2023

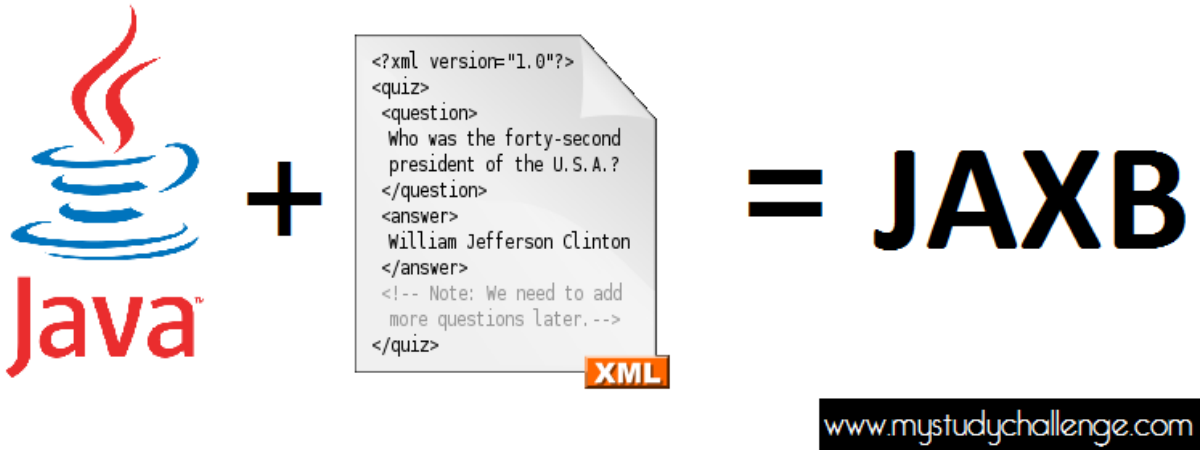
ПИСЛ

ЛК09 JAXB XML to DB

БУГИР. Каф. ЭИ.

2023

JAXB


Java
+
<?xml version="1.0"?>
<quiz>
<question>
Who was the forty-second
president of the U.S.A.?
</question>
<answer>
William Jefferson Clinton
</answer>
<!-- Note: We need to add
more questions later.-->
</quiz>
XML
= JAXB
www.mystudychallenge.com

JAXB (маршаллизация и демаршаллизация)

- **JAXB** — Java Architecture for XML Binding.

- **Маршаллизация** — программное преобразование данных из java-объектов в некоторое внешнее хранилище (XML, DB, TXT и т.д.).
- **Демаршаллизация** — обратный процесс (восстановление данных из XML, DB, TXT в java-объекты).
  - **Проблема**: организация взаимно однозначного соответствия информации в источнике, например, XML-документе, и экземпляре типа данных, принимающем эту информацию.
  - **Соединение** этих двух процессов должно корректно определять импорт-экспорт или круговорот информации без потерь и искажений на всех этапах.
  - **Информация**, взятая из XML-файла и транслированная в объект java, при обратном преобразовании должна быть возвращена **в идентичном виде**.

Некоторые аннотации JAXB

- **@XmlRootElement(name = «person»)** — указывает на корневой тег. Имя указывать необязательно.
- **@XmlType(propOrder = {«name»,»age»,»friends»})** — указываем последовательность атрибутов в **XML** схеме.
- **@XmlElement** — указывает на элемент который должен находиться в схеме. Можно не указывать, если элемент не должен менять свое имя.

Дополнительные аннотации

- **@XmlElementWrapper —** используется для коллекций, чтобы создать в схеме поверх них обертку. Таким образом, **XML** будет выглядеть более читабельно и понятно для человека.
- **@XmlTransient** — помечаются поля, которые не будут включены в маршалинг
- **@XmlSeeAlso** — используется, когда в классе существует объект другого класса
- **@XmlEnum** — для перечислений
- **@XmlEnumValue** — значение для полей в перечислении
- **@XmlAccessorType** — что именно будет сериализовано
- **@XmlElements** — контейнер для нескольких **@XmlElement**
- **@XmlMimeType** — mime-type для поля

Подготовка класса для преобразования в XML

```
//-------------------- описание отдельного класса Student ----------------------------
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)      // FIELD - все поля в XML
@XmlType(name = "Student", propOrder = {   // PROPERTY - только  с геттерами и сеттерами
        "name",                            // PUBLIC_MEMBER - только public
        "nickname",                        // NONE - ни одно
        "telephone",
        "address"
})
public class Student {
    @XmlAttribute(required = true) //опишем логин как аттрибут
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class) //тип адаптера
    @XmlID                         //и ключевой (уникальный)
    private String login;
    @XmlElement(required = true)
    private String name;
    @XmlElement(required = true)
    private String nickname;
    @XmlAttribute(required = false)
    private String faculty;
    @XmlElement(required = true)
    private int telephone;
    @XmlElement(required = true)
    private Address address = new Address();
    public Student() { } // необходим для маршаллизации/демаршалиизации XML

    public Student(String login, String name, String nickname, String faculty, int telephone, Addre
```

```
package by.it.akhmelev.JD02_09.manual;

//импорты
import javax.xml.bind.annotation.*;
import javax.xml.bind.annotation.adapters.CollapsedStringAdapter;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
```

Аннотация (главный класс + вложенный класс)

```java
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import java.util.ArrayList;

@XmlRootElement
public class Students {
    @XmlElement(name="student")
    private ArrayList<Student> list = new ArrayList<Student>();
    public Students() {
        super();
    }
    public void setList(ArrayList<Student> list) { this.list = list; }
    public boolean add(Student st) { return list.add(st); }
    @Override
    public String toString() { return "Students [list=" + list + "]"; }
}
```

```java
//-------------------- описание вложенного класса Address ----------------
@XmlRootElement
@XmlType(name = " address ", propOrder = {
        "city",
        "country",
        "street"
})
public static class Address {
    private String country;
    private String city;
    private String street;
    //конструкторы
    public Address() {// необходим для маршаллизации/демаршалиизации XML
```

После расстановок аннотаций нужны getters, setters, конструкторы, метод toString.

