Лабораторная работа №3: Инкапсуляция и статичные методы, константы

Цели и задачи:

1. Научиться правильно применять ключевые слова **`public`**, **`private`** и **`protected`**
2. Узнать, что такое геттеры и сеттеры
3. Узнать, как проверять данные, вводимые в программу

Начало работы

Убедитесь в работоспособности вашего ПК. При обнаружении неполадок, сообщите преподавателю.

Для работы вам потребуется:

1. Среда разработки Eclipse (возможно так же использование другой IDE)
2. Java Development Kit 8 и выше.
3. Тетрадь и ручка для записи важных моментов

Теоретическая часть

Каждый объект не должен выставлять наружу все свои параметры для изменения просто так. Например, если у человека есть такие поля как возраст, ФИО и т.п., то не вызывает сомнений факт, что их нельзя менять прямо. Робот должен поменять свои координаты в результате передвижения. Нельзя обратиться к переменной age внутри объекта **`Human`** и сделать самое простое присваивание. Это будет как минимум нелогично. Лучше такого вообще не позволять. Т.е. мы таким образом должны создавать описание класса, чтобы нельзя было просто так получать доступ к его внутренним переменным. Это будет похоже на то, как если бы мы при производстве телевизора давали людям доступ ко всей схеме и каждый мог переключать проводки внутри него напрямую. Обычный человек таким телевизором вряд ли бы пользовался. Конечно нашлось бы несколько энтузиастов, которые обрадовались такому положению дел. Но это скорее всего экзотика. Более правильно в случае с человеком было сделать так, чтобы можно было получить значения возраста, ФИО, и т.п. А менять их можно было бы только в результате проверки. Например, нельзя присвоить человеку возраст более 120 лет, или пол, отличающийся от мужского или женского.

Т.е. закрытие внутренних переменных — хорошая идея. Для правильного поведения объекта. Нам самим будет проще — при условии, что класс более-менее разумно спроектирован и реализован. Но возникает резонный вопрос — а как тогда обращаться к внутренним переменным. В какой-то момент мы будем вынуждены это сделать.

Геттеры и сеттеры

Геттеры (от англ. get – получить) и сеттеры (от англ. set – задавать) являются важными и самыми нужными сторонниками инкапсуляции. Именно благодаря геттерам и сеттерам становится возможным притворить в жизнь логику проверки данных. Приведем небольшой пример:

```
private int age;
public int getAge() { // <<< геттер для поля age
    return age;
}

public void setAge(int age) { // <<< сеттер для поля age
    this.age = age;
}
```

ВАЖНО!
Для геттеров мы используем запись вида **get<ИмяПоля>**. Для сеттеров **set<Имя поля>**

Давайте немного расширим логику сеттера. Приведу пример всего класса **Human**, но пока без всех полей. Нас интересует пока что только одно поле.

```
public class Human {
    private int age; // объявим приватное поле

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if(age <= 120) this.age = age; // проверим, правильно ли указан возраст
    }
}
```

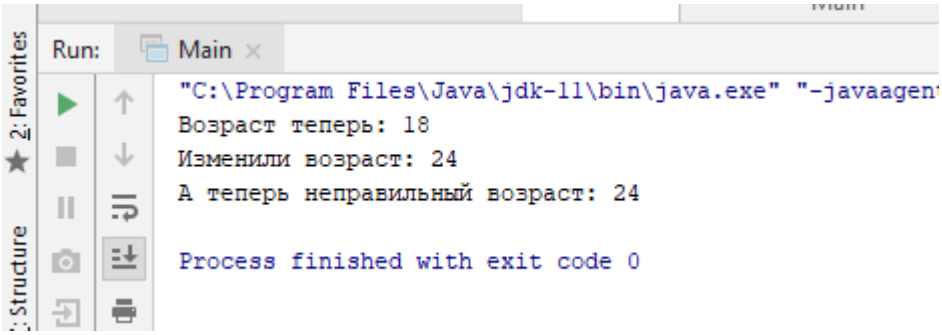
Попробуем им воспользоваться:

```
public class Main {

    public static void main(String[] args) {
        Human human = new Human();
        human.setAge(18);
        System.out.println("Возраст теперь: " + human.getAge());
        human.setAge(24);
        System.out.println("Изменили возраст: " + human.getAge());
        human.setAge(199);
        System.out.println("А теперь неправильный возраст: " + human.getAge());
    }

}
```

Попробуйте запустить этот код. Мы получим вывод:



Что случилось? Мы «скормили» нашему сеттеру данные, которые противоречат логике нашего класса. Возраст остался неизменным после того, как мы назначили «199» полю age. В данном случае мы поддерживаем логику класса Human, не давая задать неверные данные. Пусть лучше возраст будет равен нулю, зато данные в классе останутся «достоверными».

