Частное учреждение образования

«Колледж бизнеса и права»

Лабораторная работа № 17

Учащийся Е. А. Чужавко

группа: Т-992

2022

**Задание.**

1. Создать файл XML и соответствующую ему схему XSD.

2. При разработке XSD использовать простые и комплексные типы, перечисления, шаблоны и предельные значения.

3. Создать класс, соответствующий данному описанию.

4. Создать приложение для разбора XML-документа и инициализации коллекции объектов информацией из XML-файла. Для разбора использовать SAX, DOM или StAX парсеры. Для сортировки объектов использовать интерфейс Comparator.

5. Произвести проверку XML-документа с привлечением XSD.

6. Определить метод, производящий преобразование разработанного XML документа в документ, указанный в каждом задании.

**Вариант 22**

Схема: – Name – название конфеты; – Energy – калорийность (ккал); – Type (должно быть несколько) – тип конфеты (карамель, ирис, шоколадная [с начинкой либо нет]); – Ingredients (должно быть несколько) – ингредиенты: вода, сахар (в мг), фруктоза (в мг), тип шоколада (для шоколадных), ванилин (в мг); – Value – пищевая ценность: белки (в г), жиры (в г) и углеводы (в г); – Production – предприятие-изготовитель. Корневой элемент назвать Candy. С помощью XSL преобразовать XML-файл в формат HTML.

Результат работы программы представлен на рисунках 1, 2, 3, 4.

package com.mkyong.xml.dom;

import org.w3c.dom.CDATASection;

import org.w3c.dom.Comment;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.\*;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import java.io.\*;

public class CreateXmlFile {

public static void main(String[] args) throws ParserConfigurationException,

TransformerException {

DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

// root elements

Document doc = docBuilder.newDocument();

Element rootElement = doc.createElement("Candy");

doc.appendChild(rootElement);

Element candy=doc.createElement("candy");

Element ingredients=doc.createElement("ingredients");

Element energy=doc.createElement("energy");

energy.setTextContent("44");

Element type=doc.createElement("type");

type.setTextContent("chocolate");

Element name=doc.createElement("name");

name.setTextContent("kitKat");

candy.appendChild(name);

candy.appendChild(energy);

candy.appendChild(type);

Element water=doc.createElement("water");

water.setTextContent("5");

Element sugar=doc.createElement("sugar");

sugar.setTextContent("5");

Element fruktoza=doc.createElement("fruktoza");

fruktoza.setTextContent("5");

Element typechocholate=doc.createElement("typechocolate");

typechocholate.setTextContent("chocolate");

Element vanilin=doc.createElement("vanilin");

vanilin.setTextContent("5");

ingredients.appendChild(water);

ingredients.appendChild(sugar);

ingredients.appendChild(fruktoza);

ingredients.appendChild(typechocholate);

ingredients.appendChild(vanilin);

candy.appendChild(ingredients);

Element value=doc.createElement("value");

Element belki=doc.createElement("belki");

belki.setTextContent("5");

Element jiri=doc.createElement("jiri");

jiri.setTextContent("31");

Element uglevodi=doc.createElement("uglevodi");

uglevodi.setTextContent("61");

value.appendChild(belki);

value.appendChild(jiri);

value.appendChild(uglevodi);

Element production=doc.createElement("production");

production.setTextContent("Nestle");

candy.appendChild(value);

candy.appendChild(production);

rootElement.appendChild(candy);

Element candy2=doc.createElement("candy");

Element ingredients2=doc.createElement("ingredients");

Element energy2=doc.createElement("energy");

energy2.setTextContent("44");

Element type2=doc.createElement("type");

type2.setTextContent("chocolate");

Element name2=doc.createElement("name");

name2.setTextContent("Red shapka");

candy2.appendChild(name2);

candy2.appendChild(energy2);

candy2.appendChild(type2);

Element water2=doc.createElement("water");

water2.setTextContent("6");

Element sugar2=doc.createElement("sugar");

sugar2.setTextContent("6");

Element fruktoza2=doc.createElement("fruktoza");

fruktoza2.setTextContent("6");

Element typechocholate2=doc.createElement("typechocolate");

typechocholate2.setTextContent("milk");

Element vanilin2=doc.createElement("vanilin");

vanilin2.setTextContent("6");

ingredients2.appendChild(water2);

ingredients2.appendChild(sugar2);

ingredients2.appendChild(fruktoza2);

ingredients2.appendChild(typechocholate2);

ingredients2.appendChild(vanilin2);

candy2.appendChild(ingredients2);

Element value2=doc.createElement("value");