IDEA делает это автоматически

Code-Generate…

Маршаллизация

```java
public class JaxB_01_ToXML {
    public static void main(String[] args) {
        try {
            JAXBContext context = JAXBContext.newInstance(Students.class);
            Marshaller m = context.createMarshaller();
            Students st = new Students() { // анонимный класс
                {
                    // добавление первого студента
                    Student.Address addr = new Student.Address("BLR", "Minsk", "Lenina 7");
                    Student s = new Student("uvik", "Migov", "Uvik", "poit", 1234567, addr);
                    this.add(s);
                    // добавление второго студента
                    addr = new Student.Address("BLR", "Grodno", "Harna 23");
                    s = new Student("muvik", "Kurin", "Muvik","poit", 7654321, addr);
                    this.add(s);
                }

                ;
            };
            m.marshal(st, new FileOutputStream("src/by/i
            System.out.println("XML-файл создан");
            m.marshal(st, System.out); // копия на консо
        } catch (FileNotFoundException e) {
            System.out.println("XML-файл не может быть с
        } catch (JAXBException e) {
            System.out.println("JAXB-контекст ошибочен "
        }
    }
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students>
    <student login="uvik" faculty="poit">
        <name>Migov</name>
        <nickname>Uvik</nickname>
        <telephone>1234567</telephone>
        <address>
            <city>Minsk</city>
            <country>BLR</country>
            <street>Lenina 7</street>
        </address>
    </student>
    <student login="muvik" faculty="poit">
        <name>Kurin</name>
        <nickname>Muvik</nickname>
        <telephone>7654321</telephone>
        <address>
            <city>Grodno</city>
            <country>BLR</country>
            <street>Harna 23</street>
        </address>
    </student>
</students>
```

Демаршаллизация

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class JaxB_02_FromXML {
        public static void main(String[ ] args) {
            try {

                JAXBContext jc = JAXBContext.newInstance(Students.class);
                Unmarshaller u = jc.createUnmarshaller();
                FileReader reader = new FileReader("src/by/it/akhmelev/JD02_09/xml_01.xml");
                System.out.println("XML-файл прочитан:");
                Students students = (Students) u.unmarshal(reader);
                System.out.println(students);

            } catch (JAXBException e) {
                e.printStackTrace();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
```

