**Лабораторная работа №2 Объектная модель в Java**

**Цель работы**: изучение базовых конструкций языка Java.

**Задание:** Построить иерархию объектов. Показать использование и понимание принципов инкапсуляции, наследования, полиморфизма. В классах должны быть поля, конструкторы, методы. Обязательным является переопределение методов класса Object toString(), equals(), hashCode(). Использовать абстрактные классы и интерфейсы. Собрать коллекции объектов.

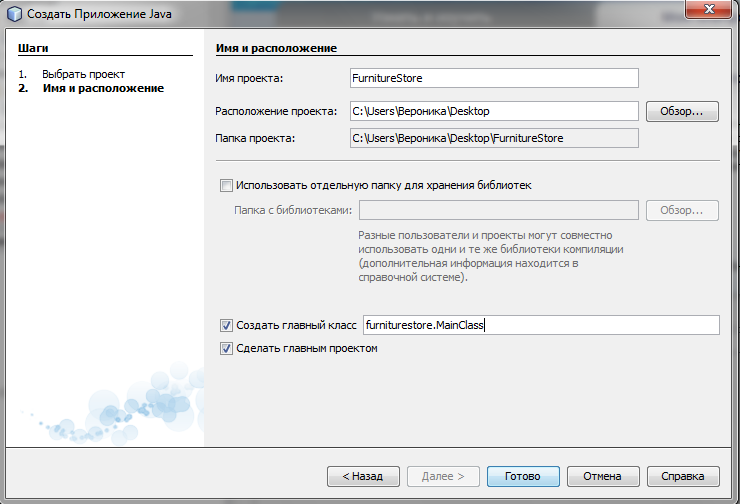
**1. Создание проекта в среде разработки NetBeans IDE.**

NetBeans IDE — свободная интегрированная среда разработки приложений (IDE) на языках программирования Java, JavaFX, Python, PHP, JavaScript, C++, Ада и ряде других.

Для того, что бы скачать данную среду разработки необходимо перейти на сайт http://netbeans.org/.

Т.к. данный проект является первой в данном курсе работой с Java, рассмотрим подробнее работу со средой NetBeans. Запускаем данную среду. Выбираем вкладку Файл – Создать проект. Из представленных категорий выбираем *Java*, а из окна проектов *приложение Java*.

Нажимаем клавишу далее. На следующей вкладке (рис. 2.1) вводим имя проекта, выбираем его месторасположение, создаём главный класс проекта, делаем наш проект главным.



*Рисунок 2.1 Создание проекта Java*

В главном классе проекта будет автоматически создан метод main() без которого невозможна работа программы.

Далее жмём клавишу Готово. В итоге получим проект с одним классом, который указали как главный.

Для создания пакетов(package) необходимо нажать правую кнопку мыши на строке *Пакеты исходных данных* или пакете, в который будет вложен другой пакет и выбрать *Создать – Пакет Java.*

Для создания классов и интерфейсов необходимо нажать правую кнопку мыши на пакете, в который будет вложен класс или интерфейс выбрать *Создать*, а потом выбрать объект, который мы хотим создать.

**2. Основные принципы ООП**

Java – объектно-ориентированный язык программирования, а значит при разработке программ должны учитываться принципы ООП.

**2.1 Классы**

Класс – это объект, описывающий содержание и поведение некой совокупности данных и действий над этими данными.

К примеру, если мы создаем объект стул, то данными могут быть: цена, модель, материал. На уровне класса мы ещё не знаем, о каком стуле идет речь, но точно знаем, что это не стол и не кресло.

Класс, обычно, создаётся в отдельном файле, при этом указывается имя пакета, в котором он расположен, тип объекта (class), спецификатор доступа (необязательно), поля, конструкторы и методы.

Поля – это атрибуты данного класса. Методы описывают действия над атрибутами данного класса. Конструкторы отвечают за создание объекта. Пример создания класса представлен на рисунке 2.1:

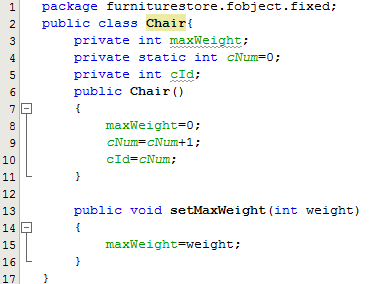


Рисунок 2.1 – Создание класса

В данном примере в строке 1 задано имя пакета, в котором находится класс. В строке 2 указаны: спецификатор доступа, тип объекта и имя класса. В строке 3-5 перечислены атрибуты класса. В строках 6-11 описан конструктор, он должен иметь такое же имя, как и класс. В строках 13-16 описан метод класса.