Element belki2=doc.createElement("belki");

belki2.setTextContent("10");

Element jiri2=doc.createElement("jiri");

jiri2.setTextContent("35");

Element uglevodi2=doc.createElement("uglevodi");

uglevodi2.setTextContent("50");

value2.appendChild(belki2);

value2.appendChild(jiri2);

value2.appendChild(uglevodi2);

Element production2=doc.createElement("production");

production2.setTextContent("communarka");

candy2.appendChild(value2);

candy2.appendChild(production2);

rootElement.appendChild(candy2);

DOMSource source = new DOMSource(doc);

StreamResult file=new StreamResult(new File("CandyCode.xml"));

TransformerFactory transformerFactory = TransformerFactory.newInstance();

Transformer transformer = transformerFactory.newTransformer();

transformer.transform(source, file);

writeXml(doc,System.out);

}

private static void writeXml(Document doc,

OutputStream output)

throws TransformerException {

TransformerFactory transformerFactory = TransformerFactory.newInstance();

Transformer transformer = transformerFactory.newTransformer();

// pretty print

transformer.setOutputProperty(OutputKeys.INDENT, "yes");

DOMSource source = new DOMSource(doc);

StreamResult result = new StreamResult(output);

transformer.transform(source, result);

}}

import java.io.File;

import java.io.IOException;

import javax.xml.XMLConstants;

import javax.xml.transform.stream.StreamSource;

import javax.xml.validation.Schema;

import javax.xml.validation.SchemaFactory;

import javax.xml.validation.Validator;

import org.xml.sax.SAXException;

public class XSDValidator {

public static void main(String[] args) {

System.out.println(validateXMLSchema("Candy.xsd", "CandyCode.xml"));

}

public static boolean validateXMLSchema(String xsdPath, String xmlPath){

try {

SchemaFactory factory =

SchemaFactory.newInstance(XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI);

Schema schema = factory.newSchema(new File(xsdPath));

Validator validator = schema.newValidator();

validator.validate(new StreamSource(new File(xmlPath)));

} catch (IOException | SAXException e) {

System.out.println("Exception: "+e.getMessage());

return false;

}

return true;

}

}

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.stream.StreamResult;

import javax.xml.transform.stream.StreamSource;

public class XmlToHTML {

public static void main(String[] args) {

try {

TransformerFactory tFactory = TransformerFactory.newInstance();

Transformer transformer = tFactory.newTransformer(new StreamSource("Candy.xsl"));

transformer.transform(new StreamSource("CandyCode.xml"), new StreamResult("Candy.html"));

} catch (TransformerConfigurationException e) { e.printStackTrace(); }

catch (TransformerException e) { e.printStackTrace(); }

}

}

import model.\*;

import org.w3c.dom.\*;

import javax.xml.parsers.DocumentBuilderFactory;

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) {

File file=new File("Candy.xml");

DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();

Document doc=null;

try{

doc= dbf.newDocumentBuilder().parse(file);

}catch (Exception e){

e.printStackTrace();

return;

}

Node qwe=doc.getFirstChild();

Node candyNode=null;

NodeList rootChilds=qwe.getChildNodes();

for(int i = 0; i < rootChilds.getLength(); i++){

System.out.println(rootChilds.item(i).getNodeName());

if(rootChilds.item(i).getNodeType()!=Node.ELEMENT\_NODE){

continue;

}

candyNode=rootChilds.item(i);

break;

}

if(candyNode==null){

return;

}

List<Candy> candyList=new ArrayList<>();

NodeList candyChilds = candyNode.getChildNodes();

for(int i =0; i< rootChilds.getLength(); i++){

if(rootChilds.item(i).getNodeType() != Node.ELEMENT\_NODE){

continue;

}

if(!rootChilds.item(i).getNodeName().equals("candy")){

continue;

}

String name="";

int energy=0;

String type="";

Ingredients ingredients=new Ingredients();

Value value=new Value();

String production="";

NodeList elementChild = rootChilds.item(i).getChildNodes();

for(int j = 0; j < elementChild.getLength(); j++){

if(elementChild.item(j).getNodeType() != Node.ELEMENT\_NODE){

continue;

}

switch (elementChild.item(j).getNodeName()){

case "name":{

name=elementChild.item(j).getTextContent();

break;

}

case "energy":{

energy=Integer.valueOf(elementChild.item(j).getTextContent());

break;

}

case "type":{

type=elementChild.item(j).getTextContent();

break;

}

case "ingredients":{

NodeList cqw=elementChild.item(j).getChildNodes();

int water=0;

int sugar=0;

int fruktoza=0;