```
D:\Java\jdk1.8.0_73\bin\java
XML-файл прочитан:
Students [list=[
Login: uvik
Name: Migov
Nickame: Uvik
Telephone: 1234567
Faculty: poit
Address:
```

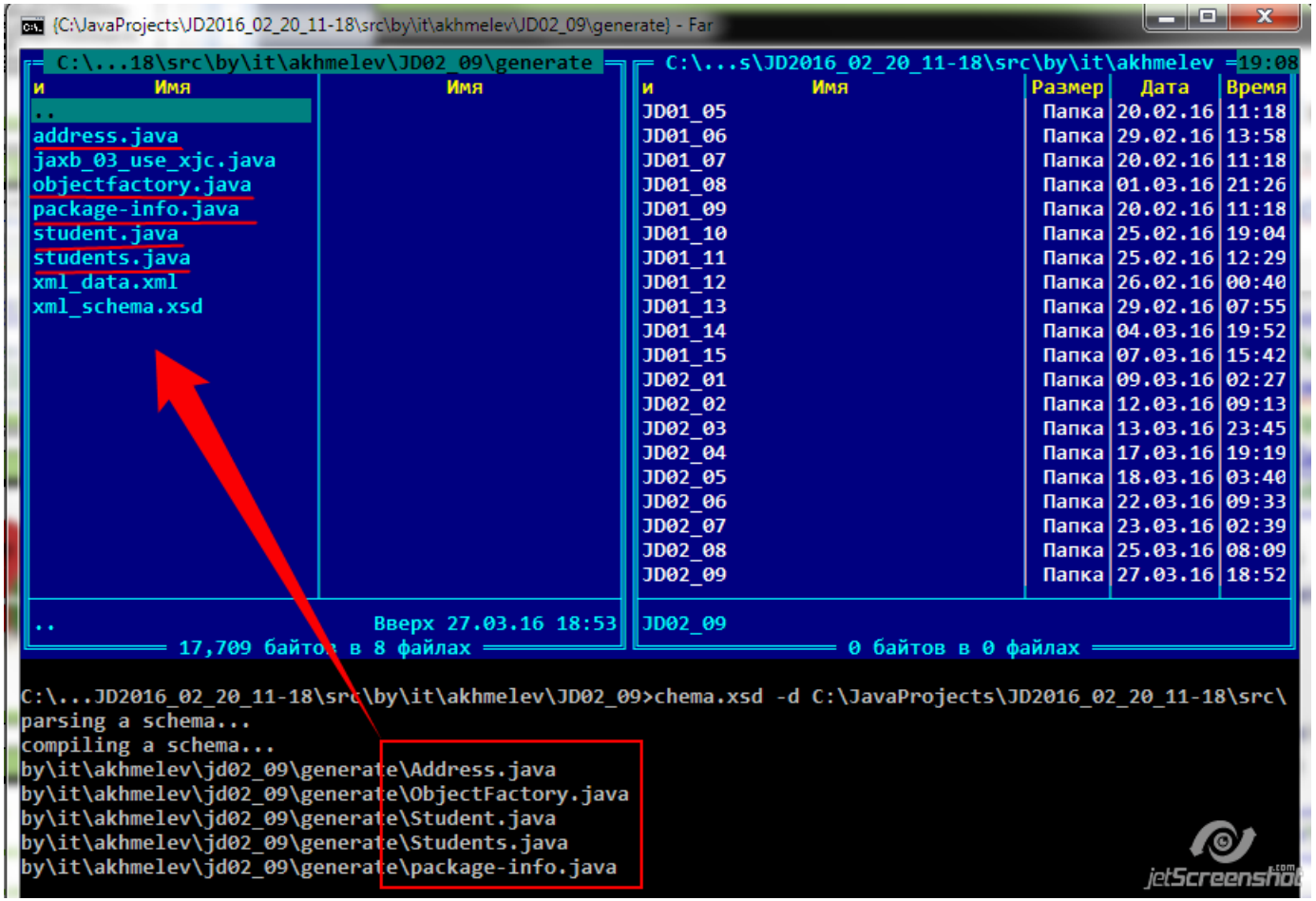
6: TODO 9: Changes Version Control

Java Bean & POJO

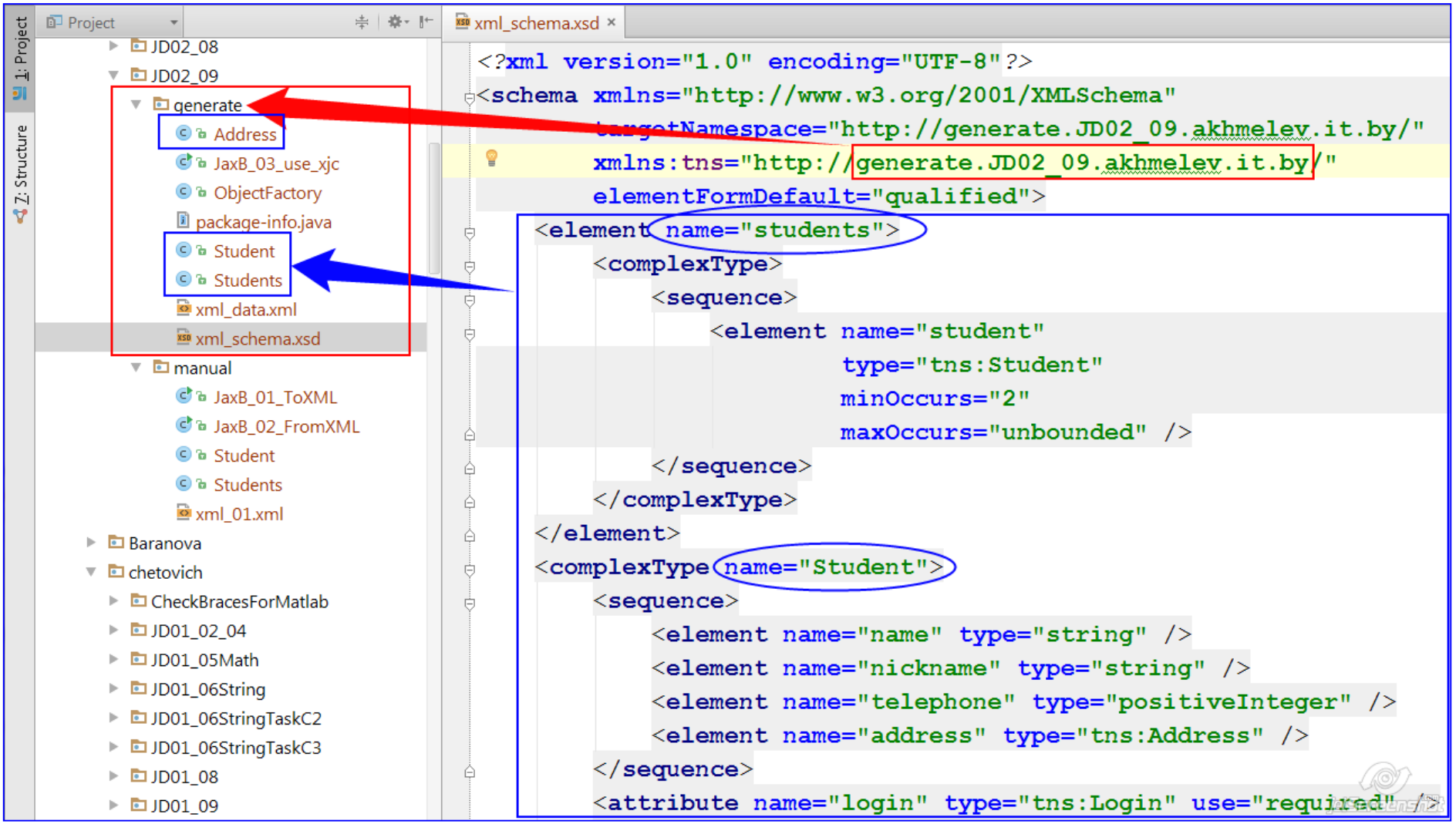
- Java **bean** – класс построенный по соглашениям об именах методов, конструкторе и поведении.
- Эти соглашения дают возможность использовать, замещать и соединять **beans**.
- Правила:
  - Классу нужен **public** конструктор без параметров, он позволяет создать объект без сложностей с параметрами.
  - Свойства класса (**private**) должны быть доступны через **get**, **set** и другие методы, которые должны подчиняться стандартному соглашению об именах. Это легко позволяет автоматически определять и обновлять содержание bean'ов.
  - Класс должен быть сериализуемым для того чтобы сохранять, хранить и восстанавливать состояние **bean** независимым от платформы и виртуальной машины способом.
  - Класс должен иметь переопределенные методы **equals**(), **hashCode**() и **toString**().
  - Так как требования в основном изложены в виде соглашения, а не интерфейса, некоторые разработчики рассматривают JavaBeans, как **Plain Old Java Object (POJO)**.

Генерация заготовок EJB через xcj.exe

- Кратчайший формат xjc.exe xml_schema.xsd но если не работает, можно указать пути:
- Путь_к_JDK\bin\xjc.exe имя_схемы.xsd -d Путь_куда_поместить_результат
- Пример: D:\Java\jdk1.8.0_73\bin\xjc.exe xml_schema.xsd -d C:\JavaProjects\JD2021\src\



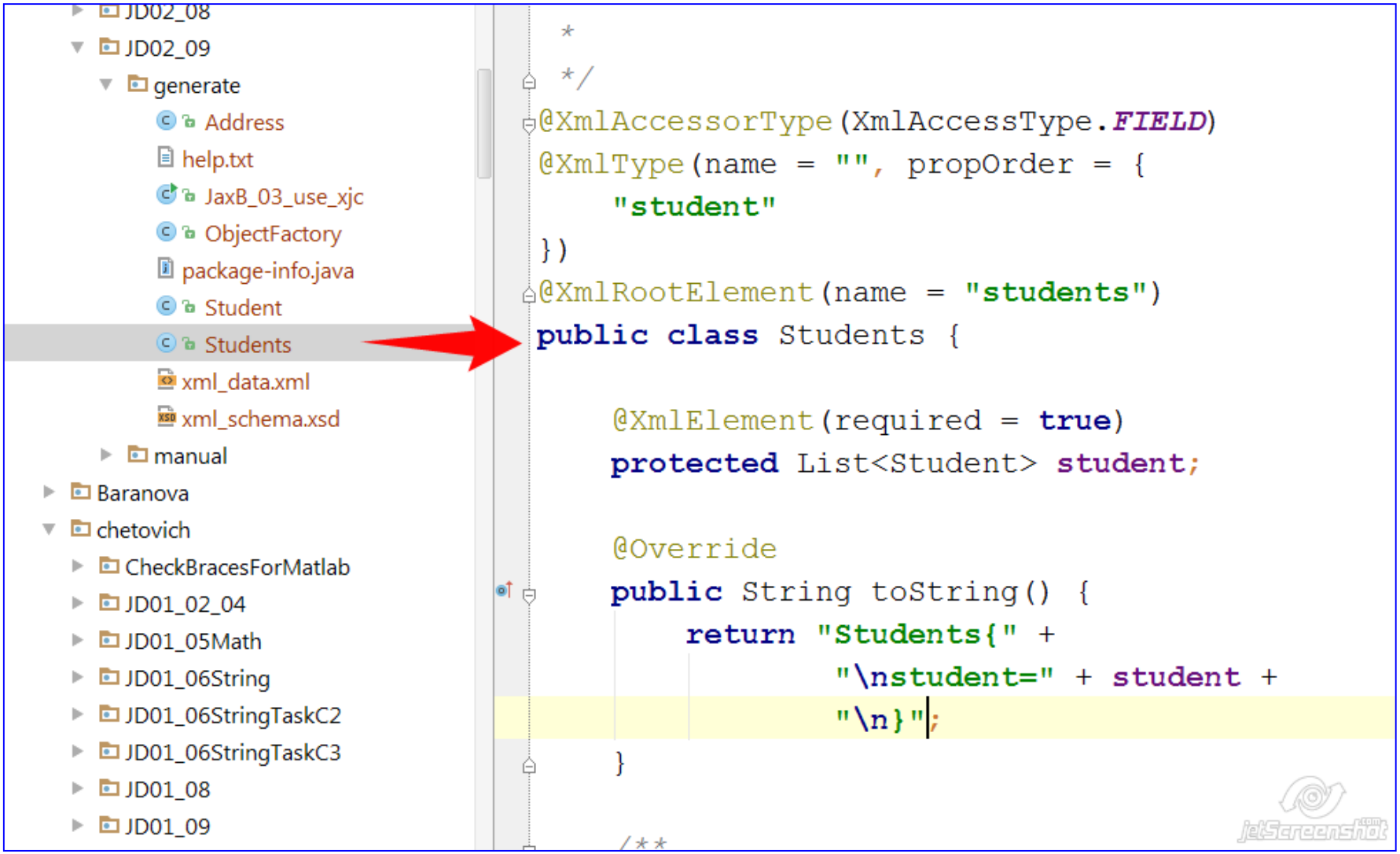
А ВОТ ТУТ ТЕПЕРЬ ОЧЕНЬ ВАЖЕН ПУТЬ В СХЕМЕ !



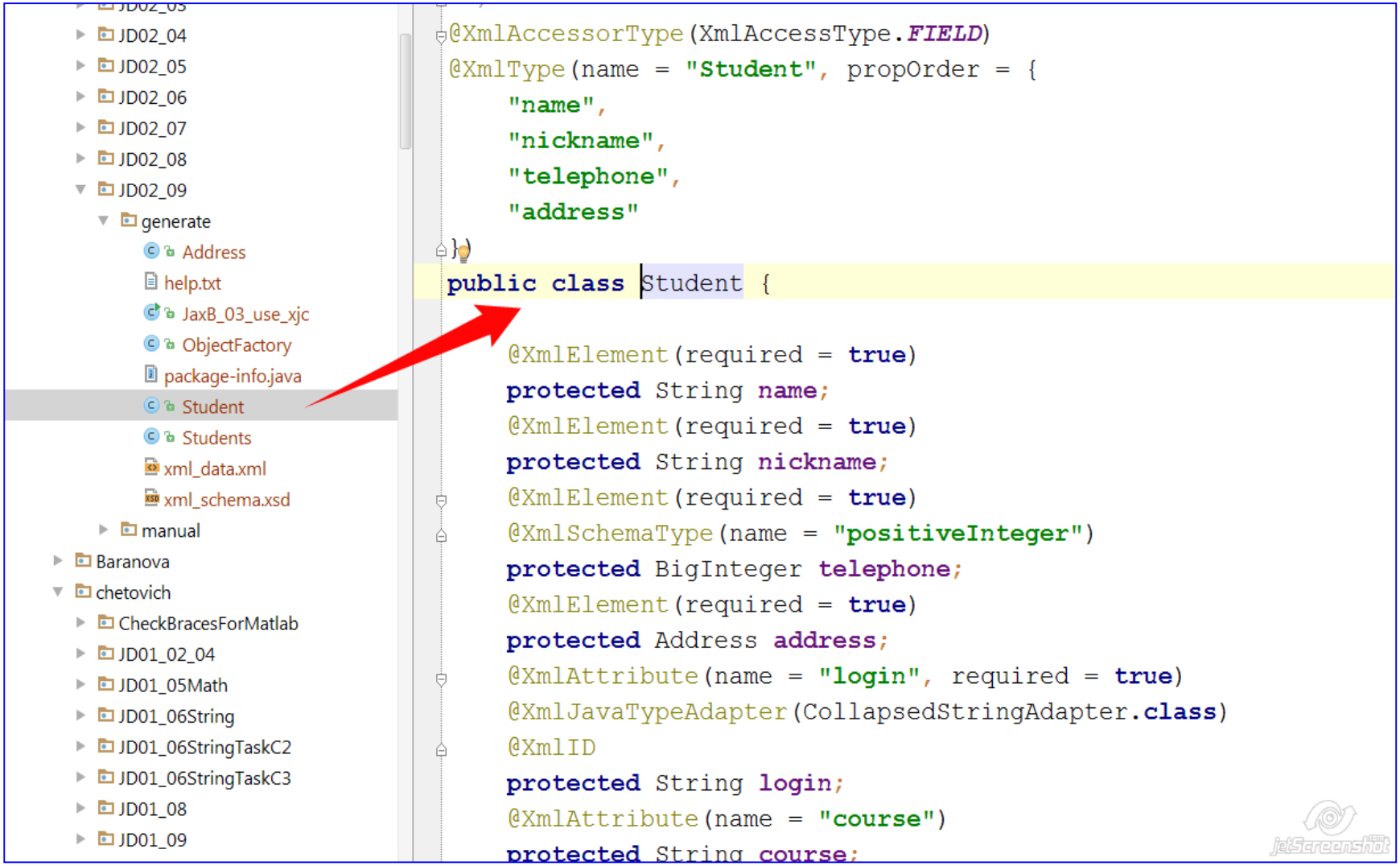
Опции генератора

