**ЛР 2. Интерфейсы. Наследование. Коллекции пакета java.util**

**Цель**: приобрести практические навыки разработки программ в среде Eclipse с использованием основных принципов ООП; изучить методы организации коллекций объектов средствами пакета java.util.

**1  Индивидуальное задание**

В качестве основы использовать исходный текст проекта предыдущей лабораторной работы. Обеспечить размещение результатов вычислений в коллекции с возможностью сохранения/восстановления.

Задание в предыдущей лабе: 15.    Найти двоичное представление целочисленного значения полной энергии физического тела при заданных значениях массы, скорости и высоты.

Используя шаблон проектирования Factory Method (Virtual Constructor), разработать иерархию, предусматривающую расширение за счёт добавления новых отображаемых классов.

Расширить иерархию интерфейсом "фабрикуемых" объектов, представляющим набор методов для отображения результатов вычислений. Реализовать эти методы для вывода результатов в текстовом виде. Разработать и реализовать интерфейс для "фабрикующего" метода.

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 тест, проверяющий основную функциональность кода. Реализуем методы:

testCalc() – для проверки основной функциональности класса ViewResult.

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

В процессе разработки необходимо обеспечить прохождение всех тестов.

**2.1.1  Используемые средства ООП**

При разработке классов используем наследование и композицию.

Интерфейс – это явно указанная спецификация набора методов, которые должны быть представлены в классе, реализующем эту спецификацию. Реализация этих методов в интерфейсе отсутствует. Интерфейсы можно многократно наследовать. Конкретный класс может быть наследником лишь одного суперкласса, но в нем может быть реализовано неограниченное число интерфейсов.

Коллекция – группа индивидуальных элементов, часто с определенными правилами, применяемыми к элементам. Список должен хранить элементы в определенной последовательности. Набор не может иметь дублирующиеся элементы. Карта – группа объектных пар ключ-значение. Карта может возвращать набор своих ключевых значений, коллекцию своих значений или набор своих пар.

Приступая к разработке приложения, далеко не всегда можно заранее решить, какие именно компоненты вам понадобятся. Обычно у разработчика есть лишь общее видение того, что должны делать компоненты, но реализация функциональности компонентов с уточнением их возможностей выполняется позже, в ходе работы над проектом.

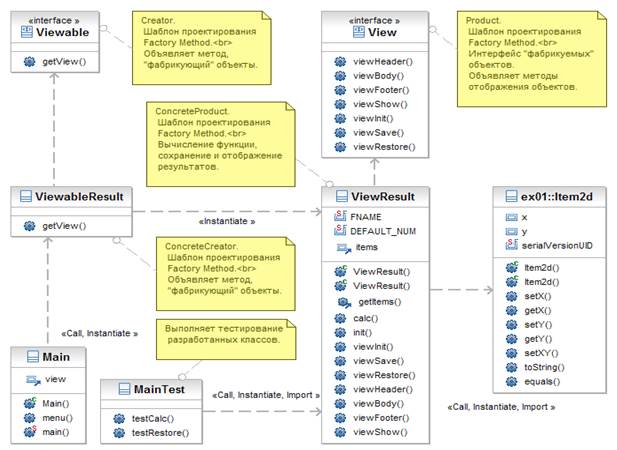
Частично данную проблему решает использование интерфейсов, описывающих подобные компоненты. Применение интерфейсов затрудняет работу программиста, так как из интерфейса невозможно создать объект. Для того чтобы получить объект, нужно реализовать класс. Поэтому, вместо того, чтобы разрабатывать реализацию некоего класса, специфичного для конкретного приложения, можно просто вычленить из класса функциональность конструктора и реализовать ее в виде специального метода, "фабрикующего" объекты приложения.

Таким образом можно получить некий класс, например ConcreteCreator, отвечающий за создание определенных объектов. Этот класс предназначается для создания экземпляров реализации (ConcreteProduct) определенного интерфейса (Product).