String typechocolate="";

int vanilin=0;

for(int k=0;k<cqw.getLength();k++){

if(cqw.item(k).getNodeType() != Node.ELEMENT\_NODE){

continue;

}

switch (cqw.item(k).getNodeName()){

case "water":{

water=Integer.valueOf(cqw.item(k).getTextContent());

break;

}

case "sugar":{

sugar=Integer.valueOf(cqw.item(k).getTextContent());

break;

}

case "fruktoza":{

fruktoza=Integer.valueOf(cqw.item(k).getTextContent());

break;

}

case "typechocolate":{

typechocolate=cqw.item(k).getTextContent();

break;

}

case "vanilin":{

vanilin=Integer.valueOf(cqw.item(k).getTextContent());

break;

}

default:{

break;

}

}

}

ingredients=new Ingredients(water,sugar,fruktoza,typechocolate,vanilin);

}

case "value":{

NodeList cqw=elementChild.item(j).getChildNodes();

int belki=0;

int jiri=0;

int uglevodi=0;

for(int k=0;k<cqw.getLength();k++){

if(cqw.item(k).getNodeType() != Node.ELEMENT\_NODE){

continue;

}

switch (cqw.item(k).getNodeName()){

case "belki":{

belki=Integer.valueOf(cqw.item(k).getTextContent());

break;

}

case "jiri":{

jiri=Integer.valueOf(cqw.item(k).getTextContent());

break;

}

case "uglevodi":{

uglevodi=Integer.valueOf(cqw.item(k).getTextContent());

break;

}

}

}

value=new Value(belki,jiri,uglevodi);

}

case "production":{

production=elementChild.item(j).getTextContent();

break;

}

}

}

Candy candy=new Candy(name,energy,type,ingredients,value,production);

candyList.add(candy);

}

System.out.println("Список конфет не отсортированных по названию:");

System.out.println(candyList);

Comparator<Candy> sortbyproduction=new Comparator<Candy>() {

@Override

public int compare(Candy o1, Candy o2) {

return o2.getProduction().compareTo(o1.getProduction());

}

};

Collections.sort(candyList,sortbyproduction);

System.out.println("\n\nСписок конфет после сортировки:");

System.out.println(candyList);

}

}

package model;

public class Candy {

private String name;

private int energy;

private String type;

private Ingredients ingredients;

private Value value;

private String production;

public Value getValue() {

return value;

}

public void setValue(Value value) {

this.value = value;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getEnergy() {

return energy;

}

public void setEnergy(int energy) {

this.energy = energy;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public Ingredients getIngredients() {

return ingredients;

}

public void setIngredients(Ingredients ingredients) {

this.ingredients = ingredients;

}

public String getProduction() {

return production;

}

public void setProduction(String production) {

this.production = production;

}

public Candy(String name, int energy, String type, Ingredients ingredients, Value value, String production) {

this.name = name;

this.energy = energy;

this.type = type;

this.ingredients = ingredients;

this.value = value;

this.production = production;

}

@Override

public String toString() {

return "Candy{" +

"name='" + name + '\'' +

", energy=" + energy +

", type='" + type + '\'' +

", ingredients=" + ingredients +

", value=" + value +

", production='" + production + '\'' +

'}'+'\n';

}

}

package model;

public class Ingredients {

private int water;

private int sugar;

private int fruktoza;

private String typechocoloate;

private int vanilin;

public Ingredients(int water, int sugar, int fruktoza, String typechocoloate, int vanilin) {

this.water = water;

this.sugar = sugar;

this.fruktoza = fruktoza;

this.typechocoloate = typechocoloate;

this.vanilin = vanilin;

}

public Ingredients(){}

public int getWater() {

return water;

}

public void setWater(int water) {

this.water = water;

}

public int getSugar() {

return sugar;

}

public void setSugar(int sugar) {

this.sugar = sugar;

}

public int getFruktoza() {

return fruktoza;

}

public void setFruktoza(int fruktoza) {

this.fruktoza = fruktoza;

}

public String getTypechocoloate() {

return typechocoloate;

}

public void setTypechocoloate(String typechocoloate) {

this.typechocoloate = typechocoloate;

}

public int getVanilin() {

return vanilin;

}

public void setVanilin(int vanilin) {

this.vanilin = vanilin;

}

@Override

public String toString() {

return "Ingredients{" +

"water=" + water +

", sugar=" + sugar +

", fruktoza=" + fruktoza +

", typechocoloate='" + typechocoloate + '\'' +

", vanilin=" + vanilin +

'}';

