**Тема:     Классы и объекты. Композиция. Сериализация**

**ЛР 1. Классы и объекты. Композиция. Сериализация**

**Цель**: ознакомиться со средой разработки программ на языке Java; приобрести практические навыки создания и отладки программ; использовать композицию при разработке классов.

**1  Индивидуальное задание**

Выбрать вариант прикладной задачи согласно номеру по списку в журнале академической группы.

На языке Java разработать сериализируемый класс для хранения параметров и результатов вычислений. Используя композицию, разработать класс для нахождения решения задачи.

Разработать класс для демонстрации в диалоговом режиме сохранения и восстановления состояния объекта, используя сериализацию. Показать особенности использования *transient* полей.

Разработать класс для тестирования корректности результатов вычислений.

Использовать комментарии для автоматической генерации документации средствами javadoc.

15.    Найти двоичное представление целочисленного значения полной энергии физического тела при заданных значениях массы, скорости и высоты.

**2  Структура отчёта**

После выполнения лабораторной работы необходимо составить отчет – документ Microsoft Word ­– техническое описание разработанной программы. Все страницы, кроме титульного листа должны быть пронумерованы (номер – в правом верхнем углу). Отчет должен быть оформлен на **украинском языке** и иметь следующую структуру:

**Титульный лист.** Оформляется для каждого отчета. Шрифт: Times New Roman, размер шрифта: 14, интервал: полуторный. Содержит следующие данные:

|  |
| --- |
|            ЗВІТз лабораторної роботи №      по курсуОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ Виконавст. гр. КІТ                 (дата)    (підпис)   (Фамілія та ініціали)              20              |

**Содержание.** На отдельном листе приводится СОДЕРЖАНИЕ отчета – список разделов с указанием номеров страниц. Шрифт: Times New Roman, размер шрифта: 12, интервал: полуторный.

**Заголовок отчета.** С нового листа начинается описательная часть. В заголовке (Шрифт: Times New Roman, размер шрифта: 12, интервал: одиночный, выравнивание: по центру) указывается номер работы и тема, например:

Лабораторная работа №1

Интегрированная среда разработки Eclipse

**Цель выполнения лабораторной работы.** После заголовка указывается цель выполнения лабораторной работы (Шрифт: Times New Roman, размер шрифта: 12, интервал: одиночный, выравнивание: по ширине, отступ: 1.25 см), например:

Цель: ознакомиться со средой разработки; на примере простейшего проекта приобрести практические навыки создания и отладки программ; использовать композицию при разработке классов.

**Описательная часть отчета.** После заголовка и цели располагаются разделы описательной части. Шрифт: Times New Roman, размер шрифта: 12, интервал: одиночный, выравнивание: по ширине, отступ: 1.25 см.

Для каждого раздела указывается номер и название. Обязательными являются следующие разделы:

**1. Индивидуальное задание**

Привести текст индивидуального задания.

**2. Разработка программы**

Дать описание программной реализации выбранного метода решения задачи, указать особенности функционирования разработанной программы. Для упрощения описательной части, рекомендуется включать в текст программы комментарии в стиле javadoc и использовать сгенерированную документацию.

**2.1. Используемые средства ООП**

Дать краткую характеристику используемым при разработке программы принципам объектно-ориентированного подхода с соответствующими ссылками на текст программы.

**2.2. Иерархия и структура классов**

Привести схему (рисунок), отражающую внутреннюю структуру и взаимосвязи (отношения) разработанных классов. Дать описание приведенной схемы, указать типы связей и отношений.

**2.3. Описание программы**

Привести описание структуры проекта. Описать с соответствующими ссылками на текст (п.4) работу программы. Дать краткую характеристику разработанным классам, их полям и методам. Привести назначение параметров разработанных методов и используемых переменных.

**3. Текст программы**

Привести исходный текст созданных файлов проекта с комментариями.

**4. Результаты тестирования**

Привести результаты тестирования, исходные данные и результаты работы программы (конкретные значения).

**Заключение.** Название последнего раздела не нумеруется.

Подвести итоги и сделать выводы о достижении целей лабораторной работы и корректности результатов тестирования разработанной программы.