```
Usage: xjc [-options ...] <schema file/URL/dir/jar> ... [-b <bindinfo>] ...
If dir is specified, all schema files in it will be compiled.
If jar is specified, /META-INF/sun-jaxb.episode binding file will be compiled.
Options:
  -nv                 :  do not perform strict validation of the input schema(s)
  -extension          :  allow vendor extensions - do not strictly follow the
                         Compatibility Rules and App E.2 from the JAXB Spec
  -b <file/dir>       :  specify external bindings files (each <file> must have its own -b)
                         If a directory is given, **/*.xjb is searched
  -d <dir>            :  generated files will go into this directory
  -p <pkg>            :  specifies the target package
  -httpproxy <proxy> :  set HTTP/HTTPS proxy. Format is [user[:password]@]proxyHost:proxyPort
  -httpproxyfile <f> :  Works like -httpproxy but takes the argument in a file to protect password
  -classpath <arg>    :  specify where to find user class files
  -catalog <file>     :  specify catalog files to resolve external entity references
                         support TR9401, XCatalog, and OASIS XML Catalog format.
  -readOnly           :  generated files will be in read-only mode
  -npa                :  suppress generation of package level annotations (**/package-info.java)
  -no-header          :  suppress generation of a file header with timestamp
  -target (2.0|2.1)   :  behave like XJC 2.0 or 2.1 and generate code that doesnt use any 2.2 features.
  -encoding <encoding> :  specify character encoding for generated source files
  -enableIntrospection :  enable correct generation of Boolean getters/setters to enable Bean Introspection a
  -contentForWildcard  :  generates content property for types with multiple xs:any derived elements
  -xmlschema          :  treat input as W3C XML Schema (default)
  -relaxng            :  treat input as RELAX NG (experimental,unsupported)
  -relaxng-compact    :  treat input as RELAX NG compact syntax (experimental,unsupported)
  -dtd                :  treat input as XML DTD (experimental,unsupported)
  -wsdl               :  treat input as WSDL and compile schemas inside it (experimental,unsupported)
```

Класс Students



Класс Student



Класс Address

Использование сгенерированного кода