}

}

package model;

public class Value {

private int belki;

private int jiri;

private int uglevodi;

public int getBelki() {

return belki;

}

public void setBelki(int belki) {

this.belki = belki;

}

public int getJiri() {

return jiri;

}

public void setJiri(int jiri) {

this.jiri = jiri;

}

public int getUglevodi() {

return uglevodi;

}

public void setUglevodi(int uglevodi) {

this.uglevodi = uglevodi;

}

public Value(int belki, int jiri, int uglevodi) {

this.belki = belki;

this.jiri = jiri;

this.uglevodi = uglevodi;

}

public Value(){}

@Override

public String toString() {

return "Value{" +

"belki=" + belki +

", jiri=" + jiri +

", uglevodi=" + uglevodi +

'}';

}

}

<?xml version="1.0" encoding="UTF-8" ?>

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

<xsl:output method="html" doctype-public="XSLT-compat" omit-xml-declaration="yes" encoding="UTF-8" indent="yes" />

<xsl:template match="Candy">

<hmtl>

<head>

<title>Candies</title>

</head>

<body>

<xsl:for-each select="candy">

<ul>

<li>

<xsl:value-of select="name"/>

</li>

<li>

<xsl:value-of select="energy"/>

</li>

<li>

<xsl:value-of select="type"/>

</li>

<li>

<xsl:for-each select="ingredients">

<ul>

<li>

<xsl:value-of select="water"/>

</li>

<li>

<xsl:value-of select="sugar"/>

</li>

<li>

<xsl:value-of select="fruktoza"/>

</li>

<li>

<xsl:value-of select="typechocolate"/>

</li>

<li>

<xsl:value-of select="vanilin"/>

</li>

</ul>

</xsl:for-each>

</li>

<li>

<xsl:for-each select="value">

<ul>

<li>

<xsl:value-of select="belki"/>

</li>

<li>

<xsl:value-of select="jiri"/>

</li>

<li>

<xsl:value-of select="uglevodi"/>

</li>

</ul>

</xsl:for-each>

</li>

<li>

<xsl:value-of select="production"/>

</li>

</ul>

<br></br>

</xsl:for-each>

</body>

</hmtl>

</xsl:template>

</xsl:transform>

<!DOCTYPE html PUBLIC "XSLT-compat">

<hmtl>

<head>

<META http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Candies</title>

</head>

<body>

<ul>

<li>kitKat</li>

<li>44</li>

<li>chocolate</li>

<li>

<ul>

<li>5</li>

<li>5</li>

<li>5</li>

<li>chocolate</li>

<li>5</li>

</ul>

</li>

<li>

<ul>

<li>5</li>

<li>31</li>

<li>61</li>

</ul>

</li>

<li>Nestle</li>

</ul>

<br>

<ul>

<li>Red shapka</li>

<li>44</li>

<li>chocolate</li>

<li>

<ul>

<li>6</li>

<li>6</li>

<li>6</li>

<li>milk</li>

<li>6</li>

</ul>

</li>

<li>

<ul>

<li>10</li>

<li>35</li>

<li>50</li>

</ul>

</li>

<li>communarka</li>

</ul>

<br>

</body>

</hmtl>

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="Candy" type="CandyType"/>

<xs:complexType name="ingredientsType">

<xs:sequence>

<xs:element name="water">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="5"/>

<xs:enumeration value="6"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="sugar">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="5"/>

<xs:enumeration value="6"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="fruktoza">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="5"/>

<xs:enumeration value="6"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element type="xs:string" name="typechocolate"/>

<xs:element name="vanilin">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="5"/>

<xs:enumeration value="6"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:sequence>

</xs:complexType>

<xs:complexType name="valueType">

<xs:sequence>

<xs:element name="belki">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="5"/>

<xs:enumeration value="10"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="jiri">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="31"/>

<xs:enumeration value="35"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="uglevodi">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="61"/>

<xs:enumeration value="50"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:sequence>

</xs:complexType>

<xs:complexType name="candyType">

<xs:sequence>

<xs:element name="name">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="kitKat"/>

<xs:enumeration value="Red shapka"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element type="xs:string" name="energy"/>

<xs:element type="xs:string" name="type"/>

<xs:element type="ingredientsType" name="ingredients"/>

<xs:element type="valueType" name="value"/>

<xs:element name="production">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="Nestle"/>

<xs:enumeration value="communarka"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:sequence>

