Практические занятия в компьютерном класса

Занятие №4 «Основы объектно ориентированного программирования. Создание своих классов»

Разработка программ, состоящих из нескольких классов. Применение интегрированной среды программирования **Eclipse** для разработки своих классов.

Задание №1

Разработать программу, которая ведет учет успеваемости студентов в рамках факультета.

Решение:

Любая программа состоит из множества классов, причем хотя бы один из них содержит метод main(), с которого начинает выполняться программа. В нашей программе каждый класс будет содержать метод main(), с помощью которого будет проведена проверка функциональности разработанного класса.

Для решения задания создадим три класса:

- класс Student, описывающий успеваемость студента;
- класс Group, моделирующий учебную группу, состоящую из студентов;
- класс Department, моделирующий факультет, состоящий из учебных групп.

Для решения задания в начале создадим класс Student, описывающий успеваемость студента. Класс должен содержать информацию о имени, фамилии студента и о количестве набранных им баллов. Кроме того, он должен позволять добавлять баллы к уже набранным баллам конкретного студента, и сообщать оценку студента по европейской системе оценивания (букву).

Как обычно, загружаем интегрированную среду программирования **Eclipse**, закрываем свои старые проекты и создаем новый *Java* проект с именем Theme4. В данном проекте создаем пакет classWork4, а в нем будем размещать свои классы.

Стандартным образом создадим класс Student. При создании класса отключим автоматическое создание функции main(). Будет автоматически создан и добавлен в пакет файл Student.java, содержащий класс Student:

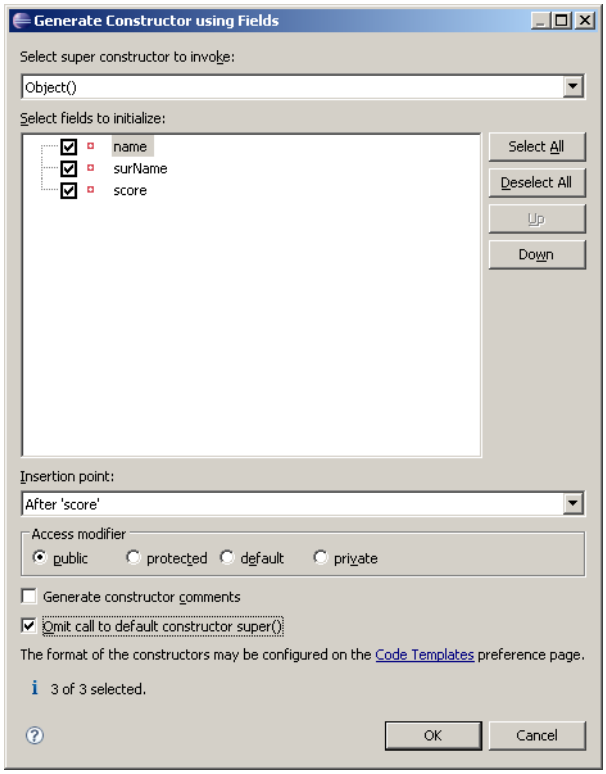
```
package classWork4;

public class Student {

}
```

Для реализации нужной функциональности добавим поля, предназначенные для хранения имени и фамилии студента, а также количества набранных им баллов:

```
public class Student {
   private String name;      // имя
   private String surName; // фамилия
   private int score;         // отчество
}
```

Затем к классу следует добавить конструкторы, которые нужны для корректного создания объектов данного класса. В дальнейшем потребуется создавать объекты класса Student по полной информации о студенте (имя, фамилия, количество набранных баллов),

только по имени и фамилии, а также вообще без данных (создать конструктор умолчанию). Текст конструкторов можно полностью написать вручную, а можно воспользоваться услугами интегрированной среды разработки, которая автоматически создаст значительную часть кода. Для создания конструкторов нужно вызвать команду меню *Source | Generate Constructor using Fields …*. На экран будет выведено диалоговое окно *Generate Constructor using Fields*, с помощью которого можно указать, какой конструктор будет создавать система.



