**ЛАБОРАТОРНАЯ РАБОТА №3. КЛАССЫ И ОБЪЕКТЫ В JAVA. ПРИНЦИПЫ ООП – ИНКАПСУЛЯЦИЯ.**

**Цель работы**. Получить практические навыки разработки программ использованием объектно-ориентированного подхода на языке Java, создавать классы и объекты.

Что такое класс?

Языки структурного программирования, такие как Cи и Pascal, следуют совсем иной парадигме программирования, чем объектно-ориентированные языки. Парадигма структурного программирования ориентирована на данные, что означает, что сначала создаются структуры данных, а затем пишутся команды для работы с этими данными. В объектно-ориентированных языках, таких как Java, данные и команды программы скомбинированы в объекты.

Класс представляет собой автономный модуль со своими атрибутами и поведением. Вместо структуры данных с полями (атрибуты), которая отражается на всей логике программы, влияющей на ее поведение, в объектно-ориентированном языке данные и логика программы объединены. Эта комбинация может быть реализована на совершенно разных уровнях детализации, от самых мелких, до самых крупных.

Родительские и дочерние объекты

Родительский объект служит в качестве структурной основы для получения более сложных дочерних объектов. Дочерний класс повторяет родительский, но является более специализированным. Объектно-ориентированная парадигма позволяет многократно использовать общие атрибуты и поведение родительского класса, добавляя к ним новые атрибуты и поведение дочерних классов.

Связь между классами и координация

Классы общаются друг с другом, отправляя сообщения (на языке Java – вызовы методов). Кроме того, в объектно-ориентированных приложениях программа координирует взаимодействие между объектами для решения задачи в контексте данной предметной области.

Хорошо написанный класс

− имеет четкие границы;

− имеет конечный набор действий;

− "знает" только о своих данных и любых других объектах, которые нужны для его деятельности.

По сути, класс – это дискретный модуль, который обладает только необходимыми зависимостями от других классов для решения собственных задач.

Начнем с примера, основанного на общем сценарии разработки приложений: физического лица, представленного классом Person.

Атрибуты (Свойства, Поля)

Какие атрибуты может иметь физическое лицо? Вот самые распространенные из них:

− имя,

− возраст,

− рост,

− вес,

− цвет глаз,

− пол.

package com.makotogroup.intro;

public class Person {

private String \_name;

private int \_age;

private int \_height;

private int \_weight;

private String \_eyeColor;

private boolean \_gender;

}

Методы

Методы класса определяют его поведение. Иногда такое поведение – не более чем возврат (геттер, getter) текущего значения атрибута. В других случаях поведение может быть довольно сложным

Предлагаем посмотреть в Интернете, каким образом можно быстро реализовать геттеры и сеттеры в Java.

Существуют две категории методов: конструкторы и все прочие методы. Метод-конструктор используется только для создания экземпляра класса. Другие методы могут использоваться практически для любого поведения программы.

Методы-конструкторы

Конструкторы позволяют указать, как создавать экземпляр класса.

accessSpecifier ClassName([argumentList]) {

constructorStatement(s)

}

Например

public Person(String name, int age, int height, String eyeColor, boolean gender) {

this.\_name = name;

this.\_age = age;

this.\_height = height;

this.\_weight = weight;

this.\_eyeColor = eyeColor;

this.\_gender = gender;

}

Обратите внимание на использование ключевого слова this при присвоении значений переменным. В Java оно означает "this object" (этот объект) и служит для обращения к двум переменным с одинаковыми именами (как в данном случае, когда age – это и параметр конструктора, и переменная класса), а также помогает компилятору при неоднозначности ссылки.

У любого класса есть конструктор по умолчанию, например

public Person() { }

Но при указании нового конструктора, конструктор по умолчанию становится не доступным до тех пор, пока его явно не указать.

Конструктор – это метод особого рода с особой функцией. Точно так же методы многих других видов выполняют конкретные обязанности в Java-программах.

public String getName() {

return \_name;

}

public void setName(String value) {

name = value;

} // Другие комбинации геттеров/сеттеров...

Обратите внимание на комментарий о "комбинациях геттеров/сеттеров". Геттер – это метод для получения значения атрибута, а сеттер – для изменения этого значения.

Методы экземпляра и статические методы

Существуют два основных типа методов (кроме конструкторов): методы экземпляра (обычные методы) и методы класса (статические методы). Поведение метода экземпляра зависит от состояния конкретного экземпляра объекта. Статические методы иногда еще называют методами класса, так как их поведение не зависит от состояния какого-либо одного объекта. Поведение статического метода определяется на уровне класса.