**3  Пример проекта**

**3.1  Задание**

Разработать сериализируемый класс для хранения параметров и результатов вычислений. Используя композицию, разработать класс для вычисления функции *y=sin(x)*. Примечание: аргумент функции задавать в градусах.

Разработать класс для демонстрации в диалоговом режиме сохранения и восстановления состояния объекта, используя сериализацию. Показать особенности использования *transient* полей.

Разработать класс для тестирования корректности результатов вычислений.

Использовать комментарии для автоматической генерации документации средствами javadoc.

**3.2  Разработка программы**

Используя материал, изложенный в п.2.2 на стр.12 создадим проект pro. В пакете ex01 (меню File/New/Package, поле Name: ex01) реализуем классы, структура которых соответствует схеме п.4.2.2.

Согласно TDD (Test Driven Development) разработаем тест, который будет проверять основную функциональность нашего кода. Предварительно создадим новую папку test (меню File/New/Source Folder, поле Folder name: test)

Теперь создадим в ней JUnit Test Case (меню File/New…/JUnit Test Case, поле Source folder: pro/test, поле Package: ex01, поле Name: MainTest, поле Class under test: ex01.Main, кнопка Finish). Для тестов используем JUnit 4.x.

В JUnit 4 тестом является метод класса, который не принимает никаких параметров и не возвращает значение. Этот метод должен быть помечен аннотацией @Test и может выбрасывать исключения.

Для быстрого создания методов можно воспользоваться одним из Code Snippets. Достаточно в теле класса набрать public и нажать Ctrl+Space, выбрать public\_method и нажать Enter. Eclipse автоматически сгенерирует блок кода для объявления public метода.

Просмотреть набор существующих и добавить свои шаблоны можно на вкладке Window/Preferences…/Java/Editor/Templates.

В общем тесте проверяем функциональность на примере основного варианта использования кода. В нашем случае создадим методы:

testCalc(), где проверим корректность вычисления заданной функции методом ex01.Calc

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

Для запуска JUnit Test Case можно щелкнуть правой кнопкой на коде метода и в контекстном меню выбрать Run As/JUnit Test.

После этого появится представление JUnit, в котором отобразится результат выполнения тестов. Задача разработчика состоит в том, чтобы запустить тест с успешным результатом – зеленым индикатором.

После того, как все тесты будут проведены – приложение можно считать завершенным.

**3.2.1  Используемые средства ООП**

Средства ООП реализуют различные способы повторного использования кода. Создавая новые классы можно использовать уже существующие, приспосабливая к решению собственных задач и не затрагивая при этом их исходного кода.

Один из способов предполагает создание объектов существующих классов внутри нового класса. Это называют композицией, потому, что новый класс создается из объектов других классов. Таким образом, повторно используется функциональность ранее разработанного кода, но не сам код непосредственно.

Второй подход предполагает создание нового класса с типом существующего. То есть  берётся оболочка (интерфейс) существующего класса и добавляется к нему новый код, при этом ранее созданный класс остаётся неизменным, он "передаёт" свою функциональность потомку.

При разработке классов используем композицию и неявное наследование.

**3.2.2  Иерархия и структура классов**

Структура классов и схема их отношений:



**3.2.3  Описание программы**

Сериализация объектов Java позволяет любой объект, который реализует интерфейс Serializable представить в виде последовательности байт, которая может быть полностью восстановлена для регенерации оригинального объекта.

Полезное свойство сериализации объектов состоит в том, что при этом сохраняется не только образ объекта, а за ним также следуют все ссылки, содержащиеся в объекте. Эти объекты также сохраняются, а за ними следуют все ссылки из каждого объекта, и т.д.

Для управления сериализацией при определении полей можно использовать ключевое слово transient, таким образом исключив поля из общего процесса сериализации.

При написании исходного кода используем стиль комментариев документации javadoc.

Структура проекта:

|  |  |
| --- | --- |
| Папка srchttp://cde.kpi.kharkov.ua/courses/course_285/Lb1/r_0_files/image002.png | Папка testhttp://cde.kpi.kharkov.ua/courses/course_285/Lb1/r_0_files/image003.png |

Выполним генерацию документации (меню Project/Generate Javadoc...). Описательную часть, начиная со страницы



рекомендуется приводить в отчёте при описании программы.