Для того, чтобы создать конструктор с тремя параметрами, нужно в диалоговом окне в списке *Select fields to initialize* нужно выбрать поля, которые будет инициализировать конструктор. В нашем случае нужно выбрать все три поля. Затем, с помощью раскрывающегося списка *Insertion point* следует указать место в исходном коде класса, куда будет помещен созданный конструктор. Дальше, с помощью переключателей *Access modifier* нужно выбрать требуемый спецификатор доступа (в нашем случае public) и установить флажок *Omit call to default constructor super()* (смысл данного действия мы подробно обсудим на лекции). После того, как указаны все настройки нажимаем кнопку *OK*. Требуемый конструктор будет размещен в указанном месте исходного кода класса.

```
public Student(String name, String surName, int score) {
   this.name = name;
   this.surName = surName;
   this.score = score;
}
```

Аналогично создаем конструктор с двумя параметрами (имя, фамилия студента) и конструктор по умолчанию (вообще без параметров). Среда программирования сгенерирует такой код:

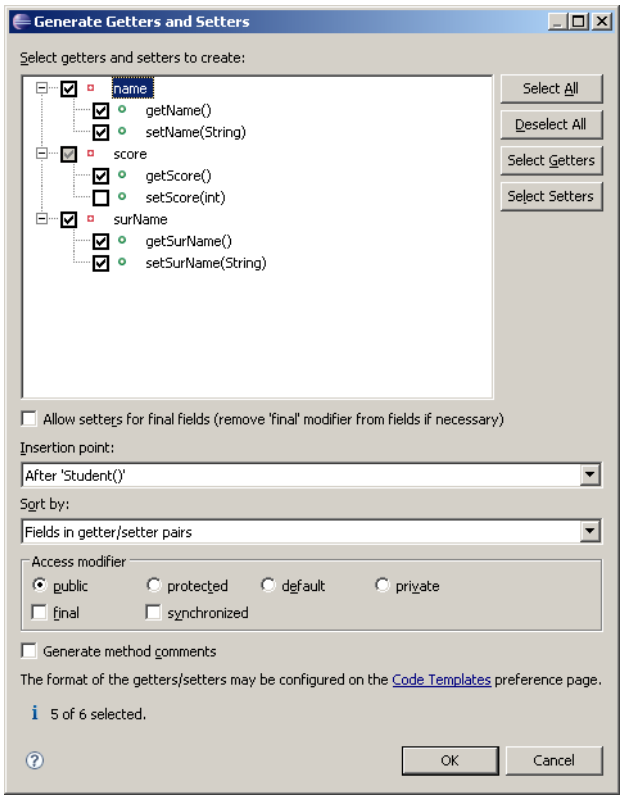
```
public Student(String name, String surName) {
   this.name = name;
   this.surName = surName;
}
```

```
public Student() {
}
```

Подправим код, созданный средой. Конструктор с двумя параметрами и конструктор по умолчанию примут такой вид:

```
public Student(String name, String surName) {
   this(name, surName, 0);
}

public Student() {
   this("Noname", "Noname", 0);
}
```

После создания конструкторов, создадим требуемые методы доступа. Как и конструкторы, их можно полностью создать самостоятельно, а можно прибегнуть к помощи среды разработки. Для этого нужно выбрать команду меню *Source | Generate Getters and Setters* .... На экран будет выведено одноименное окно, в котором нужно указать, какие методы доступа должны быть в исходном тексте класса.



Для дальнейшей работы нам понадобится возможность как узнавать имя, фамилию конкретного студента и количество набранных им баллов, так и изменять имя и фамилию студента. При этом для изменения количества баллов, набранных студентом мы чуть позже напишем специальную функцию, которая будет добавлять указанное количество баллов к уже набранным. Таким образом, создадим открытые (public) методы доступа (*getters*) для полей name, surName и score, а также открытые модифицирующие (public) методы (*setters*) для полей name и surName. Разместим их в тексте класса сразу после конструкторов. Интегрированная среда разработки должна сгенерировать такой код:

```
public String getName() {
   return name;
}
```

```
public void setName(String name) {
   this.name = name;
}

public String getSurName() {
   return surName;
}

public void setSurName(String surName) {
   this.surName = surName;
}

public int getScore() {
   return score;
}
```

После автоматически созданных методов доступа вручную добавим методы, которые будут решать поставленную задачу. Во-первых создадим общедоступный метод, который будет добавлять баллы к уже набранным.

```
public void addToScore(int ball) {
   score += ball;
}
```

Во-вторых, нужно добавить метод, который по набранным студентом баллам возвращал оценку по европейской системе оценивания.

```
public String scoreLetter() {
   String result = null;
   switch(score / 10){
   case 0: case 1: case 2: case 3: case 4:
      result = "Fx";
      break;
   case 5:
      result = "E";
      break;
   case 6:
      result = "D";
      break;
   case 7:
      result = "C";
      break;
   case 8:
      result = "B";
      break;
   case 9: case 10:
      result = "A";
      break;
   default:
      System.out.println("Неверное значение оценки");
      System.exit(-1);
   }
   return result;
}
```