Производящий шаблон проектирования Factory Method (также известен как Virtual Constructor) определяет стандартный метод создания объекта, не связанный с вызовом конструктора, оставляя решение о том, какой именно объект создавать, за подклассами. Он "фабрикует" объекты, когда в них возникает необходимость.

**2.1.2  Иерархия и структура классов**

Структура классов и схема их отношений:



**2.1.3  Описание программы**

Разработаем интерфейсы View и Viewable для представления методов отображения (вывода) классов и создания отображаемого объекта. Реализуем эти интерфейсы в классах ViewResult и ViewableResult соответственно.

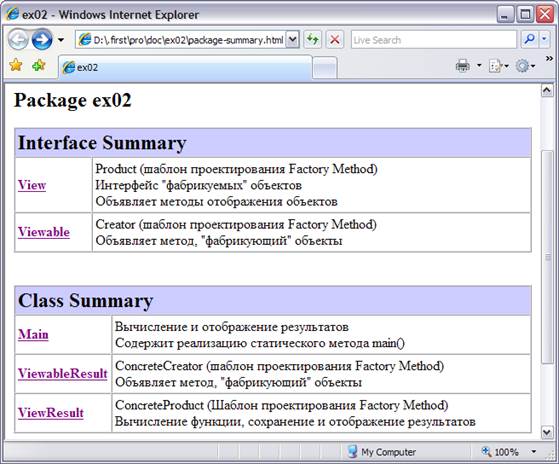
Результаты и исходные данные будем хранить в списке ArrayList<Item2d>.

При написании исходного кода используем стиль комментариев документации javadoc.

Структура проекта:

|  |  |
| --- | --- |
| Папка src  http://cde.kpi.kharkov.ua/courses/course_285/Lb2/r_0_files/image002.png | Папка test  http://cde.kpi.kharkov.ua/courses/course_285/Lb2/r_0_files/image003.png |

Выполним генерацию документации:



После проверки работоспособности готовой программы, создадим исполняемый JAR файл ex02.jar

**2.2  Текст программы**

**2.2.1  Main.java**

**package** ex02;

**import** java.io.IOException;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

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

 \* Содержит реализацию статического метода main()

 \* **@author** xone

 \* **@version** 2.0

 \* **@see** Main#main

 \*/

**public** **class** Main {

/\*\* Объект, реализующий интерфейс {@linkplain View};

 \* обслуживает коллекцию объектов {@linkplain ex01.Item2d}

 \*/

**private** View view;

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

**public** Main(View view) {

**this**.view = view;

}

/\*\* Отображает меню. \*/

**protected** **void** menu() {

       String s = **null**;

       BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.*in*));

**do** {

**do** {

                     System.*out*.println("Enter command...");

                     System.*out*.print("'q'uit, 'v'iew, 'g'enerate,  's'ave, 'r'estore: ");

**try** {

                            s = in.readLine();

                     } **catch**(IOException e) {

                            System.*out*.println("Error: " + e);

                            System.*exit*(0);

                     }

              } **while** (s.length() != 1);

**switch** (s.charAt(0)) {

**case** 'q':

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

**break**;

**case** 'v':

                     System.*out*.println("View current.");

                     view.viewShow();

**break**;

**case** 'g':

                     System.*out*.println("Random generation.");

                     view.viewInit();

                     view.viewShow();

**break**;

**case** 's':

                     System.*out*.println("Save current.");

**try** {

                            view.viewSave();

                     } **catch** (IOException e) {

                            System.*out*.println("Serialization error: " + e);

                     }

                     view.viewShow();

**break**;

**case** 'r':

                     System.*out*.println("Restore last saved.");

**try** {

                            view.viewRestore();

                     } **catch** (Exception e) {

                            System.*out*.println("Serialization error: " + e);

                     }

                     view.viewShow();

**break**;

**default**:

                     System.*out*.println("Wrong command.");

              }

       } **while**(s.charAt(0) != 'q');

}

/\*\* Выполняется при запуске программы;

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

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

 \*/

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

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

       main.menu();

}

}

**2.2.2  View.java**

**package** ex02;

**import** java.io.IOException;

/\*\* Product

 \* (шаблон проектирования

 \* Factory Method)<br>

 \* Интерфейс "фабрикуемых"