Статические методы используются в основном для удобства, их можно представить как способ создания глобальных методов с сохранением при этом самого кода сгруппированным с классом, которому они нужны.

Метод toString()

Этот метод служит для представления объекта в виде строки. Это требуется, например, если необходимо вывести объект на экран.

Самое главное знать, что метод toString() есть у всех объектов и все объекты используют этот метод при работе со строками. Этот метод является методом класса Object. В случаи, когда от объекта требуется результат типа String, например: System.out.println(new Object()), этот метод вызывается автоматически. Он возвращает представление объекта в виде строки и по-умолчанию состоит из двух составляющих разделенных собачкой. Эти составляющие: имя\_класса\_объекта и хэш\_кода. Пример: java.lang.Integer;@24d200d8.

Оператор new создает экземпляр (объект) указанного класса и возвращает ссылку на вновь созданный объект. Ниже приведен пример создания и присваивание переменной person экземпляра класса Person.

Person person = new Person("Иван", 20, 175, true);

Инкапсуляция.

Инкапсуляция в Java реализована с помощью использования модификаторов доступа.

Язык Java предоставляет несколько уровней защиты, которые позволяет настраивать область видимости данных и методов. В Java имеется четыре категории видимости элементов класса

* private– члены класса доступны только членам данного класса. Всё что объявлено private, доступно только конструкторам и методам внутри класса и нигде больше. Они выполняют служебную или вспомогательную роль в пределах класса и их функциональность не предназначена для внешнего пользования. Закрытие (private) полей обеспечивает инкапсуляцию;
* по умолчанию (package-private) – члены класса доступны классам, которые находятся в этом же пакете;
* protected– члены класса доступны классам, находящимся в том же пакете, и подклассам – в других пакетах;
* public– члены класса доступны для всех классов в этом и других пакетах.

Модификатор класса указывается перед остальной частью описания типа отдельного члена класса. Это означает, что именно с него должен начинаться оператор объявления класса.

public String errMessage;

private AccountBalance balance;

private boolean isError(byte status) {}

public class Account {}

Когда член класса обозначается модификатором доступа public, он становится доступным для любого другого кода в программе, включая и методы, определенные в других классах.

Когда член класса обозначается модификатором private, он может быть доступен только другим членам этого класса. Следовательно, методы из других классов не имеют доступа к закрытому члену класса.

При отсутствии модификатора доступа, члены класса доступны другим членам класса, который находится в этом же пакете.

Модификатор доступа protected связан с использованием механизма наследования и будет рассмотрен позже.

Модификатор доступа указывается перед остальной частью описания типа отдельного члена класса (то есть, именно с модификатора доступа начинается объявление члена класса).

Член класса (переменная, конструктор, методы), объявленный public, доступен из любого метода вне класса.

Всё что объявлено private, доступно только конструкторам и методам внутри класса и нигде больше. Они выполняют служебную или вспомогательную роль в пределах класса и их функциональность не предназначена для внешнего пользования. Закрытие (private) полей обеспечивает инкапсуляцию.

**Сокрытие полей класса**

В подавляющем большинстве случаев, поля класса объявляются как private (это не касается статических переменных и констант, там ситуация может быть другая). Должны быть веские основания объявить поле класса общедоступным. Манипулирование данными должно осуществляться только с помощью методов.

Для того чтобы дать возможность получить доступ к переменной или дать возможность изменить ее значение, объявляют специальные методы, которые называются "**геттерами**" и "**сеттерами**".

**Геттер** возвращает значение приватного поля, тогда как **сеттер** меняет значение приватного поля (новое значение передается в качестве аргумента метода).

Хотя сигнатура и имена геттеров и сеттеров могут быть любыми, приучите себя соблюдать строгий шаблон для объявления геттеров и сеттеров.

Геттер должен иметь префикс get, после которого идет название поля с большой буквы. Геттер, как правило, не имеет входных аргументов.

Сеттер должен иметь префикс set, после которого идет название поля с большой буквы. Сеттер принимает на вход новое значение поля. Возвращаемый тип, как правило, void.

public class Account {

private double balance;

public double getBalance() {

return balance;

}

public void setBalance(double balance) {

this.balance = balance;

}

}

**Пример использования инкапсуляции**

Представим, что нам необходимо создать класс «Корзина» (Cart), который хранит в себе набор объектов класса «Товар» (Item).