Для тестирования разработанного класса добавим в него метод main(). В этом методе проверим работу всех конструкторов и методов класса. (**В приведенном ниже коде проверена работа только одного конструктора и методов класса. Самостоятельно дописать недостающее**).

```
public static void main(String[] args) {
   Student std = new Student("Иванов", "Иван");
   System.out.println("Начало семестра");
   System.out.println("Студент " + std.getSurName() + " " +
         std.getName() + ": " + std.getScore() + " балл. ");
   System.out.println("Сдача ДоЗы");
   std.addToScore(10);
   System.out.println("Студент " + std.getSurName() + " " +
         std.getName() + ": " + std.getScore() + " балл. ");
   System.out.println("Контрольная работа");
   std.addToScore(45);
   System.out.println("Студент " + std.getSurName() + " " +
         std.getName() + ": " + std.getScore() + " балл. " );
   System.out.println("Зачетная работа");
   std.addToScore(40);
   System.out.println("Студент " + std.getSurName() + " " +
         std.getName() + ": " + std.getScore() + " балл. ");
   System.out.println("Конец семестра. ЗАЧЕТ!");
   System.out.println("Студент " + std.getSurName() + " " +
         std.getName() + " набрал " + std.getScore() + " балл. " +
         "Оценка " + std.scoreLetter());
}
```

Затем создадим класс Group, описывающий успеваемость студенческой группы. Класс должен содержать название группы и массив студентов, которые учатся в этой учебной группе. В этом классе должны быть средства, позволяющие изменять параметры студентов, которые учатся в этой группе, а также вычислять средний балл и выводить на экран список студентов, которые набрали как больше, так и меньше заданного количества баллов.

Стандартным образом начнем создавать класс Group. При создании класса включим автоматическое создание функции main(). Будет автоматически создан и добавлен в пакет файл Group.java, содержащий шаблон класса Group:

```
package classWork4;

public class Group {
   /**
    * @param args
    */
   public static void main(String[] args) {
      // TODO Auto-generated method stub
   }

}
```

Затем к классу добавим поля, которые будут хранить нужную информацию.

```
private String name;        // Название группы
private Student[] team;   // Массив студентов
```

На следующей шаге автоматически добавим конструкторы, которые позволят корректно создавать студенческие группы:

- по названию и массиву студентов;
- без информации о студентах и названии группы;
- только по названию группы;
- по количеству студентов, которые в этой группе будут учиться.

Затем немного подправим автоматически сгенерированный код.

```
public Group(String name, Student[] team) {
   this.name = name;
   this.team = team;
}

public Group(String name) {
   this(name, new Student[0]);
}

public Group() {
   this("Noname", new Student[0]);
}

public Group(int groupSize) {
   name = "Noname";
   team = new Student[groupSize];
   for (int i = 0; i < team.length; i++)
      team[i] = new Student();
}
```

При создании группы по названию или вообще без информации о группе размер группы не известен. Поэтому массив студентов становится массивом нулевого размера. Когда создается учебная группа по количеству студентов, при ее создании нужно автоматически создать массив студентов нужного размера.

После того, как нужные конструкторы созданы, создадим методы доступа, с помощью которых можно узнавать и устанавливать название группы, а также задавать массив студентов, которые в ней обучаются.

```
public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public void setTeam(Student[] team) {
   this.team = team;
}
```

После создания методов доступа напишем методы, с помощью которых можно узнать количество студентов, которые обучаются в учебной группе и вывести состав учебной группы на экран:

```
public int numStudents() {
   return team.length;
}

public void showGroup(boolean withScore) {
```

```
    System.out.println("Группа " + name);
    int num = 0;
    for (Student std : team)
        if (withScore)
            System.out.println((++num) + ". " + std.getSurName() +
                    " " + std.getName() + " " + std.getScore() +
                    " баллов оценка " + std.scoreLetter());
        else
            System.out.println((++num) + ". " + std.getSurName() +
                    " " + std.getName());
}
```

Метод, который выводит состав учебной группы на экран имеет один параметр логического типа, с помощью которого можно вывести либо просто имена и фамилии студентов, обучающихся в указанной учебной группе, либо еще указать их балл и оценку.