После проверки работоспособности готовой программы, создадим исполняемый JAR файл ex01.jar, предварительно проверив имена приложений в меню настройки конфигураций запуска (меню Run/Run configurations... раздел Java Application).

**3.3  Текст программы**

**3.3.1  Calc.java**

**package** ex01;

**import** java.io.IOException;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

/\*\* Содержит реализацию методов для вычисления и отображения результатов.

 \* **@author** xone

 \* **@version** 1.0

 \*/

**public** **class** Calc {

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

**private** **static** **final** String *FNAME* = "Item2d.bin";

/\*\* Сохраняет результат вычислений. Объект класса {@linkplain Item2d} \*/

**private** Item2d result;

/\*\* Инициализирует {@linkplain Calc#result} \*/

**public** Calc() {

       result = **new** Item2d();

}

/\*\* Установить значение {@linkplain Calc#result}

 \* **@param** result - новое значение ссылки на объект {@linkplain Item2d}

 \*/

**public** **void** setResult(Item2d result) {

       **this**.result = result;

}

/\*\* Получить значение {@linkplain Calc#result}

 \* **@return** текущее значение ссылки на объект {@linkplain Item2d}

 \*/

**public** Item2d getResult() {

       **return** result;

}

/\*\* 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 объекте {@linkplain Calc#result}

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

 \*/

**public** **double** init(**double** x ) {

       result.setX(x);

       **return** result.setY(calc(x));

}

/\*\* Выводит результат вычислений. \*/

**public** **void** show() {

       System.*out*.println(result);

}

/\*\* Сохраняет {@linkplain Calc#result} в файле {@linkplain Calc#FNAME}

 \* **@throws** IOException

 \*/

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

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

       os.writeObject(result);

       os.flush();

       os.close();

}

/\*\* Восстанавливает {@linkplain Calc#result} из файла {@linkplain Calc#FNAME}

 \* **@throws** Exception

 \*/

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

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

       result = (Item2d)is.readObject();

       is.close();

}

}

**3.3.2  Item2d.java**

**package** ex01;

**import** java.io.Serializable;

/\*\* Хранит исходные данные и результат вычислений.

 \* **@author** xone

 \* **@version** 1.0

 \*/

**public** **class** Item2d **implements** Serializable {

/\*\* Аргумент вычисляемой функции. \*/

// transient

**private** **double** x;

/\*\* Результат вычисления функции. \*/

**private** **double** y;

/\*\* Автоматически сгенерированная константа \*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

/\*\* Инициализирует поля {@linkplain Item2d#x}, {@linkplain Item2d#y} \*/

**public** Item2d() {

       x = .0;

       y = .0;

}

/\*\* Устанавливает значения полей: аргумента

 \* и результата вычисления функции.

 \* **@param** x - значение для инициализации поля {@linkplain Item2d#x}

 \* **@param** y - значение для инициализации поля {@linkplain Item2d#y}

 \*/

**public** Item2d(**double** x, **double** y) {

       **this**.x = x;

       **this**.y = y;

}

/\*\* Установка значения поля {@linkplain Item2d#x}

 \* **@param** x - значение для {@linkplain Item2d#x}

 \* **@return** Значение {@linkplain Item2d#x}

 \*/

**public** **double** setX(**double** x) {

       **return** **this**.x = x;

}

/\*\* Получение значения поля {@linkplain Item2d#x}

 \* **@return** Значение {@linkplain Item2d#x}

 \*/

**public** **double** getX() {

       **return** x;

}

/\*\* Установка значения поля {@linkplain Item2d#y}

 \* **@param** y - значение для {@linkplain Item2d#y}

 \* **@return** Значение {@linkplain Item2d#y}

 \*/

**public** **double** setY(**double** y) {

       **return** **this**.y = y;

}

/\*\* Получение значения поля {@linkplain Item2d#y}

 \* **@return** значение {@linkplain Item2d#y}

 \*/

**public** **double** getY() {

       **return** y;

}

/\*\* Установка значений {@linkplain Item2d#x} и {@linkplain Item2d#y}

 \* **@param** x - значение для {@linkplain Item2d#x}

 \* **@param** y - значение для {@linkplain Item2d#y}

 \* **@return** this

 \*/

**public** Item2d setXY(**double** x, **double** y) {

       **this**.x = x;

       **this**.y = y;

       **return** **this**;

}

/\*\* Представляет результат вычислений в виде строки.<br>{@inheritDoc} \*/

@Override

**public** String toString() {

       **return** "x = " + x + ", y = " + y;

}

/\*\* Автоматически сгенерированный метод.<br>{@inheritDoc} \*/

@Override

**public** **boolean** equals(Object obj) {

       **if** (**this** == obj)

              **return** **true**;

       **if** (obj == **null**)

              **return** **false**;

       **if** (getClass() != obj.getClass())

              **return** **false**;

       Item2d other = (Item2d) obj;

