**Занятие № 22-23.** Пакеты и интерфейсы.

Цель данной лабораторной работы - изучить основные концепции объектно-ориентированного программирования, ввести понятие интерфейса и научиться создавать собственные интерфейсы.

**Задание 1.** Запустить на выполнение код программ, приведенных в конспекте лекционного занятия «Занятие № 7. Пакеты и интерфейсы». Выполнять только те программы, для которых приведены примеры вывода на консоль. Обращать внимание на возникающие ошибки компиляции, которые нужно исправить. Добиться успешного выполнения программ.

**Задание 2.** Прочитать теоретические сведения. Интерфейсы предназначены для объявления типов, состоящих только из абстрактных методов и констант. Они позволяют задать для объявленных методов произвольную реализацию. Методы, входящие в интерфейс, могут быть реализованы в некотором классе так, как сочтет нужным разработчик класса. Следовательно, концептуально интерфейс представляет собой контракт между разработчиком и пользователем интерфейса. Пользователь интерфейса может разрабатывать свою программу, ориентируясь на спецификацию интерфейса. Между интерфейсом и абстрактным классом есть много общего, однако они существенно отличаются друг от друга. Интерфейс имеет следующие особенности:

• Интерфейс объявляется с помощью ключевого слова interface. По негласному соглашению, имя интерфейса начинается с буквы “ I ”.

• Нельзя создавать экземпляр интерфейса (так же, как и экземпляр абстрактного класса).

• В интерфейсе нельзя использовать директивы ограничения области видимости. Все методы интерфейса являются общедоступными по умолчанию, поэтому применение директивы public в объявлении интерфейса является недопустимым.

• В интерфейсе нельзя объявлять поля данных. В интерфейсе определяются только функциональные возможности, которые должны быть реализованы. На реализацию этих возможностей не накладывается никаких ограничений. Отсюда следует, что включение полей данных в интерфейс ограничивает возможности реализации его функциональности.

• Все методы, объявленные в интерфейсе, являются, по определению, абстрактными виртуальными функциями.

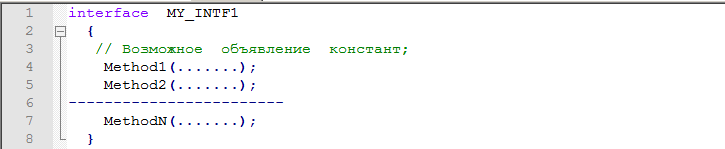
• После того, как интерфейс объявлен, создан и отлажен, его нельзя модифицировать. После того, как интерфейс становится достоянием пользователей, любые изменения в нем являются недопустимыми. Если возникает необходимость модернизации интерфейса, выпускается его новая версия.

Целью отделения интерфейса от реализации является скрытие всех деталей внутренней структуры объекта.

**Объявление интерфейса**

Не являясь классом, интерфейс должен объявляться именно в таком качестве.

В языке программирования Java интерфейс объявляется языковыми средствами с помощью ключевого слова interface. Объявление интерфейса похоже на объявление класса и имеет вид:



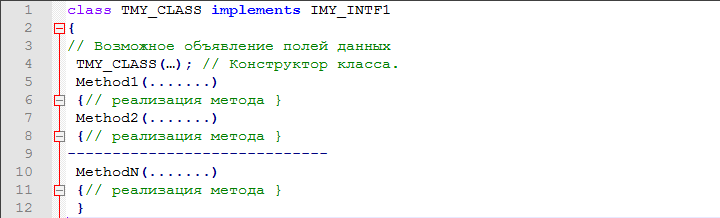
Интерфейс целесообразно объявлять в отдельном заголовочном файле, который затем подключается к разрабатываемой программе.

Следует отметить, что интерфейсы, подобно классам, могут объединяться в иерархии. При этом, несмотря на то, что в языке Java не поддерживается множественное наследование в иерархиях классов, иерархия интерфейсов допускает такое наследование. В языке Java интерфейсы, подобно классам, расширяются с помощью ключевого слова extends, например:

D:\Y-Disk\Скриншоты\2021-01-23_03-51-45.png

**Реализация интерфейса**

Интерфейс описывает контракт в абстрактной форме, однако он представляет интерес лишь после того, как он будет реализован в некотором классе. Класс, реализующий некоторый интерфейс, в своем объявлении должен содержать на него соответствующую ссылку. Так, в языке Java класс, реализующий интерфейс, объявляется следующим образом:



Следует отметить, что в языке Java не поддерживается множественное наследование в иерархиях классов, однако класс может реализовывать несколько интерфейсов. В этом случае они перечисляются через запятую, например:

D:\Y-Disk\Скриншоты\2021-01-23_03-55-41.png

Реализация объявленных методов находится в реализационном файле. Так же, как и в языке Java, класс может реализовывать несколько интерфейсов. В этом случае они перечисляются через запятую, например:

D:\Y-Disk\Скриншоты\2021-01-23_03-56-38.png

Следует отметить, что и в том, и в другом случаях реализация нескольких интерфейсов не является множественным наследованием.

