**ЛР 3. Полиморфизм. Форматированный вывод**

**Цель**: приобрести практические навыки разработки программ с использованием полиморфизма; изучить особенности перегрузки и переопределения методов; ознакомиться со средствами консольного форматированного вывода текста.

**1  Индивидуальное задание**

В качестве основы использовать исходный текст проекта предыдущей лабораторной работы.

Текст предыдущей лабы: Найти двоичное представление целочисленного значения полной энергии физического тела при заданных значениях массы, скорости и высоты.

Используя шаблон проектирования Factory Method (Virtual Constructor), расширить иерархию производными классами, реализующими методы для представления результатов в виде текстовой таблицы. Параметры отображения таблицы должны определяться пользователем.

Продемонстрировать замещение (переопределение, overriding), совмещение (перегрузка, overloading), динамическое назначение методов (позднее связывание, полиморфизм, dynamic method dispatch).

Обеспечить диалоговый интерфейс с пользователем.

Разработать класс для тестирования основной функциональности.

Использовать комментарии для автоматической генерации документации средствами javadoc.

**2  Пример проекта**

**2.1  Разработка программы**

Реализуем классы, структура которых соответствует схеме п.2.1.2.

Разработаем класс MainTest для проведения теста основной функциональности кода. Реализуем методы:

testCalc() – для проверки основной функциональности класса ViewTable.

testRestore() – для проверки корректности восстановления данных при сериализации.

В процессе разработки необходимо обеспечить прохождение всех тестов.

**2.1.1  Используемые средства ООП**

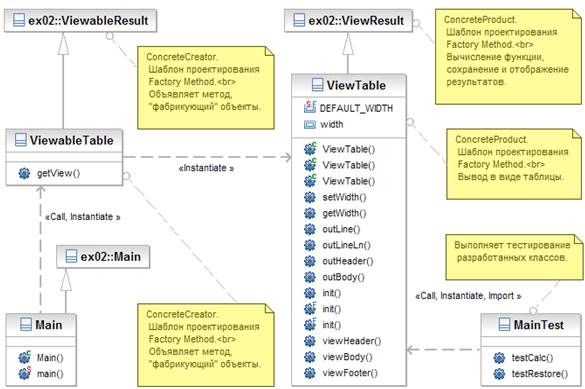
Метод называют абстрактным, если он имеет только объявление и не имеет реализации. Класс, содержащий абстрактные методы, называется абстрактным классом. Если класс содержит один или больше абстрактных методов, этот класс должен быть определен как abstract.

Все методы дочерних классов, совпадающие с объявлением сигнатуры в базовом, выполняют переопределение (замещение, overriding) и используют динамическое связывание. Если имя метода совпадает с именем метода базового класса, а аргументы различны, то это означает перегрузку (совмещение, overloading).

Полиморфизм позволяет повысить возможности по организации кода и читабельность исходных текстов, облегчить создание расширяемых программ, предполагающих дальнейшее наращивание функциональности.

**2.1.2  Иерархия и структура классов**

Структура классов и схема их отношений:



**2.1.3  Описание программы**

При использовании полиморфизма (позднего связывания) для создания конструкторов руководствуемся следующим принципом: производим в конструкторе как можно меньше действий, стараемся исключить вызов каких-либо методов.

При разработке классов используем наследование и композицию. При вызове методов объектов используется позднее связывание.

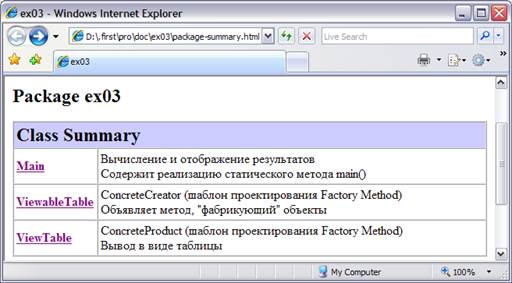
Расширим ex02.ViewResult и ex02.ViewableResult классами ViewTable и ViewableTable соответственно для вывода содержимого коллекции в виде текстовой таблицы.

