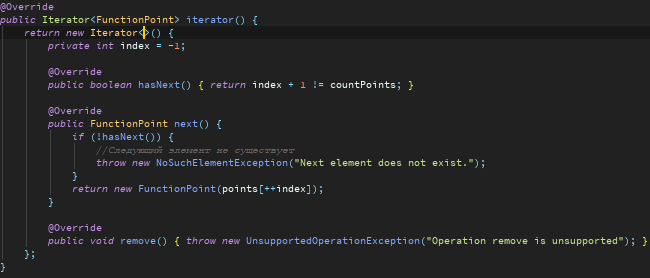


Рисунок 26 - Реализация паттерна «Итератор»

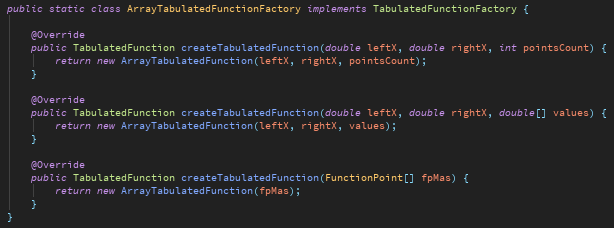


Рисунок 27 - Реализация паттерна «Фабричный метод»

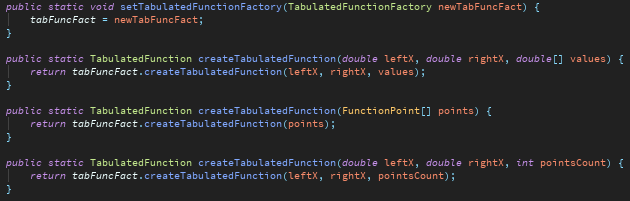


Рисунок 28 - Реализация паттерна «Фабричный метод»

На рисунке 29 - Реализация средств рефлексии

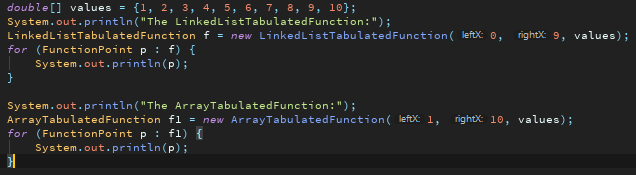


Рисунок 30 - Тестирование паттерна «Итератор»

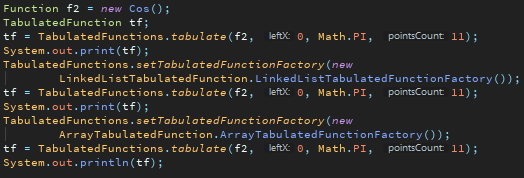


Рисунок 31 - Тестирование паттерна «Фабричный метод»

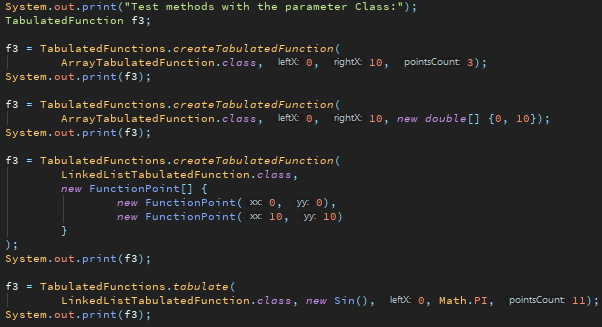


Рисунок 32 - Тестирование средств рефлексии

**Заключение**

В процессе выполнения заданий произведено ознакомление со структурой исходного кода объектно-ориентированного языка высокого уровня Java, с базовыми классами и стандартной библиотекой. Изучены и апробированы особенности областей видимости и использования пакетов, лексики и базовых конструкций языка, использования интерфейсов и обработки исключительных ситуаций , использования потоков ввода/вывода, в том числе состояния объекта (сериализации /десериализации), создания, перегрузки и переопределения методов (в том числе класса Object), создания и использования конструкций, определенных паттернами проектирования. Необходимый программный функционал реализован в виде графического приложения со стандартным набором компонент. Для вычисления результатов применен многопоточный подход, в том числе регулируемый с помощью объекта семафора.

При написании программного приложения были выполнены следующие требования:

- продемонстрировано владение языком Java, его стандартной библиотекой, для создания нетривиальной реализации необходимых методов и алгоритмов;

- необходимый функционал реализован, основываясь на таких базовых парадигмах как инкапсуляция, наследование и полиморфизм,

- выполнено соответствие принципам SOLID,

- программный код написан в полном соответствии с правилами именования и комментариями, в том числе с применением технологии Javadoc.