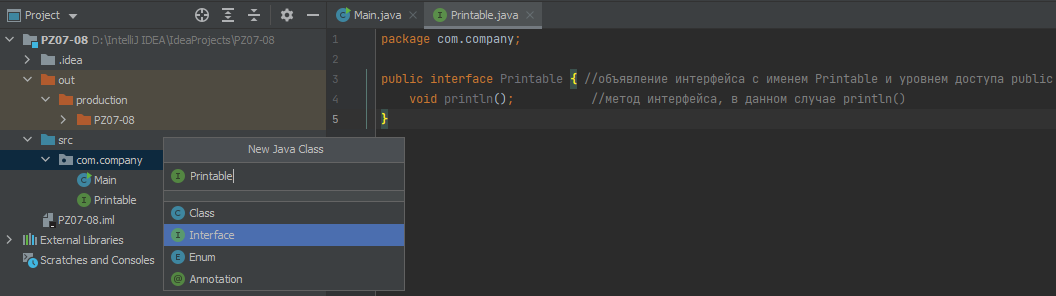
После того, как интерфейс реализован в некотором классе, можно создавать объекты этого класса, например:

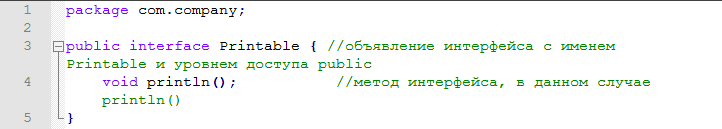
D:\Y-Disk\Скриншоты\2021-01-23_03-57-26.png

Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.

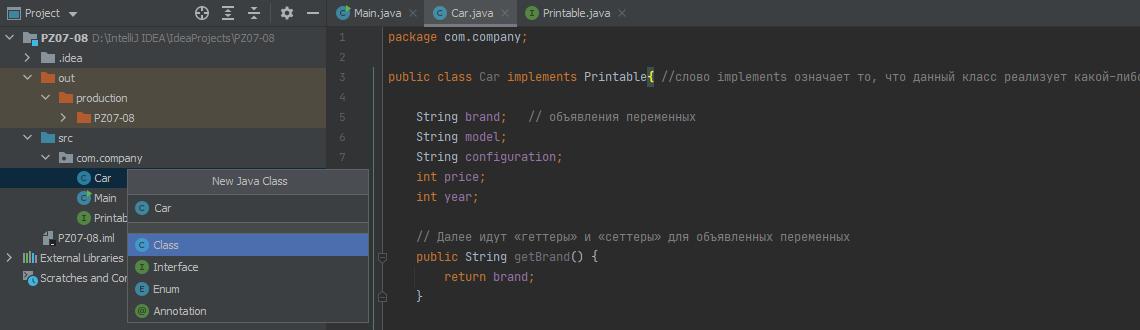
**Задание 3.** Реализовать интерфейс Printable, имеющий метод println(), используемый для вывода строк. Проверить работу.

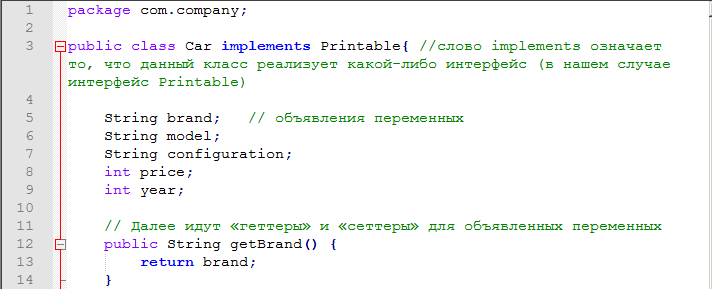
Для начала создадим интерфейс Printable в отдельном заголовочном файле.