</xs:complexType>

<xs:complexType name="CandyType">

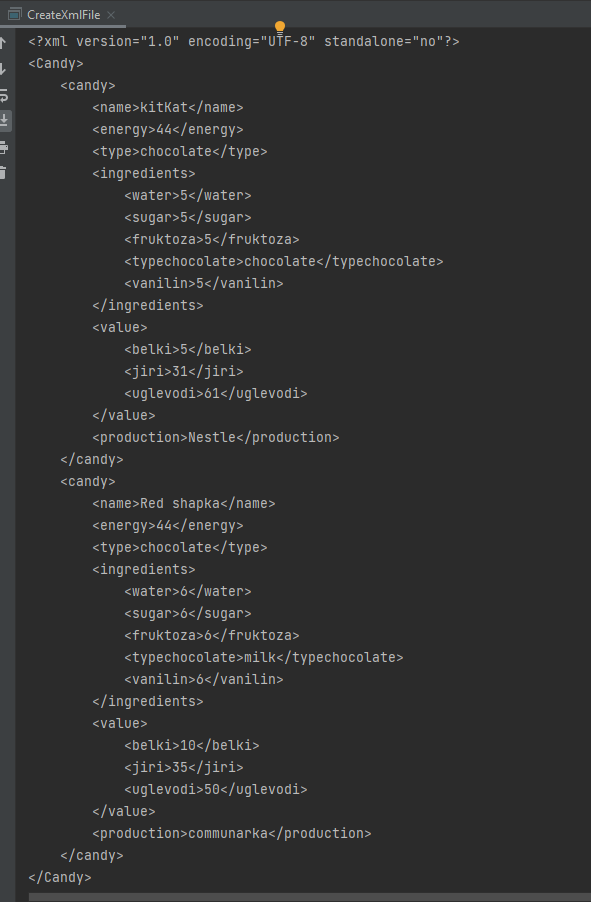
<xs:sequence>

<xs:element type="candyType" name="candy" maxOccurs="unbounded" minOccurs="0"/>

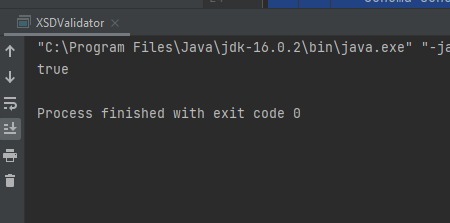
</xs:sequence>

</xs:complexType>

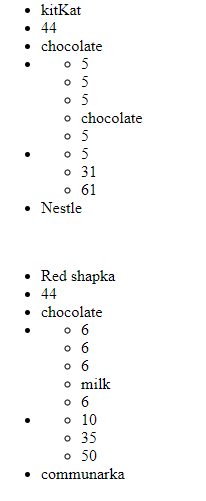
</xs:schema>



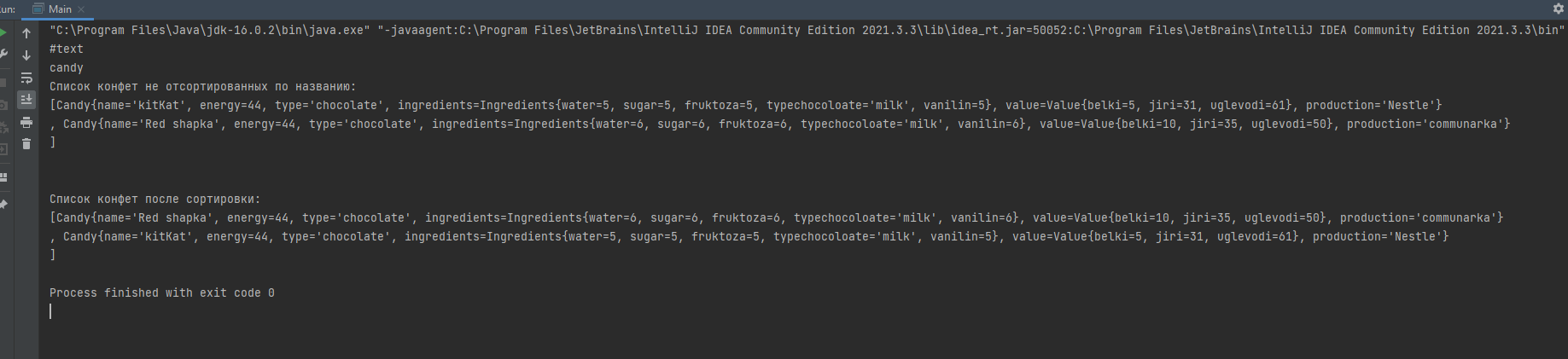
*Рисунок 1.*

**

*Рисунок 2.*

**

*Рисунок 3.*

**

*Рисунок 4.*