```java
package by.it.akhmelev.JD02_09.generate;
//тут нужно указать путь к сгенерированным классам

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class JaxB_03_use_xjc {
        public static void main(String[ ] args) {
            try {

                JAXBContext jc = JAXBContext.newInstance(Students.class);
                Unmarshaller u = jc.createUnmarshaller();
                FileReader reader = new FileReader("src/by/it/akhmelev/JD02_09/generate/xml_data.xml");
                System.out.println("XML-файл прочитан:");
                Students students = (Students) u.unmarshal(reader);
                System.out.println(students);

            } catch (JAXBException e) {
                e.printStackTrace();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
        }
```

```
D:\Java\jdk1.8.0_73\bin\java ...
XML-файл прочитан:
Students{
student=[
Student{
 name='Ivanov Alexander',
 nickname='"M&M's"',
 telephone=4586954,
 address=Address{country='Belarus', city='Minsk', street='Le
 login='IvanovMitarAlex7',
 course='it.jd.02'
},
Student{
 name='Zimin Pavel',
```

2023

JSON

```
@Test
public void testFromJson() throws IOException {
    StringBuffer sb = readJsonFile("person.json");

    // fromJson
    Gson gson = new Gson();
    Person person = gson.fromJson(sb.toString(), Person.class);

    assertEquals("홍길동", person.getName());
    assertEquals("비빔밥", person.getFavoriteFood().get(2));
    assertEquals("abc@naver.com", person.getEtc().get("email"));
}
```

JSON (JavaScript Object Notation)

- **Wikipedia:**
  - **JSON** (*JavaScript Object Notation*) — текстовый формат обмена данными.
  - Как и многие другие текстовые форматы, JSON легко читается людьми.
  - Формат JSON был разработан Дугласом Крокфордом.
  - Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования.
  - Для многих языков существует готовый код для создания и обработки данных в формате JSON.

Формат данных JSON

- JSON это одна из двух структур:
  - Набор пар **ключ**: **значение** (объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив).
    - **Ключом** может быть только **строка**, значением — любая форма.
  - Упорядоченный набор значений (массив, вектор, список или последовательность).
- Это универсальные структуры данных:
  - как правило, любой современный язык программирования поддерживает их в той или иной форме.

- В качестве значений в JSON могут быть использованы:
  - **Объект** — это неупорядоченное множество пар **ключ**:**значение**, заключённое в фигурные скобки *«{ }»*. Ключ описывается **строкой**, между ним и значением стоит символ *«:»*.
    - Пары *ключ-значение* отделяются друг от друга запятыми.
  - **Массив** (одномерный) — это упорядоченное множество **значений**.
    - Массив заключается в квадратные скобки *«[ ]»*.
    - Значения разделяются запятыми.
  - **Число.**
  - **Литералы** *true*, *false* и *null*.
  - **Строка** — это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки.
    - Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты *«\»*, или записаны в кодировке UTF-8.

- *Строка* очень похожа на одноимённый тип данных в языках C и Java.
- *Число* тоже очень похоже на C- или Java-число, за исключением того, что используется только десятичный формат.
- Пробелы могут быть вставлены между любыми двумя синтаксическими элементами.

Сравнение JSON, XML, XML+attr