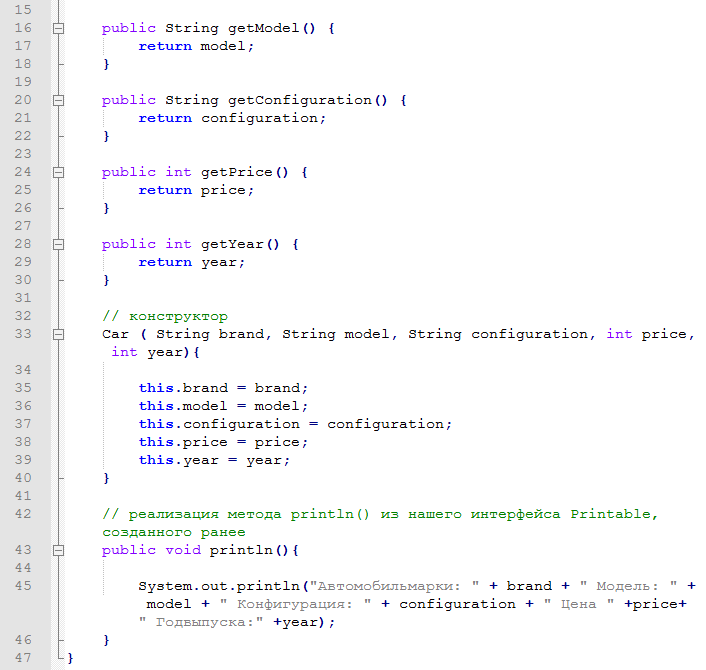




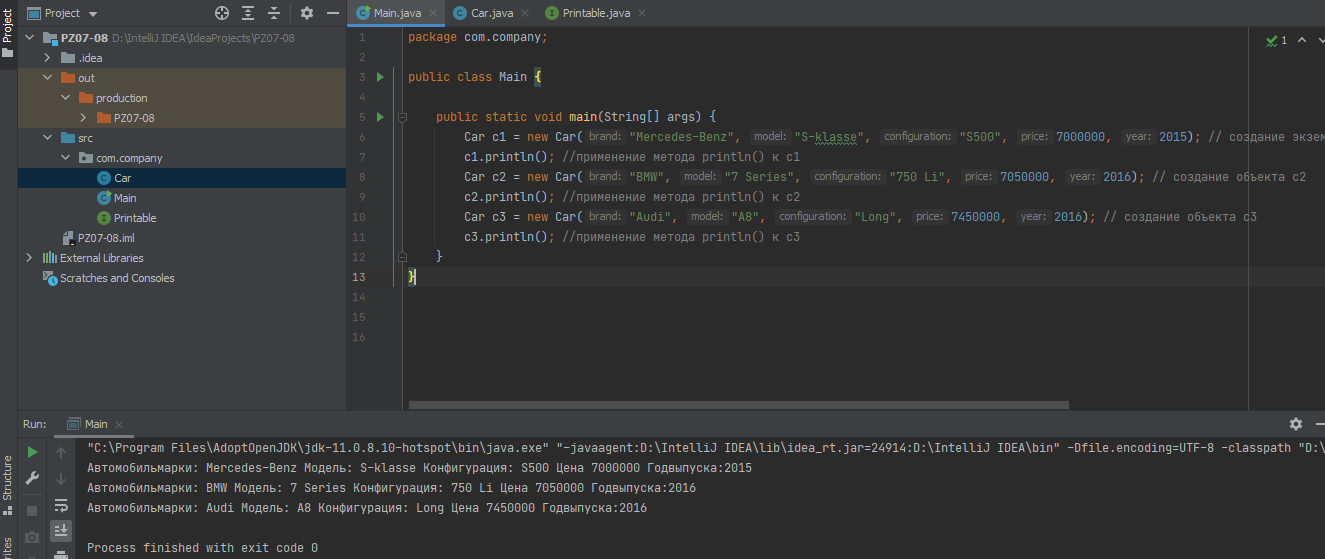
Теперь создадим класс Car, использующий интерфейс Printable и его методы.



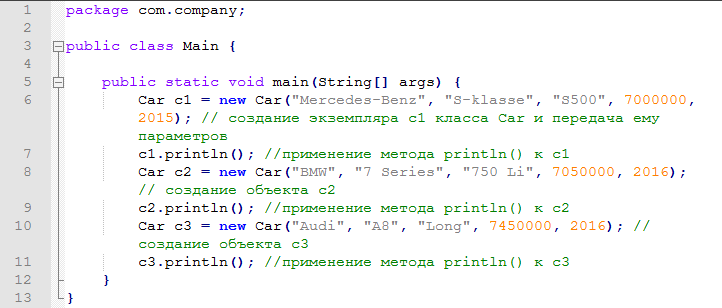




В классе Main проверим работы программы:



Код программы главного класса:



Одним из стандартных подходов при проектировании классов на языке Java является управление доступом к атрибутам класса через пару методов get и set. Метод get позволяет получить значение поля, set — установить новое значение. Общий принцип именования этих методов (называемых, также, геттером и сеттером).