Следует помнить, что в классе может быть несколько конструкторов, которые должны отличаться списком параметров.

**2.2 Инкапсуляция**

Инкапсуляция – это свойство, которое позволяет пользователю не задумываться о реализации, посредством предоставления интерфейса для работы.

Т.е. можно разделить программу на две части: реализация и интерфейс.

Для реализации данного свойства в Java предусмотрено понятие спецификатор доступа. Он указывает на то, каким образом к данному объекту, полю, методу можно получить доступ.

Таблица 2.1 – Спецификаторы доступа

|  |  |
| --- | --- |
| Спецификатор | Описание |
| public | свободный доступ из любой части приложения |
| private | доступ возможен только внутри класса и невозможен внутри подкласса |
| protected | доступ возможен только внутри данного класса и класса потомка |
| по умолчанию | внутри пакета распознаётся, как public спецификатор, не доступен вне пакета |

**2.3 Наследование**

Наследование-это расширение свойств. Данное расширение происходит не за счёт изменения существующего класса или интерфейса, а по средствам создания нового класса или интерфейса.

Интерфейс описывает предполагаемое поведение класса не упоминая конкретных действий. Конкретные действия описываются уже в классах, которые наследуют данный интерфейс. Пример объявления интерфейса представлен на рисунке 2.2.

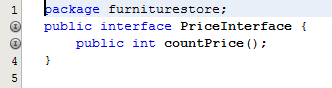


Рисунок 2.2 – Пример интерфейса.

Необходимо помнить, что Java не поддерживает множественное наследование классов, но при этом возможно множественное наследование интерфейсов. Т.е. у подкласса может быть только один суперкласс, а интерфейсов через запятую может быть указано несколько.

В случае наследования классов используется ключевое слово extends, а во втором – implements.

Абстрактный класс- класс, представляющий наиболее общие свойства объектов, и не предполагает создания экземпляров классов. Примером абстрагирования может служить понятие мебель. В данном понятии описываются свойства присущие всем предметам мебели (например, цена, материал). Но конкретное значение данных свойств мы можем узнать, только для конкретного объекта, такого как, например, стул. И для того, что бы определить данные свойства мы расширяем данный класс при помощи наследования, добавляя свойства, характерные уже для конкретных понятий.

Такой класс объявляет с ключевым словом abstract. Он может содержать абстрактные функции, которые присущи каждому объекту, но для каждого из них реализуются по-разному. Такие функции так же объявляются с ключевым словом abstract и их реализация представлена только в подклассах. Пример создания абстрактного класса представлен на рисунке 2.3:

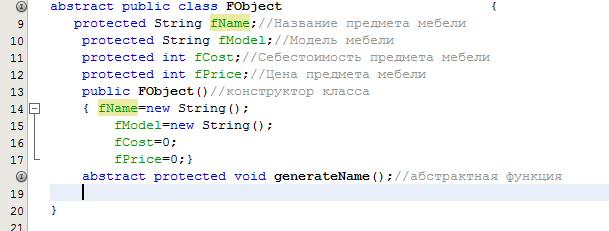


Рисунок 2.3 – реализация абстрактного класса.

В данном коде создаётся абстрактный класс FObject, в котором объявляются поля, характеризующие общие свойства всех предметов мебели (название, модель, цена, себестоимость). Присутствует конструктор, но только для того, что бы быть вызванным в конструкторе подкласса. Объявлена абстрактная функция, реализация которой будет проведена в подклассах.

При наследовании стоит быть внимательными со спецификаторами доступа (см. таблицу 2.1 пункт 2.2).

Для вызова конструктора суперкласса в подклассе используется выражение super(). Оно должно быть использовано в начале конструктора. Внутри данной super() должны быть перечислены параметры, соответствующие параметрам вызываемого нами конструктора суперкласса.

**2.4 Полиморфизм**

Полиморфизм – это способность функций с одинаковой спецификацией поддерживать различную реализацию. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Данный принцип включает в себя понятия перегрузки и переопределения.

При перегрузке в одном классе объявляется несколько функций, имеющих одинаковое название, но различающиеся параметрами, которые принимает функция. Примером перегрузки может служить перегрузка конструкторов класса. Пример представлен на рисунке 2.4:

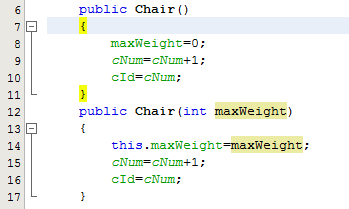


Рисунок 2.4 – Пример перегрузки

Как видно из примера данные конструкторы отличаются списком параметров. В зависимости от параметров при вызове будет вызван один из них.