Далее реализуем методы добавления студентов в группу и удаления студентов из группы. Для добавления студента в группу реализуем перегруженный метод: метод без параметров, когда добавляется студент без какой-либо информации о нем и метод с одним параметром, когда добавляется уже существующий студент. Когда в группу добавляется студент, то нужно увеличить размер уже существующего массива. Удобнее всего это сделать с помощью метода copyOf() класса Arrays. Для того, чтобы можно было воспользоваться этим методом нужно импортировать средства пакета util (import java.util.*;).

```
public void addStudent() {
    team = Arrays.copyOf(team, team.length+1);
    team[team.length-1] = new Student();
}

public void addStudent(Student std) {
    team = Arrays.copyOf(team, team.length+1);
    team[team.length-1] = std;
}

public void deleteStudent(int num) {
    for (int i = num-1; i < team.length-1; i++)
        team[i] = team[i+1];
    team = Arrays.copyOf(team, team.length-1);
}
```

Для удаления студентов из списка группы воспользуемся тем же методом, но уже не для увеличения, а для уменьшения размера массива. Сначала «передвинем» остаток массива (элементы, расположенные в массиве после удаляемого) на одну позицию влево (к началу массива), а затем с помощью метода copyOf() класса Arrays «удалим» последний элемент массива (точнее создадим опию исходного массива без последнего элемента).

Для того, чтобы можно было воспользоваться возможностями, предоставляемыми пользователю классом Student (изменение параметров конкретного студенты), создадим метод, который возвращает ссылку на конкретного студента, входящего в состав группы.

```
public Student getStudent(int i) {
    return team[i-1];
}
```

Внимание, в данной реализации методов deleteStudent и getStudent параметр метода — это не *индекс*, а НОМЕР студента в списке, отображаемом на экране функцией showGroup.

Осталось создать методы, которые вычисляют средний балл студентов в указанной группе и выводят на экран списки студентов, набравших больше и меньше заданного количества баллов.

```
public double averageScore() {
   double allScore = 0;
   for (Student std : team)
      allScore += std.getScore();
   return allScore / team.length;
}

public void printStudentsLess(int score) {
   System.out.println("Группа " + name);
   System.out.println("Студенты, набравшие меньше " +
         score + " баллов");
   int num = 0;
   for (Student std : team)
      if (std.getScore() < score)
         System.out.println((++num) + ". " + std.getSurName() +
               " " + std.getName() + ": " + std.getScore() +
               " : " + std.scoreLetter());
}

public void printStudentsGreater(int score) {
   System.out.println("Группа " + name);
   System.out.println("Студенты, набравшие не менее " +
         score + " баллов");
   int num = 0;
   for (Student std : team)
      if (std.getScore() >= score)
         System.out.println((++num) + ". " + std.getSurName() +
               " " + std.getName() + ": " + std.getScore() +
               " : " + std.scoreLetter());
}
```

Осталось написать статический метод main(), с помощью которого протестируем готовый класс Group.

```
/**
 * @param args
 */
public static void main(String[] args) {
   // TODO Auto-generated method stub
   Student[] stds = {new Student("Иван", "Иванов"),
               new Student("Петр", "Петров"),
               new Student("Сидор", "Сидоров"),
               new Student("Вася", "Пупкин"),
               new Student()};
   Group gr1 = new Group("Тя-11", stds);
   gr1.showGroup(false);
   gr1.addStudent();
   gr1.addStudent(new Student("Джон", "До"));
   gr1.deleteStudent(5);
   gr1.showGroup(false);
   Student std = gr1.getStudent(5);
```

```
    std.setName("Васисуалий");
    std.setSurName("Лоханкин");
    std.addToScore(50);
    gr1.showGroup(true);
    System.out.printf("Средний балл: %6.3f%n", gr1.averageScore());
    gr1.printStudentsLess(30);
    gr1.printStudentsGreater(30);
}
```

В этом методе нужно проверить работу всех конструкторов и методов класса. (**В приведенном ниже коде проверена работа только одного конструктора и методов класса. Самостоятельно дописать недостающее**).

Теперь можно создавать класс Department, описывающий успеваемость факультета. Класс должен содержать название факультета и массив учебных групп, входящих в состав этого факультета. Функциональность этого класса должна быть аналогична, функциональности класса Group. В этом классе должны быть средства, позволяющие изменять параметры групп, которые входят в состав факультета, а также вычислять средний балл и выводить на экран список студентов по группам, которые набрали как больше, так и меньше заданного количества баллов.

Данный класс разрабатывается аналогично классу Group. При создании класса включим автоматическое создание функции main(). Будет автоматически создан и добавлен в пакет файл Department.java, содержащий шаблон класса Department.

Рекомендуется разработать и проверить работоспособность этого класса самостоятельно, взяв за образец класс Group.

Для разрешения возможных проблем ниже приведен один из вариантов возможного написания этого класса.