 \* объектов<br>

 \* Объявляет методы

 \* отображения объектов

 \* **@author** xone

 \* **@version** 1.0

 \*/

**public** **interface** View {

/\*\* Отображает заголовок \*/

**public** **void** viewHeader();

/\*\* Отображает основную часть \*/

**public** **void** viewBody();

/\*\* Отображает окончание \*/

**public** **void** viewFooter();

/\*\* Отображает объект целиком \*/

**public** **void** viewShow();

/\*\* Выполняет инициализацию \*/

**public** **void** viewInit();

/\*\* Сохраняет данные для последующего восстановления \*/

**public** **void** viewSave() **throws** IOException;

/\*\* Восстанавливает ранее сохранённые данные \*/

**public** **void** viewRestore() **throws** Exception;

}

**2.2.3  Viewable.java**

**package** ex02;

/\*\* Creator

 \* (шаблон проектирования

 \* Factory Method)<br>

 \* Объявляет метод,

 \* "фабрикующий" объекты

 \* **@author** xone

 \* **@version** 1.0

 \* **@see** Viewable#getView()

 \*/

**public** **interface** Viewable {

/\*\* Создаёт объект, реализующий {@linkplain View} \*/

**public** View getView();

}

**2.2.4  ViewableResult.java**

**package** ex02;

/\*\* ConcreteCreator

 \* (шаблон проектирования

 \* Factory Method)<br>

 \* Объявляет метод,

 \* "фабрикующий" объекты

 \* **@author** xone

 \* **@version** 1.0

 \* **@see** Viewable

 \* **@see** ViewableResult#getView()

 \*/

**public** **class** ViewableResult **implements** Viewable {

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

@Override

**public** View getView() {

**return** **new** ViewResult();

}

}

**2.2.5  ViewResult.java**

**package** ex02;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.util.ArrayList;

**import** ex01.Item2d;

/\*\* ConcreteProduct

 \* (Шаблон проектирования

 \* Factory Method)<br>

 \* Вычисление функции,

 \* сохранение и отображение

 \* результатов

 \* **@author** xone

 \* **@version** 1.0

 \* **@see** View

 \*/

**public** **class** ViewResult **implements** View {

/\*\* Имя файла, используемое при сериализации \*/

**private** **static** **final** String *FNAME* = "items.bin";

/\*\* Определяет количество значений для вычисления по умолчанию \*/

**private** **static** **final** **int** *DEFAULT\_NUM* = 10;

/\*\* Коллекция аргументов и результатов вычислений \*/

**private** ArrayList<Item2d> items = **new** ArrayList<Item2d>();

/\*\* Вызывает {@linkplain ViewResult#ViewResult(int n) ViewResult(int n)}

 \* с параметром {@linkplain ViewResult#DEFAULT\_NUM DEFAULT\_NUM}

 \*/

**public** ViewResult() {

**this**(*DEFAULT\_NUM*);

}

/\*\* Инициализирует коллекцию {@linkplain ViewResult#items}

 \* **@param** n начальное количество элементов

 \*/

**public** ViewResult(**int** n) {

**for**(**int** ctr = 0; ctr < n; ctr++) {

              items.add(**new** Item2d());

       }

}

/\*\* Получить значение {@linkplain ViewResult#items}

 \* **@return** текущее значение ссылки на объект {@linkplain ArrayList}

 \*/

**public** ArrayList<Item2d> getItems() {

**return** items;

}

/\*\* Вычисляет значение функции

 \* **@param** x аргумент вычисляемой функции

 \* **@return** результат вычисления функции

 \*/

**private** **double** calc(**double** x) {

**return** Math.*sin*(x \* Math.*PI* / 180);

}

/\*\* Вычисляет значение функции и сохраняет

 \* результат в коллекции {@linkplain ViewResult#items}

 \* **@param** stepX шаг приращения аргумента

 \*/

**public** **void** init(**double** stepX) {

**double** x = 0.0;

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

              item.setXY(x, calc(x));

              x += stepX;