При написании исходного кода используем стиль комментариев документации javadoc.

Структура проекта:

|  |  |
| --- | --- |
| Папка src  http://cde.kpi.kharkov.ua/courses/course_285/Lb3/r_0_files/image002.jpg | Папка test  http://cde.kpi.kharkov.ua/courses/course_285/Lb3/r_0_files/image003.png |

Выполним генерацию документации:



После проверки работоспособности готовой программы, создадим исполняемый JAR файл ex03.jar

**2.2  Текст программы**

**2.2.1  Main.java**

**package** ex03;

**import** ex02.View;

/\*\* Вычисление и отображение результатов<br>

 \* Содержит реализацию статического метода main()

 \* **@author** xone

 \* **@version** 3.0

 \* **@see** Main#main

 \*/

**public** **class** Main **extends** ex02.Main {

/\*\* Инициализирует поле {@linkplain ex02.Main#view view} \*/

**public** Main(View view) {

**super**(view);

}

/\*\* Выполняется при запуске программы;

 \* вызывает метод {@linkplain ex02.Main#menu menu()}

 \* **@param** args - параметры запуска программы

 \*/

**public** **static** **void** main(String[] args) {

       Main main = **new** Main(**new** ViewableTable().getView());

       main.menu();

}

}

**2.2.2  ViewableTable.java**

**package** ex03;

**import** ex02.ViewableResult;

**import** ex02.View;

/\*\* ConcreteCreator

 \* (шаблон проектирования

 \* Factory Method)<br>

 \* Объявляет метод,

 \* "фабрикующий" объекты

 \* **@author** xone

 \* **@version** 1.0

 \* **@see** ViewableResult

 \* **@see** ViewableTable#getView()

\*/

**public** **class** ViewableTable **extends** ViewableResult {

/\*\* Создаёт отображаемый объект {@linkplain ViewTable} \*/

@Override

**public** View getView() {

**return** **new** ViewTable();

}

}

**2.2.3  ViewTable.java**

**package** ex03;

**import** java.util.Formatter;

**import** ex01.Item2d;

**import** ex02.ViewResult;

/\*\* ConcreteProduct

 \* (шаблон проектирования

 \* Factory Method)<br>

 \* Вывод в виде таблицы

 \* **@author** xone

 \* **@version** 1.0

 \* **@see** ViewResult

 \*/

**public** **class** ViewTable **extends** ViewResult {

/\*\* Определяет ширину таблицы по умолчанию \*/

**private** **static** **final** **int** *DEFAULT\_WIDTH* = 20;

/\*\* Текущая ширина таблицы \*/

**private** **int** width;

/\*\* Устанавливает {@linkplain ViewTable#width width}

 \* значением {@linkplain ViewTable#DEFAULT\_WIDTH DEFAULT\_WIDTH}<br>

 \* Вызывается конструктор суперкласса {@linkplain ViewResult#ViewResult() ViewResult()}

 \*/

**public** ViewTable() {

       width = *DEFAULT\_WIDTH*;

}

/\*\* Устанавливает {@linkplain ViewTable#width} значением <b>width</b><br>

 \* Вызывается конструктор суперкласса {@linkplain ViewResult#ViewResult() ViewResult()}

 \* **@param** width определяет ширину таблицы

 \*/

**public** ViewTable(**int** width) {

**this**.width = width;

}

/\*\* Устанавливает {@linkplain ViewTable#width} значением <b>width</b><br>

 \* Вызывается конструктор суперкласса {@linkplain ViewResult#ViewResult(int n) ViewResult(int n)}

 \* **@param** width определяет ширину таблицы

 \* **@param** n количество элементов коллекции; передаётся суперконструктору

 \*/

**public** ViewTable(**int** width, **int** n) {

**super**(n);

**this**.width = width;

}

/\*\* Устанавливает поле {@linkplain ViewTable#width} значением <b>width</b>

 \* **@param** width новая ширина таблицы

 \* **@return** заданная параметром <b>width</b> ширина таблицы

 \*/

**public** **int** setWidth(**int** width) {

**return** **this**.width = width;

}

/\*\* Возвращает значение поля {@linkplain ViewTable#width}

 \* **@return** текущая ширина таблицы

 \*/

**public** **int** getWidth() {

**return** width;

}

/\*\* Выводит вертикальный разделитель шириной {@linkplain ViewTable#width} символов \*/