Смотреть только в случае крайней необходимости. Совершенно секретно!!! ☺

```
package classWork4;

import java.util.Arrays;

public class Department {

    private String name;
    private Group[] department;

    public Department(String name, Group[] department) {
        this.name = name;
        this.department = department;
    }

    public Department(String name) {
        this(name, new Group[0]);
    }

    public Department() {
        this("Noname", new Group[0]);
    }

    public String getName() {
        return name;
```

```
}

public void setName(String name) {
   this.name = name;
}

public void setDepartment(Group[] department) {
   this.department = department;
}

public void showDepartment(boolean withScore) {
   System.out.println("\t\tФакультет " + name);
   int num = 0;
   for (Group gr : department){
      System.out.print("\t" + (++num) + ". ");
      gr.showGroup(withScore);
      System.out.println();
   }
}

public int numGroups() {
   return department.length;
}

public void addGroup() {
   department = Arrays.copyOf(department, department.length+1);
   department[department.length-1] = new Group();
}

public void addGroup(Group gr) {
   department = Arrays.copyOf(department, department.length+1);
   department[department.length-1] = gr;
}

public void deleteGroup(int num) {
   for (int i = num-1; i < department.length-1; i++)
      department[i] = department[i+1];
   department = Arrays.copyOf(department, department.length-1);
}

public Group getGroup(int i) {
   return department[i-1];
}

public double averageScore() {
   double allScore = 0;
   int allNum = department.length;
   for (Group gr : department)
      allScore += gr.averageScore();

   return allScore / allNum;
}

public void printStudentsLess(int score) {
```

```
    for (Group gr : department) {
        gr.printStudentsLess(score);
        System.out.println();
    }
}

public void printStudentsGreater(int score) {
    for (Group gr : department) {
        gr.printStudentsGreater(score);
        System.out.println();
    }
}

/**
 * @param args
 */
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Student[] stds = {new Student("Иван", "Иванов"),
            new Student("Петр", "Петров"),
            new Student("Сидор", "Сидоров"),
            new Student("Вася", "Пупкин"),
            new Student()};

    Group[] grs = {new Group("ТЯ-31", stds),
                new Group("ТЯ-32", new Student[]
                        {new Student("Филип", "Филипов"),
                         new Student("Сергей", "Сергеев"),
                         new Student("Геннадий", "Генадьев"),
                         new Student("Степан", "Степанов"),
                         new Student("Спиридон", "Спиридонов")})};

    Department faculty = new Department("ФТФ", grs);
    faculty.showDepartment(false);
    faculty.addGroup();
    faculty.addGroup();
    faculty.addGroup(new Group("ТЯ-32", new Student[]
          {new Student("Олег", "Олегов"),
           new Student("Михаил", "Мишин"),
           new Student("Виктор", "Викторов"),
           new Student("Борис", "Борисов"),
           new Student("Илья", "Ильин")}));
    faculty.showDepartment(false);
    faculty.deleteGroup(4);
    faculty.showDepartment(false);
    Group gr = faculty.getGroup(3);
    gr.setName("ТЯ-33");
    gr.addStudent(new Student("Виталий", "Витальев", 67));
    gr.addStudent(new Student("Григорий", "Григорьев", 77));
    gr.addStudent(new Student("Александр", "Александров", 88));
    faculty.showDepartment(true);
    System.out.printf("Средний балл: %6.3f%n", faculty.averageScore());
    faculty.printStudentsLess(30);
    faculty.printStudentsGreater(30);
```

```
    }
}
```

Дополнительное задание:

1. Создать класс Institute, описывающий успеваемость всего высшего учебного заведения, состоящего из факультетов Department, которые в свою очередь состоят из учебных групп Group, в которых учатся студенты Student.
2. В уже созданной и работающей программе учета успеваемости студентов изменить тип поля score с целого на вещественный (дать возможность работать с вещественными значениями баллов). Добиться того, чтобы сохранилась старая функциональность и появилась новая возможность работать с дробными баллами.
3. Организовать проверки допустимости индексов и передаваемых в метод значений.

Задание №2

Написать класс Fraction, представляющий простую дробь. Реализовать возможность создавать дроби, задавая как числитель, так и знаменатель, только числитель и вообще без аргументов. Реализовать возможность выполнять все арифметические операции с дробями (в результате должна получаться сокращенная дробь) и операции сравнения. Написать класс FractionDriver, с помощью которого проверить правильность реализации класса Fraction.

Fraction

- int numer; // числитель
- int denom; // знаменатель

+ Fraction(int numer, int denom); + Fraction(int numer); + Fraction(); - int GCD(int m, int n); - void Cancel(); + static boolean equal(Fraction frac1, Fraction frac2); + boolean equal(Fraction frac);

Возможная реализация класса Fraction представлена на *UML* диаграмме. На диаграмме представлены не все методы. Остальные создаются аналогично.