       **if** (Double.*doubleToLongBits*(x) != Double.*doubleToLongBits*(other.x))

              **return** **false**;

       // изменено сравнение результата вычисления функции

       **if** (Math.*abs*(Math.*abs*(y) - Math.*abs*(other.y)) > .1e-10)

              **return** **false**;

       **return** **true**;

}

}

**3.3.3  Main.java**

**package** ex01;

**import** java.io.IOException;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

/\*\* Вычисление и отображение результатов.

 \* Содержит реализацию статического метода main().

 \* **@author** xone

 \* **@version** 1.0

 \* **@see** Main#main

 \*/

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

/\*\* Объект класса {@linkplain Calc}.<br>Решает задачу инд. задания. \*/

**private** Calc calc = **new** Calc();

/\*\* Отображает меню. \*/

**private** **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.");

                     calc.show();

                     **break**;

              **case** 'g':

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

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

                     calc.show();

                     **break**;

              **case** 's':

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

                     **try** {

                            calc.save();

                     } **catch** (IOException e) {

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

                     }

                     calc.show();

                     **break**;

              **case** 'r':

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

                     **try** {

                            calc.restore();

                     } **catch** (Exception e) {

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

                     }

                     calc.show();

                     **break**;

              **default**:

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

              }

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

}

/\*\* Выполняется при запуске программы.

 \* Вычисляется значение функции для различных аргументов.

 \* Результаты вычислений выводятся на экран.

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

 \*/

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

       Main main = **new** Main();

       main.menu();

}

}

**3.3.4  MainTest.java**

**package** ex01;

**import** org.junit.Test;

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

**import** junit.framework.Assert;

**import** java.io.IOException;

**import** ex01.Calc;

/\*\* Выполняет тестирование разработанных классов.

 \* **@author** xone

 \* **@version** 1.0

 \*/

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

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

@Test

**public** **void**   testCalc() {

       Calc calc = **new** Calc();

       calc.init(0.0);

       *assertEquals*(0.0, calc.getResult().getY(), .1e-10);

       calc.init(90.0);

       *assertEquals*(1.0, calc.getResult().getY(), .1e-10);

       calc.init(180.0);

       *assertEquals*(0.0, calc.getResult().getY(), .1e-10);

       calc.init(270.0);

       *assertEquals*(-1.0, calc.getResult().getY(), .1e-10);

       calc.init(360.0);

       *assertEquals*(0.0, calc.getResult().getY(), .1e-10);

}

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

@Test

**public** **void**   testRestore() {

       Calc calc = **new** Calc();

       **double** x, y;

       **for**(**int** ctr = 0; ctr < 1000; ctr++) {

              x = Math.*random*() \* 360.0;

              y = calc.init(x);

              **try** {

                     calc.save();

              }  **catch** (IOException e) {

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

              }

              calc.init(Math.*random*() \* 360);

              **try** {

                     calc.restore();

              } **catch** (Exception e) {

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

              }

              *assertEquals*(y, calc.getResult().getY(), .1e-10);

              *assertEquals*(x, calc.getResult().getX(), .1e-10);

       }

}

}

**3.4  Результаты тестирования**

Выполним ex01.MainTest как JUnit Test



Выполним запуск программы из командной строки:

java -jar ex01.jar

В результате выполнения получим:



**4  Заключение**

В процессе ознакомления со средой разработки выполнили развертывание и настройку IDE Eclipse. На примере простейшего проекта получили представление о средствах, облегчающих создание программ. Приобрели практические навыки использования основных возможностей редактирования, автоматического исправления и отладки кода.

Разработали программу решения задачи индивидуального задания. При разработке классов использовали композицию. Результаты тестирования подтверждают корректность реализованных алгоритмов.