```json
{
    "firstName": "Иван",
    "lastName": "Иванов",
    "address": {
        "streetAddress": "Московское ш., 101, кв.101",
        "city": "Ленинград",
        "postalCode": 101101
    },
    "phoneNumbers": [
        "812 123-1234",
        "916 123-4567"
    ]
}
```

```xml
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101, кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

```xml
<person firstName="Иван" lastName="Иванов">
  <address streetAddress="Московское ш., 101, кв.101" city="Ленинград" postalCode="101101" />
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

Пример использования GSON

```
package by.it.akhmelev.JD02_10;
//для корректной работы добавьте библиотеку
//File->Project Structure->Libraries->maven->com.google.gson
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class GSON_Demo {

    public static void main(String[] args) {
        //маршаллизация и демаршаллизация в/из JSON. Подготовим  обработчики.
        Gson gson = new GsonBuilder().serializeNulls().setPrettyPrinting().create();
        /* Строка выше это кратко. Создадим еще один точно такой же обработчик json
        GsonBuilder builder = new GsonBuilder(); //Это строитель
        builder=builder.serializeNulls();        //в нем пишем что сериализуются null
        builder=builder.setPrettyPrinting();     //и вывод будет форматированный
        Gson gson=builder.create();              //теперь строитель строит того же обработчика
        */

        // Создаем объект для тестов
        String[] skills={"java","gson","json"};
        Dev dev=new Dev("Ivanov",skills);
        //маршаллизация
        String json = gson.toJson(dev);
        System.out.print(json);
        //демаршаллизация
        Dev dev2=gson.fromJson(json, Dev.class);
        System.out.print("\n\ndev2=" + dev2.toString());
    }
}
```

Пример (де) сериализации POJO-класса

```
class Dev {
    private int id = 0;
    private String name;
    private List skills = new ArrayList<String>();

    public Dev() {};

    public Dev(String name, String[] arraySkills) {
        this.id = ++id;
        this.name = name;
        setSkills(arraySkills);
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