       }

}

/\*\* Вызывает <b>init(double stepX)</b> со случайным значением аргумента<br>

 \* {@inheritDoc}

 \*/

@Override

**public** **void** viewInit() {

       init(Math.*random*() \* 360.0);

}

/\*\* Реализация метода {@linkplain View#viewSave()}<br>

 \* {@inheritDoc}

 \*/

@Override

**public** **void** viewSave() **throws** IOException {

       ObjectOutputStream os = **new** ObjectOutputStream(**new** FileOutputStream(*FNAME*));

       os.writeObject(items);

       os.flush();

       os.close();

}

/\*\* Реализация метода {@linkplain View#viewRestore()}<br>

 \* {@inheritDoc}

 \*/

@SuppressWarnings("unchecked")

@Override

**public** **void** viewRestore() **throws** Exception {

       ObjectInputStream is = **new** ObjectInputStream(**new** FileInputStream(*FNAME*));

       items = (ArrayList<Item2d>) is.readObject();

       is.close();

}

/\*\* Реализация метода {@linkplain View#viewHeader()}<br>

 \* {@inheritDoc}

 \*/

@Override

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

       System.*out*.println("Results:");

}

/\*\* Реализация метода {@linkplain View#viewBody()}<br>

 \* {@inheritDoc}

 \*/

@Override

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

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

              System.*out*.printf("(%.0f; %.3f) ", item.getX(), item.getY());

       }

       System.*out*.println();

}

/\*\* Реализация метода {@linkplain View#viewFooter()}<br>

 \* {@inheritDoc}

 \*/

@Override

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

       System.*out*.println("End.");

}

/\*\* Реализация метода {@linkplain View#viewShow()}<br>

 \* {@inheritDoc}

 \*/

@Override

**public** **void** viewShow() {

       viewHeader();

       viewBody();

       viewFooter();

}

}

**2.2.6  MainTest.java**

**package** ex02;

**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** 2.0

 \*/

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

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

@Test

**public** **void**   testCalc() {

       ViewResult view = **new** ViewResult(5);

       view.init(90.0);

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

**int** ctr = 0;

       item.setXY(0.0, 0.0);

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

       ctr++;

       item.setXY(90.0, 1.0);

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

       ctr++;

       item.setXY(180.0, 0.0);

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

       ctr++;

       item.setXY(270.0, -1.0);

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

       ctr++;

       item.setXY(360.0, 0.0);

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

}

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

@Test

**public** **void**   testRestore() {

       ViewResult view1 = **new** ViewResult(1000);

       ViewResult view2 = **new** ViewResult();

       // Вычислим значение функции со случайным шагом приращения аргумента

       view1.init(Math.*random*()\*100.0);

       //     Сохраним коллекцию view1.items

**try** {

              view1.viewSave();

       }  **catch** (IOException e) {

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

       }

       //     Загрузим коллекцию view2.items

**try** {

              view2.viewRestore();

       } **catch** (Exception e) {

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

       }

       //     Должны загрузить столько же элементов, сколько сохранили

*assertEquals*(view1.getItems().size(), view2.getItems().size());

       //     Причем эти элементы должны быть равны.

       //     Для этого нужно определить метод equals

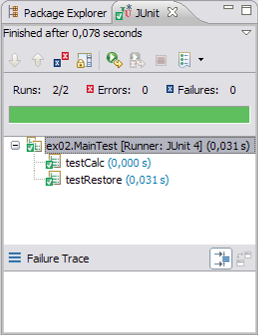
*assertTrue*("containsAll()", view1.getItems().containsAll(view2.getItems()));

}

}

**2.3  Результаты тестирования**

Выполним ex02.MainTest как JUnit Test



Выполним запуск программы из командной строки:

java -jar ex02.jar

В результате выполнения получим:



**3  Заключение**

При создании классов использовали наследование и композицию. Приобрели практические навыки разработки программ в среде Eclipse с использованием средств пакета java.util для организации коллекции объектов.

Использовали шаблон проектирования Factory Method.

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