**private** **void** outLine() {

**for**(**int** i = width; i > 0; i--) {

              System.*out*.print('-');

       }

}

/\*\* Вызывает {@linkplain ViewTable#outLine()}; завершает вывод разделителем строки \*/

**private** **void** outLineLn() {

       outLine();

       System.*out*.println();

}

/\*\* Выводит заголовок таблицы шириной {@linkplain ViewTable#width} символов \*/

**private** **void** outHeader() {

       Formatter fmt = **new** Formatter();

       fmt.format("%s%d%s%2$d%s", "%", (width-3)/2, "s | %", "s\n");

       System.*out*.printf(fmt.toString(), "x    ", "y    ");

}

/\*\* Выводит тело таблицы шириной {@linkplain ViewTable#width} символов \*/

**private** **void** outBody() {

       Formatter fmt = **new** Formatter();

       fmt.format("%s%d%s%2$d%s", "%", (width-3)/2, ".0f | %", ".3f\n");

**for**(Item2d item : getItems()) {

              System.*out*.printf(fmt.toString(), item.getX(), item.getY());

       }

}

/\*\* Перегрузка (совмещение, overloading) метода суперкласса;

 \* устанавливает поле {@linkplain ViewTable#width} значением <b>width</b><br>

 \* Вызывает метод {@linkplain ViewResult#viewInit() viewInit()}

 \* **@param** width новая ширина таблицы

 \*/

**public** **final** **void** init(**int** width) { // method overloading

**this**.width = width;

       viewInit();

}

/\*\* Перегрузка метода суперкласса;

 \* устанавливает поле {@linkplain ViewTable#width} значением <b>width</b><br>

 \* Для объекта {@linkplain ViewTable} вызывает метод {@linkplain ViewTable#init(double stepX)}

 \* **@param** width новая ширина таблицы.

 \* **@param** stepX передается методу <b>init(double)</b>

 \*/

**public** **final** **void** init(**int** width, **double** stepX) { // method overloading

**this**.width = width;

       init(stepX);

}

/\*\* Переопределение (замещение, overriding) метода суперкласса;

 \* выводит информационное сообщение и вызывает метод суперкласса

 \* {@linkplain ViewResult#init(double stepX) init(double stepX)}<br>

 \* {@inheritDoc}

 \*/

@Override

**public** **void** init(**double** stepX) { // method overriding

       System.*out*.print("Initialization... ");

**super**.init(stepX);

       System.*out*.println("done. ");

}

/\*\* Вывод элемента таблицы<br>{@inheritDoc} \*/

@Override

**public** **void** viewHeader() {

       outHeader();

       outLineLn();

}

/\*\* Вывод элемента таблицы<br>{@inheritDoc} \*/

@Override

**public** **void** viewBody() {

       outBody();

}

/\*\* Вывод элемента таблицы<br>{@inheritDoc} \*/

@Override

**public** **void** viewFooter() {

       outLineLn();

}

}

**2.2.4  MainTest.java**

**package** ex03;

**import** org.junit.Test;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*assertTrue*;

**import** junit.framework.Assert;

**import** java.io.IOException;

**import** ex01.Item2d;

/\*\* Выполняет тестирование

 \* разработанных классов.