Переопределение позволяет создать специфическую реализацию функции, определённой в суперклассе. Пример перегрузки показан на рисунках 2.4-2.5.

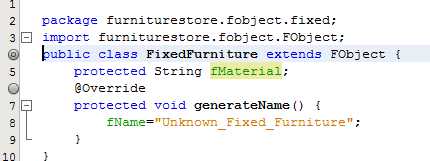


Рисунок 2.4 – Функция суперкласса

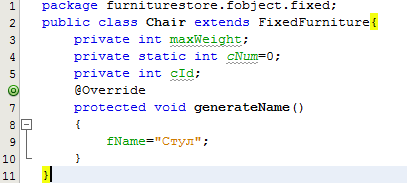


Рисунок 2.5 – переопределённая функция подкласса.

В классе FixedFurniture имеется функция generateName(). В подклассе Сhair класса FixedFurniture данная функция имеет уже другую реализацию. В случае, если бы данная функция не была переопределена был бы вызван метод суперкласса.

**3. Спецификаторы final, static**

Спецификатор final:

* поле является константой;
* метод нельзя заменить в подклассе;
* нельзя создать подкласс.

Спецификатор static:

* поле является общим для всех объектов данного класса: изменение поля в одно объекты приведёт к изменению значения этого поля в другом объекте;
* метод является общим для всех объектов данного класса: не содержит указателя this, поэтому и не может менять нестатические поля, пригоден только для изменения статических полей.

**4. Класс Object**

На вершине иерархии классов находится класс Object. Он является суперклассом для вех остальных классов. В данном классе определён набор методов, для корректной работы которых необходимо их переопределить в подклассах. Рассмотрим переопределение методов equals(Object ob), hashCode() и toString().

При переопределении equals(Object ob) должны выполняться следующие соглашения:

* рефлективность - объект равен самому себе;
* если x.equals(y) возвращает значение true, то и y.equals(x) всегда возвращает значение true;
* транзитивность – если метод equals() возвращает значение true при сравнении объектов x и y, а также y и z, то и при сравнении x и z будет возвращено значение true;
* непротиворечивость – при многократном вызове метода для двух не подвергшихся изменению за это время объектов возвращаемое значение всегда должно быть одинаковым;
* ненулевая ссылка при сравнении с литералом null всегда возвращает значение false.

Пример переопределения данной функции представлен на рисунке 4.1:

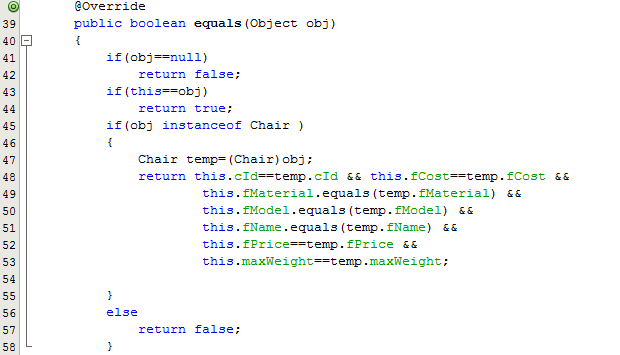


Рисунок 4.1. Переопределение метода equals().

Проверка в данном случае разбита на три части. Первое - проверка на пустоту объекта. Второе – проверка на то, что сравниваемые объекты являются на самом деле одним. Третья часть – проверка равенства атрибутов.

Метод hashCode() возвращает число, являющееся уникальным идентификатором объекта. Его следует переопределять всегда, когда переопределен метод equals(). Метод hashCode() возвращает хэш-код объекта, вычисление которого управляется следующими соглашениями:

* во время работы приложения при неизменности объекта неизменен и хэш-код;
* одинаковые объекты одного типа имеют одинаковый хэш-код;
* различные по содержанию объекты одного типа могут иметь различные хэш-коды.

На рисунке 4.2 представлен пример переопределения метода hashCode():

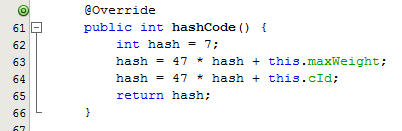


Рисунок 4.1 - Переопределение метода hashcode().

Стандартный метод toString() возвращает месторасположение класса, имя класса и его хэш-код. Если мы хотим, что бы данный метод возвращал другие значения, следует его переопределить. На рис. 4.3 представлено переопределение данного метода для того, что бы он возвращал содержимое класса.