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

    public List getSkills() { return skills; }

    @Override
    public String toString() {
        return "DevPOJO{" +
```

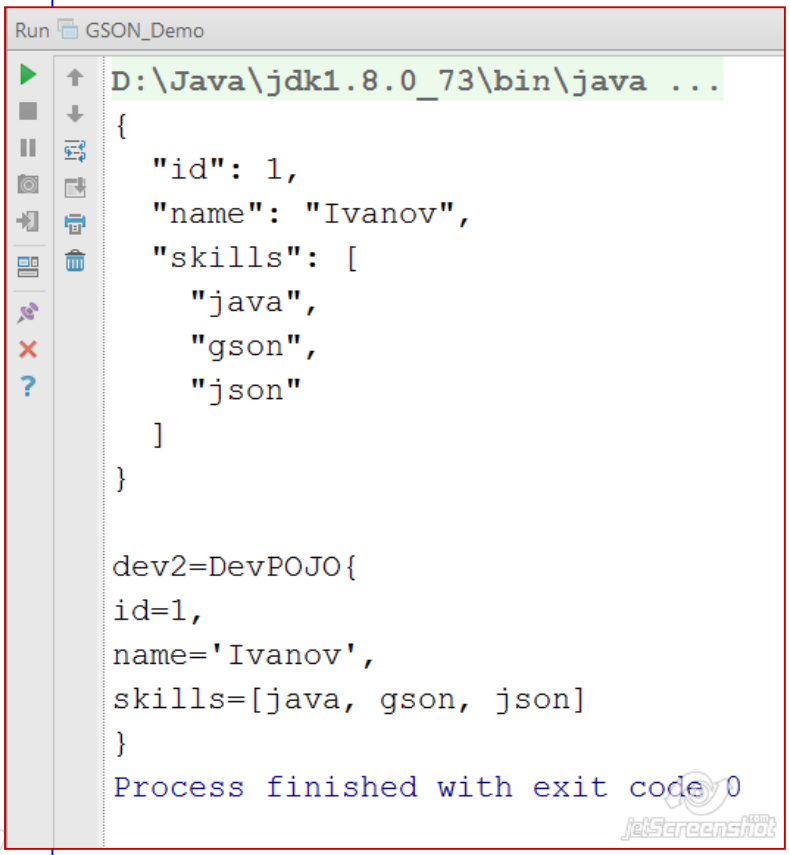


Схема JSON

- **Схема** - JSON-объект описания данных в формате JSON.
- Свойства этого объекта не являются обязательными, каждое их них является инструкцией определённого правила валидации (далее — правило).
- Прежде всего, схема может ограничивать тип данных (правило type или disallow, может быть как строкой, так и массивом):
  - **string** (строка)
  - **number** (число, включая все действительные числа)
  - **integer** (целое число, является подмножеством number)
  - **boolean** (true или false)
  - **object** (объект, в некоторых языках зовётся ассоциативным массивом, хэшем, хэш-таблицей, картой или словарём)
  - **array** (массив)
  - **null** («нет данных» или «не известно», возможно только значение null)
  - **any** (любой тип, включая null)
- Далее, в зависимости от типа проверяемых данных, есть дополнительные правила:
  - данные являются числом, к нему могут быть применены **minimum, maximum, divisibleBy**.
  - данные являются массивом, в силу вступают правила: **minItems, maxItems, uniqueItems, items**.
  - данные являются строкой, применяюся: **pattern, minLength, maxLength**.
  - Объект, рассматриваются правила: **properties, patternProperties, additionalProperties**.

2023

Hibernate

БУГИР. Каф. ЭИ.

Основы Hibernate

- Hibernate – одна из технологий ORM (Object Relational Mapping), реализует технологию JPA (Java Persistence Architecture). Аналоги – iBATIS, Oracle TopLink
- Работа с СУБД происходит через простые java-классы (POJO – Plain Old Java Objects), для которых указывается дополнительная информация (в XML или через аннотации)

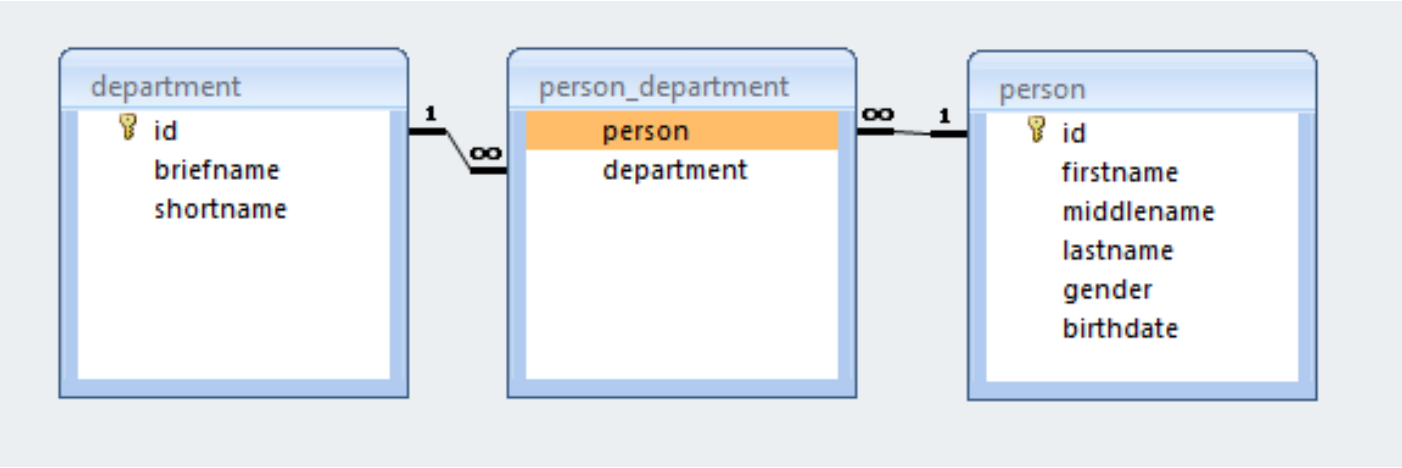
Файлы, необходимые для работы

- Библиотека hibernate (включает: ядро – **core**, аннотации – **annotation**)
- Файл конфигурации (**hibernate.cfg.xml**), в котором указаны общие параметры для hibernate и список классов для ORM
- Классы POJO для воспроизведения структуры БД и хранения данных

Пример hibernate.cfg.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Драйвер СУБД -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <!-- Строка подключения -->
        <property name="hibernate.connection.url">jdbc:mysql://localhost:2016/pisl</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <!-- hbm2ddl.auto: create, create-drop, update -->
        <property name="hibernate.hbm2ddl.auto">create</property>
        <!-- DerbyDialect, MySQLDialect, Oracle8iDialect, Oracle10gDialect, HSQLDialect -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <!-- вывод SQL-выражений на System.out -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
        <!-- jta, managed, thread - контекст, в котором поддерживаются транзакции -->
        <!--<property name="current_session_context_class">thread</property>-->

        <!-- Список классов для проецирования в СУБД -->
        <mapping class="by.it.pisl.lr3.Department"/>
        <mapping class="by.it.pisl.lr3.Person"/>
    </session-factory>
</hibernate-configuration>
```

Целевая структура БД



Класс Person

```
import java.util.*;
import javax.persistence.*;

@Entity
@Table(name="PERSON")
public class Person implements java.io.Serializable{

    @Id()
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    private String firstname;
    private String middlename;
    private String lastname;
    private String gender;
    @Temporal(value = TemporalType.DATE)
    private Date bdate;
    @ManyToMany(cascade=CascadeType.ALL)
    @JoinTable(name = "PERSON_DEPARTMENT",
            joinColumns=@JoinColumn(name="PERSON"),
            inverseJoinColumns=@JoinColumn(name="DEPARTMENT")
    )
    private List<Department> departments = new ArrayList<~>();
```

Класс Department

```
import java.util.*;
import javax.persistence.*;
@Entity
public class Department {

    @Id()
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    private String briefname;
    private String fullname;
    @ManyToMany(cascade=CascadeType.ALL)
    @JoinTable(name = "PERSON_DEPARTMENT",
            joinColumns = @JoinColumn(name = "DEPARTMENT"),
            inverseJoinColumns = @JoinColumn(name = "PERSON")
    )
    private List<Person> persons = new ArrayList<~>();

    public Department() {
    }

    Department(String briefname, String fullname) {
        this.briefname = briefname;
        this.fullname = fullname;
    }
```