Самостоятельно введите код описанной выше программы вывода данных об автомобилях и добейтесь ее успешного выполнения.

**Задание 4. Перечисления enum**

Кроме отдельных примитивных типов данных и классов в Java есть такой тип как enum или перечисление. Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора enum, после которого идет название перечисления. Затем идет список элементов перечисления через запятую:

enum Day{

MONDAY,

TUESDAY,

WEDNESDAY,

THURSDAY,

FRIDAY,

SATURDAY,

SUNDAY

}

Перечисление фактически представляет новый тип, поэтому мы можем определить переменную данного типа и использовать ее:

public class Program{

public static void main(String[] args) {

Day current = Day.MONDAY;

System.out.println(current); // MONDAY

}

}

enum Day{

MONDAY,

TUESDAY,

WEDNESDAY,

THURSDAY,

FRIDAY,

SATURDAY,

SUNDAY

}

Перечисления могут использоваться в классах для хранения данных:

public class Program{

public static void main(String[] args) {

Book b1 = new Book("War and Peace", "L. Tolstoy", Type.BELLETRE);

System.out.printf("Book '%s' has a type %s", b1.name, b1.bookType);

switch(b1.bookType){

case BELLETRE:

System.out.println("Belletre");

break;

case SCIENCE:

System.out.println("Science");

break;

case SCIENCE\_FICTION:

System.out.println("Science fiction");

break;

case PHANTASY:

System.out.println("Phantasy");

break;

}

}

}

class Book{

String name;

Type bookType;

String author;

Book(String name, String author, Type type){

bookType = type;

this.name = name;

this.author = author;

}

}

enum Type

{

SCIENCE,

BELLETRE,

PHANTASY,

SCIENCE\_FICTION

}

Само перечисление объявлено вне класса, оно содержит четыре жанра книг. Класс Book кроме обычных переменных содержит также переменную типа нашего перечисления. В конструкторе мы ее также можем присвоить, как и обычные поля класса.

С помощью конструкции switch..case можно проверить принадлежность значения bookType определенной константе перечисления.

**Методы перечислений**

Каждое перечисление имеет статический метод values(). Он возвращает массив всех констант перечисления:

public class Program{

public static void main(String[] args) {

Type[] types = Type.values();

for (Type s : types) { System.out.println(s); }

}

}

enum Type

{

SCIENCE,

BELLETRE,

PHANTASY,

SCIENCE\_FICTION

}

**Конструкторы, поля и методы перечисления**

Перечисления, как и обычные классы, могут определять конструкторы, поля и методы. Например:

public class Program{

public static void main(String[] args) {

System.out.println(Color.RED.getCode()); // #FF0000

System.out.println(Color.GREEN.getCode()); // #00FF00

}

}

enum Color{

RED("#FF0000"), BLUE("#0000FF"), GREEN("#00FF00");

private String code;

Color(String code){

this.code = code;

}

public String getCode(){ return code;}

}

Перечисление Color определяет приватное поле code для хранения кода цвета, а с помощью метода getCode оно возвращается. Через конструктор передается для него значение. Следует отметить, что конструктор по умолчанию приватный, то есть имеет модификатор private. Любой другой модификатор будет считаться ошибкой. Поэтому создать константы перечисления с помощью конструктора мы можем только внутри перечисления.

Также можно определять методы для отдельных констант:

public class Program{

public static void main(String[] args) {

Operation op = Operation.SUM;

System.out.println(op.action(10, 4)); // 14

op = Operation.MULTIPLY;

System.out.println(op.action(6, 4)); // 24

}

}

enum Operation{

SUM{

public int action(int x, int y){ return x + y;}

},

SUBTRACT{

public int action(int x, int y){ return x - y;}

},

MULTIPLY{

public int action(int x, int y){ return x \* y;}

};

public abstract int action(int x, int y);

}

Разобрать примеры программ, добиться понимания всех строк кода, выполнить их в среде разработки Java. Добиться успешного выполнения примеров.

Отчет о выполнении практического занятия должен содержать скриншоты выполненных вами программных модулей Задания 1, Задания 3, Задания 4.