Давайте теперь введем еще одно поле, содержащее булево значение, определяющее, совершеннолетний ли человек. Назовем ее **inLegalAge** и создадим для него геттеры и сеттеры.

```
private Boolean inLegalAge;

public boolean isInLegalAge() { // обратите внимание на is
```

```
    return inLegalAge;
}

public void setInLegalAge(boolean inLegalAge) {
    this.inLegalAge = inLegalAge;
}
```

ВАЖНО!
Для полей, хранящих булево значение (**boolean**) для геттера используется префикс is

Теперь, давайте изменим метод **setAge**:

```
public void setAge(int age) {
    if(age <= 120) {
        this.age = age;
        this.setInLegalAge((this.getAge() >= 18) ? true : false); // можно ли это упростить?
    }
}
```

Таким образом, мы задали значение не только полю age, но и inLegalAge. Давайте выкинем исключение при попытке задать значение этого поля в true, если возраст человека < 18 лет:

```
public void setInLegalAge(boolean inLegalAge) throws IllegalArgumentException {
    if (inLegalAge && this.getAge() >= 18) {
        this.inLegalAge = true;
    }
    else {
        throw new IllegalArgumentException("Возраст < 18 лет!");
    }
}
```

Статические переменные

Статическая переменная в Java является переменной, которая принадлежит классу и инициализируется только один раз в начале выполнения.

- Это переменная, которая принадлежит классу, а не объекту (экземпляру)
- Статические переменные инициализируются только один раз, в начале выполнения. Эти переменные будут инициализированы сначала, прежде чем инициализировать любые переменные экземпляра
- Единая копия для совместного использования всеми экземплярами класса
- Статическая переменная может быть доступна непосредственно по имени класса и не нуждается в каком-либо объекте

Объявим статичную переменную. Для ее объявления воспользуемся ключевым словом **static**:

```
public static String title;
```

Для того, чтобы задать значение статической переменной используйте конструкцию **static {}**:

```
static {
    title = “Заголовок чего-либо, например”;
}
```

Статические методы

Статический метод в Java – это метод, который принадлежит классу, а не объекту. Статический метод может иметь доступ только к статическим данным.

- Это метод, который принадлежит классу, а не объекту (экземпляру)
- Статический метод может иметь доступ только к статическим данным. Он не может получить доступ к нестатистическим данным (переменные экземпляра)
- Статический метод может вызывать только другие статические методы и не может вызывать из него нестатический метод.
- Доступ к статическому методу можно получить непосредственно по имени класса и не нуждается в каком-либо объекте
- Статический метод не может ссылаться на **`this`** или **`super`**

ВАЖНО!
основной метод (**`main`**) является статическим, так как он должен быть доступен для запуска приложения, прежде чем произойдет какое-либо создание.

Приведем пример статического метода:

```
class Human {
// ...
    public static String getFio(Human human) {
        return human.getFirstName() + human.getLastName() + human.getMiddleName();
    }
// ...
}
```

В данном случае возьмите на заметку, как решается проблема доступа. Мы используем объект Human, и выводим поля оттуда, передавая в статический метод экземпляр.

Воспользуемся им:

```
System.out.println(Human.getFio(human));
```

Как видите, мы можем обращаться к такому методу по имени класса. Все довольно просто. На протяжении следующих лабораторных занятий мы закрепим полученные знания о статичных методах и переменных.

Константы

Константа — постоянное значение, известное до начала работы программы и заданное в коде один раз.

Почему это удобно:

- Если вы используете значение многократно, а в ходе разработки его потребуется изменить, достаточно будет одной правки.
- Вместо длинного значения можно ставить в коде короткое имя константы.
- Нет риска ошибиться при очередном вводе значения.

По логике это надо бы называть «постоянной» — в противоположность переменной, — но традиция уже сложилась.

Константа может хранить число, строку, символ, ссылку на объект и др. Значение константы еще называют литералом (от англ. *literal* — буквальный), потому что его записывают как есть — без предварительных вычислений.

При объявлении констант в Java используют ключевое слово **`final`** — оно показывает, что литерал не должен меняться. Именовать константы принято заглавными буквами со знаком подчеркивания вместо пробела: **`ВОТ_ТАК`**.

Давайте зададим в нашем классе константу **DISTANCE_TO_WALK**:

```
public static final double DISTANCE_TO_WALK = 25.4;
```

Таким образом мы сможем получить доступ к дистанции, которую должен пройти человек следующими только одним способом:

```
System.out.println(Human.DISTANCE_TO_WALK);
```

Здесь мы должны использовать имя класса для доступа. И никак иначе.

Неименованные константы

Неименованная константа — постоянное произвольное значение, которое вносят в код однократно:

```
if (max < 67) // 67 - это константа;
```

Если значение нужно использовать ещё где-то, лучше вынести его в именованную константу или финальную переменную.

Самостоятельная работа

Создайте класс для описания автомобиля (**Car**). Поместите в него информацию о марке, цвете, пробеге, номере, количестве бензина, количестве мест и т.п.

Создайте геттеры и сеттеры для доступа к членам класса. Сделайте проверку, не позволяющую задать автомобилю количество мест для пассажиров более восьми. Так же пробег автомобиля должен задаваться в метрах, а выводится в километрах.

Сделайте статичный метод, выводящий полное описание автомобиля, например,

```
«Toyota Corolla, цвет чёрный, номер а000бв, мест 4, осталось 27.3 л. бензина, пробег 1000км».
```

Оформите данную задачу как отдельный проект в среде разработки Eclipse или любой другой IDE.