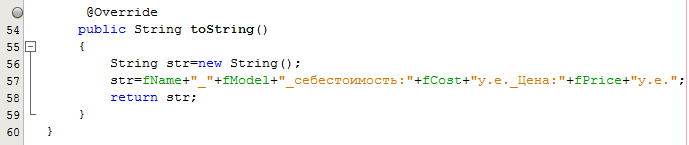


Рисунок 4.3 - Переопределение метода toString().

**5. Коллекции классов**

Коллекции – это хранилища объектов одного типа, предназначенные для упрощения работы с ними. В коллекциях поддерживаются 3 основные операции: добавление нового элемента в коллекцию, изменение элемента в коллекции, удаление элемента из коллекции.

Коллекции в языке Java объединены в библиотеке классов java.util и представляют собой контейнеры для хранения и манипулирования объектами.

В коллекциях хранится набор ссылок на объекты одного типа. Механизм работы с коллекциями:

* предварительное сообщение компилятору о типе ссылок, которые будут храниться в коллекции, при этом проверка осуществляется на этапе компиляции;
* отсутствие необходимости постоянно преобразовывать возвращаемые по ссылке объекты (тип Object) к требуемому типу.

Структура коллекций характеризует способ, с помощью которого программы Java обрабатывают группы объектов. Так как Object – суперкласс для всех классов, то в коллекции можно хранить объекты любого типа, кроме базовых.

Интерфейсы коллекций:

Map<K,V> – карта отображения вида “ключ-значение”;

Collection<E> – вершина иерархии остальных коллекций;

List<E> – специализирует коллекции для обработки списков;

Set<E> – специализирует коллекции для обработки множеств, содержащих уникальные элементы.

Все классы коллекций реализуют также интерфейсы Serializable, Cloneable (кроме WeakHashMap). Кроме того, классы, реализующие интер-фейсы List<E> и Set<E>, реализуют также интерфейс Iterable<E>.

Интерфейс Iterator<E> используется для построения объектов, которые обеспечивают доступ к элементам коллекции. К этому типу относится объект, возвращаемый методом iterator(). Такой объект позволяет просматривать содержимое коллекции последовательно, элемента за элементом. Позиции итератора располагаются в коллекции между элементами. В коллекции, состоящей из N элементов, существует N+1 позиций итератора.

В данном примере (мебельный магазин) коллекция необходима для сбора заказа, состоящего из предметов мебели. Т.к. класс FObject я бляется суперклассом для всех предметов мебели, то объекты именно данного типа будут указаны при объявлении и работы с коллекцией (рисунок 5.1).

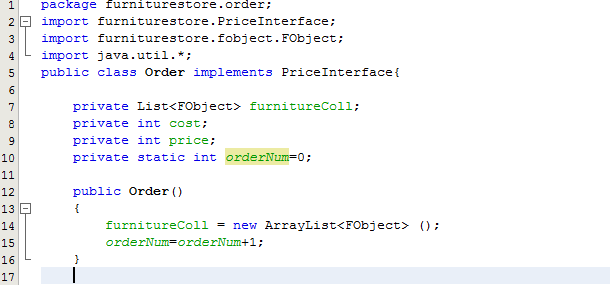


Рисунок 5.1 – Работа с коллекцией

В данном примере в классе заказов мы используем коллекцию предметов мебели. В строке 7 мы объявляем объект типа List, который является интерфейсом. В конструкторе класса (строка 15) мы создаём объект класса ArrayList. При этом не забываем указывать, тип объекта. В случае, если он не будет указан по умолчанию в коллекции смогут храниться объекты типа Object, а значит объекты всех классов, что не совсем корректно.

Пример работы с итераторами представлен на рисунке 5.2:

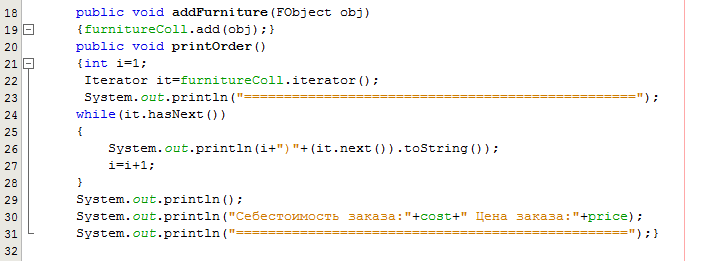


Рисунок 5.2 – Работа с итераторами

Объявляется объект типа Iterotor. Он инициализируется значением итератора, полученным из коллекции furnitureColl, создание которой описано ранее. Затем в цикле, пока у итератора есть следующее значение, выводится информация, объекта на который указывает итератор. Для получения этого объекта используется функция next().