 \* **@author** xone

 \* **@version** 3.0

 \*/

**public** **class** MainTest {

/\*\* Проверка основной функциональности класса {@linkplain ViewTable} \*/

@Test

**public** **void**   testCalc() {

       ViewTable tbl = **new** ViewTable(10, 5);

*assertEquals*(10, tbl.getWidth());

*assertEquals*(5, tbl.getItems().size());

       tbl.init(40, 90.0);

       Item2d item = **new** Item2d();

**int** ctr = 0;

       item.setXY(0.0, 0.0);

*assertTrue*("expected:<" + item + "> but was:<" + tbl.getItems().get(ctr) + ">", tbl.getItems().get(ctr).equals(item));

       ctr++;

       item.setXY(90.0, 1.0);

*assertTrue*("expected:<" + item + "> but was:<" + tbl.getItems().get(ctr) + ">", tbl.getItems().get(ctr).equals(item));

       ctr++;

       item.setXY(180.0, 0.0);

*assertTrue*("expected:<" + item + "> but was:<" + tbl.getItems().get(ctr) + ">", tbl.getItems().get(ctr).equals(item));

       ctr++;

       item.setXY(270.0, -1.0);

*assertTrue*("expected:<" + item + "> but was:<" + tbl.getItems().get(ctr) + ">", tbl.getItems().get(ctr).equals(item));

       ctr++;

       item.setXY(360.0, 0.0);

*assertTrue*("expected:<" + item + "> but was:<" + tbl.getItems().get(ctr) + ">", tbl.getItems().get(ctr).equals(item));

}

/\*\* Проверка сериализации. Корректность восстановления данных. \*/

@Test

**public** **void**   testRestore() {

       ViewTable tbl1 = **new** ViewTable(10, 1000);

       ViewTable tbl2 = **new** ViewTable();

       // Вычислим значение функции со случайным шагом приращения аргумента

       tbl1.init(30, Math.*random*()\*100.0);

       //     Сохраним коллекцию tbl1.items

**try** {

              tbl1.viewSave();

       }  **catch** (IOException e) {

              Assert.*fail*(e.getMessage());

       }

       //     Загрузим коллекцию tbl2.items

**try** {

              tbl2.viewRestore();

       } **catch** (Exception e) {

              Assert.*fail*(e.getMessage());

       }

       //     Должны загрузить столько же элементов, сколько сохранили

*assertEquals*(tbl1.getItems().size(), tbl2.getItems().size());

       //     Причем эти элементы должны быть равны.

       //     Для этого нужно определить метод equals

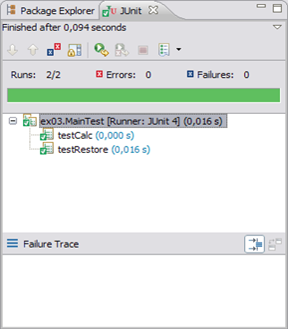
*assertTrue*("containsAll()", tbl1.getItems().containsAll(tbl2.getItems()));

}

}

**2.3  Результаты тестирования**

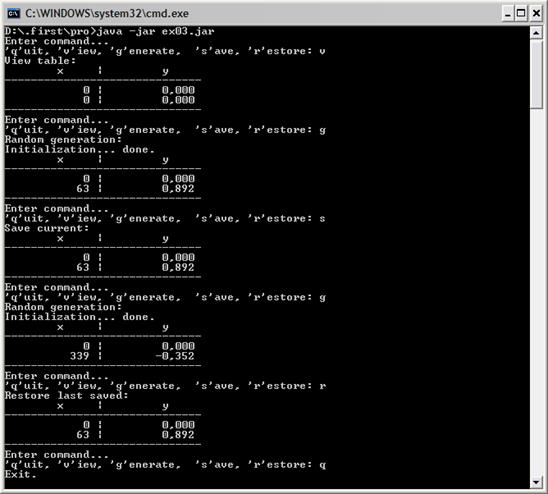
Выполним ex03.MainTest как JUnit Test



Выполним запуск программы из командной строки:

java -jar ex03.jar

В результате выполнения получим:



**3  Заключение**

Разработали программу решения задачи индивидуального задания. Результаты тестирования подтверждают корректность используемых алгоритмов.

При реализации иерархии использовался шаблон проектирования Factory Method. Выполнена перегрузка и переопределение методов. Продемонстрирован полиморфизм.

При отображении результатов использовались средства форматированного вывода текста на консоль.

Для тестирования программы использовались средства J