Какие методы «Корзина» должна предоставлять для внешнего использования? Это могут быть, например, методы «Добавить товар», «Убрать последний добавленный товар», «Подсчет суммы цен товаров в корзине», «Повышение цен в корзине на N процентов» и «Снижение цен в корзине на N процентов».

| Название метода | Описание |
| --- | --- |
| public Cart(int capacity) | Конструктор с 1 параметром – максимальным количеством товаров в корзине. |
| public boolean addItem(Item item) | Добавление товара в корзину. Возвращает успешность операции. |
| public Item deleteLastAddedItem() | Удаление последнего добавленного товара в корзину. Возвращает удаленный товар. |
| public double calculateItemPrices() | Подсчет суммы цен всех товаров в корзине. |
| public void raiseItemPrices(double percent) | Поднять цены товаров в корзине на определенный процент (значение процента передается как аргумент метода). |
| public void cutItemPrices(double percent) | Снизить цены товаров в корзине на определенный процент (значение процента передается как аргумент метода). |

Как вы можете заметить, это публичные методы, а значит, их можно вызвать через оператор-точку имея ссылку ну объект.

Cart cart = new Cart();

cart.addItem(new Item("Клавиатура", 2000));

Перечень этих публичных методов и составляет **интерфейс** класса – то есть, с помощью этих методов объект класса будет взаимодействовать с внешним миром.

Эти методы имеют вполне четко определенные входные аргументы и могут возвращать значения четко определенных типов, и никак иначе. По аналогии с этим, поворот колес автомобиля осуществляется четко определенным образом – поворотом руля, и бензин надо заливать в четко определенное отверстие крышки бензобака, а не как-то еще.

То – как будет реализовано хранение товаров в корзине – это внутренняя логика класса и она не должна быть доступна внешнему миру, она должна быть скрыта от внешнего вмешательства. Другие классы, которые будут использовать объекты класса Cartне должны знать и не должны иметь доступ к тому – как там «внутри» реализовано хранение товаров, подсчет цен и изменение цены на определенный процент и так далее, они могут только лишь использовать предоставленные им публичные методы. Давайте реализуем «Корзину» с помощью структуры «стек», которая, в свою очередь, реализована обычным массивом.

public class Cart {

private Item[] stack; // массив для реализации стека

private int topIndex; // указатель на вершину стека

// При создании корзины мы должны

// указать максимальное количество элементов

// в корзине

public Cart(int capacity) {

stack = new Item[capacity];

topIndex = -1;

}

// Добавление нового товара в корзину

public boolean addItem(Item item) {

return push(item);

}

// Приватный метод, который реализует добавление в стек

private boolean push (Item item) {

// Добавляем товар в стек

return true; // или false если не стек переполнен

}

// Удаление последнего добавленного товара в корзину

public Item deleteLastAddedItem() {

return pop();

}

// Приватный метод, который реализует извлечение из стека

private Item pop() {

return new Item(); // Извлеченный из стека товар

}

}

**Перегрузка методов**

В Java разрешается в одном и том же классе определять два или более метода с одинаковым именем, если только объявления их параметров отличаются. В этом случае методы называются **перегружаемыми**, а сам процесс – **перегрузкой метода** (**method overloading**).

class MyClass {

public void foo() {

// ... код

}

public void foo(String s) {

// ... код

}

}

**Перегрузка методов позволяет поддерживать принцип «один интерфейс, несколько методов».**

Контрольные вопросы

1. Что такое сигнатура метода?
2. Зачем нужен конструктор?
3. Дайте определение классу и объекту?
4. Для каких целей используются пакеты в java?
5. Зачем необходимо ключевое слово new?
6. Описать сигнатуру пользовательского конструктора и конструктора по умолчанию.
7. Зачем нужны операторы импорта?

Задания по вариантам.

1. Создать программу на языке Java для определения класса в некоторой предметной области. Описать свойства, конструктор, методы геттеры/сеттеры, перекрыть метод toString() для вывода полной информации об объекте в отформатированном виде:
2. Создайте публичный класс Group\*Subject\* (Subject пишется в зависимости от вашего варианта). Реализовать в классе   
   - Поля: уникальный номер, массив объектов.   
   - Конструкторы: по умолчанию, принимающий на вход массив объектов  
   - Методы: get/set объект из массива, get/set массив, добавление/удаление из массива по атрибуту класса, сортировка массива (По вашему выбору)