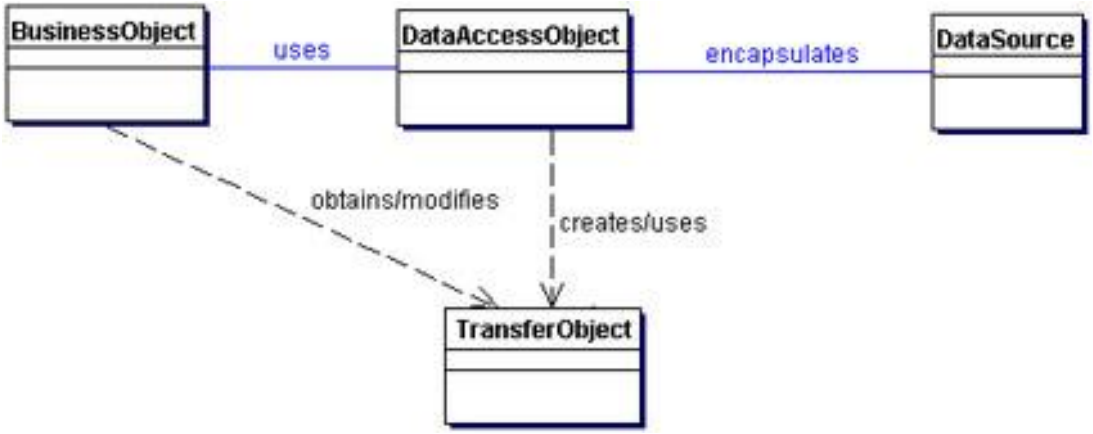
Класс HibernateUtil

```java
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

class HibernateUtil {
    private static SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)
            // config file.
            Configuration cfg=new Configuration().configure();
            StandardServiceRegistryBuilder builder=new StandardServiceRegistryBuilder()
                    .applySettings(cfg.getProperties());
            sessionFactory = cfg.buildSessionFactory(builder.build());

        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Data Access Object

http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

DataSource – наши классы-сущности (POJO)



Класс OrganizationDAO

```java
public class OrganizationDAO {

    private static OrganizationDAO organizationDAO;

    private OrganizationDAO() {
    }

    static OrganizationDAO getInstance() {
        if (organizationDAO==null)
            organizationDAO=new OrganizationDAO();
        return organizationDAO;
    }

    private Session s;

    void beginTransaction() throws Exception {
        if (s != null) throw new Exception("Session already opened!");
        s = HibernateUtil.getSessionFactory().openSession();
        s.beginTransaction();
    }

    void commit() throws Exception {
        try {
            s.getTransaction().commit();
            s.close();
        } finally {
            try {
                s.close();
            } catch (Exception exc) {
            }
            s = null;
        }
    }

    public void rollback() throws Exception {
        s.getTransaction().rollback();
        s.close();
    }

    List<Person> getPesons() throws Exception {
        return (List<Person>) s.createQuery("select p from Person p").list();
    }

    public Person getPerson(long id) throws Exception {
        return (Person) s.createQuery("select p from Person p where p.id=?")
                .setLong(0, id).uniqueResult();
    }

    void savePerson(Person p) throws Exception {
        s.saveOrUpdate(p);
    }

    void saveDepartment(Department d) throws Exception {
        s.saveOrUpdate(d);
    }

    void close(){
        if (s!=null)
            s.close();
    }

    void destruct(){
        HibernateUtil.getSessionFactory().close();
    }
}
```

Пример – добавление сотрудника

```
Person p = new Person();
p.setFirstname("oleg");
p.setLastname("ivanov");
p.setGender("M");
p.setBdate( new java.text.SimpleDateFormat("dd.mm.yyyy").parse("01.01.1980") );

Department dep = new Department("ЭИ", "Кафедра экономической информатики");
dep.getPersons().add( p );
p.setDepartments(new ArrayList<Department>(Arrays.asList(dep)));
OrganizationDAO dao = OrganizationDAO.getInstance();
dao.beginTransaction();
dao.savePerson( p );
dao.saveDepartment( dep );
dao.commit();
dao.close();
```

Выбор всех сотрудников

```
OrganizationDAO dao = OrganizationDAO.getInstance();
dao.beginTransaction();
List<Person> persons = dao.getPesons();

for( Person p : persons ){
    System.out.println( " - " +
            p.getFirstname() + " " + p.getLastname() );
    System.out.println( "\tDepartments:" );
    for( Department dep : p.getDepartments() )
        System.out.println( "\t - [" + dep.getBriefname() + "] " +
                dep.getFullname() );
}

dao.commit();
dao.close();
```

Информация по Hibernate

- Настройка ORM на основе XML-конфигурации: http://docs.jboss.org/hibernate/stable/core/reference/en/html/index.html

- Настройка ORM на основе аннотаций: http://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/

Задание

Наименование поля	Тип поля	Обязательное (да, нет)	**Вариант N**
Фамилия	Текстовый	Да	Любой
Имя	Текстовый	Да	Любой
Отчество	Текстовый	Да	Любой
Дата рождения	Дата	Да	Любой
Пол	Boolean	Да	**Четный**
Серия паспорта	Текстовый	Да	Любой
№ паспорта	Текстовый (с маской)	Да	Любой
Идент. номер	Текстовый (с маской)	Нет	**Четный**
Город проживания	**Список** (от 5 городов)	Да	Любой
Адрес факт.проживания	Текстовый	Да	Любой
Телефон дом	Текстовый (с маской)	Нет	**Четный**
Телефон моб	Текстовый (с маской)	Нет	Любой
E-mail	Текстовый	Нет	**Нечетный**
Трудостроен	Boolean	Нет	**Нечетный**
Должность	Текстовый	Нет	**Нечетный**
Город прописки	**Список** (от 5 городов)	Да	**1,4,7... (N%3==1)**
Адрес прописки	Текстовый	Да	**2,5,8... (N%3==2)**
Гражданство	**Список**	Да	Любой
Военнообязанный	Boolean	Да	**3,6,9... (N%3==0)**

Задание

1. Выбрать поля базы данных для ввода и хранения информации о клиентах для выполнения лабораторной работы согласно **своему номеру варианта**.
2. Реализовать XML-файл с данными, его XSD-схему и API для добавления, чтения, обновления и удаления (CRUD) клиентов в СУБД из XML.
3. Разработать схему данных в которой для элементов типа **Список** предусмотрены отдельные справочники.
4. **A.** На оценку **6 баллов** реализовать схему данных в выбранной СУБД используя генератор классов **xjc** (**JAXB)** и **hibernate**.
5. **B.** На оценку **8 баллов** дополнительно реализовать обратное преобразование (чтение из СУБД и маршаллинг в XML).
6. **C.** На оценку **10 баллов** предусмотреть аналогичное преобразование в JSON и обратно (рекомендуется gson).
7. На защите продемонстрировать операции преобразований из консоли, либо (рекомендуется) показать соответствующее покрытие тестами JUnit.