**CSC 285 Session 8**

**Introduction to XML**

**===================================================**

**Products.xml**



**Product.java**

import java.text.NumberFormat;

public class Product {

private String code;

private String id;

private String description;

private int amount;

private double price;

private char discount;

public Product() {

this("", "", "", 0, 0, ' ');

}

public Product( String code,String id, String description, int amount, double price, char discount) {

this.code = code;

this.id = id;

this.description = description;

this.amount = amount;

this.price = price;

this.discount = discount;

}

public void setCode(String code) {

this.code = code;

}

public String getCode() {

return code;

}

public void setId(String id) {

this.id = id;

}

public String getId() {

return id;

}

public void setDescription(String description) {

this.description = description;

}

public String getDescription() {

return description;

}

public void setAmount(int amount) {

this.amount = amount;

}

public double getAmount() {

return amount;

}

public String getAmountString() {

String amountString = Integer.toString(amount);

return amountString;

}

public void setPrice(double price) {

this.price = price;

}

public double getPrice() {

return price;

}

public String getFormattedPrice() {

NumberFormat currency = NumberFormat.getCurrencyInstance();

return currency.format(price);

}

public void setDiscount(char discount) {

this.discount = discount;

}

public char getDiscount() {

return discount;

}

public boolean equals(Object object) {

if (object instanceof Product) {

Product product2 = (Product) object;

if ( code.equals(product2.getCode())

&&id.equals(product2.getId())

&& description.equals(product2.getDescription())

&& amount == product2.getAmount()

&& price == product2.getPrice()

&& discount == product2.getDiscount()) {

return true;

}

}

return false;

}

public String toString() {

return "Code: " + code + "\n"

+ "Id: " + getId()+"\n"

+ "Description: " + description + "\n"

+ "amount: " + amount + "\n"

+ "Price: " + this.getFormattedPrice() + "\n"

+ "discount: " + discount;

}

}

**ProductDAO.com**

public interface ProductDAO extends ProductReader, ProductWriter, ProductConstants

{

// all methods from the ProductReader and ProductWriter interfaces

// all static constants from the ProductConstants interface

}

**ProductReader.java**

import java.util.ArrayList;

public interface ProductReader

{

Product getProduct(String code);

ArrayList<Product> getProducts();

}

**ProductWriter.java**

public interface ProductWriter

{

boolean addProduct(Product p);

boolean updateProduct(Product p);

boolean deleteProduct(Product p);

}

**ProductConstants.java**

public interface ProductConstants

{

int CODE\_SIZE = 4;

int DESCRIPTION\_SIZE = 20;

}

**ProductMaintApp.java**

import java.util.Scanner;

import java.util.ArrayList;

public class ProductMaintApp implements ProductConstants {

// declare two class variables

private static ProductDAO productDAO = null;

private static Scanner sc = null;

public static void main(String args[]) {

// display a welcome message

System.out.println("Welcome to the Product Maintenance application\n");

// set the class variables

productDAO = DAOFactory.getProductDAO();

sc = new Scanner(System.in);

// display the command menu

displayMenu();

// perform 1 or more actions

String action = "";

while (!action.equalsIgnoreCase("exit")) {

// get the input from the user

action = Validator.getString(sc,

"Enter a command: ");

System.out.println();

if (action.equalsIgnoreCase("list")) {

displayAllProducts();

} else if (action.equalsIgnoreCase("add")) {

addProduct();

} else if (action.equalsIgnoreCase("del") || action.equalsIgnoreCase("delete")) {

deleteProduct();

} else if (action.equalsIgnoreCase("update")) {

updateProduct();

} else if (action.equalsIgnoreCase("help") || action.equalsIgnoreCase("menu")) {

displayMenu();

} else if (action.equalsIgnoreCase("exit")) {

System.out.println("Bye.\n");

} else {

System.out.println("Error! Not a valid command.\n");

}

}

}

public static void displayMenu() {

System.out.println("COMMAND MENU");

System.out.println("list - List all products");

System.out.println("add - Add a product");

System.out.println("del - Delete a product");

System.out.println("update - update a product");

System.out.println("help - Show this menu");

System.out.println("exit - Exit this application\n");

}

public static void displayAllProducts() {

System.out.println("PRODUCT LIST");

ArrayList<Product> products = productDAO.getProducts();

if (products == null) {

System.out.println("Error! Unable to get products.\n");

} else {

Product p = null;

StringBuilder sb = new StringBuilder();

for (int i = 0; i < products.size(); i++) {

p = products.get(i);

sb.append(

StringUtils.padWithSpaces(

p.getCode(), CODE\_SIZE + 4)

+StringUtils.padWithSpaces(

p.getId(), DESCRIPTION\_SIZE + 4)

+ StringUtils.padWithSpaces(

p.getDescription(), DESCRIPTION\_SIZE + 4)

+ StringUtils.padWithSpaces(

p.getAmountString(), CODE\_SIZE + 4)

+ p.getFormattedPrice() + "\t"

+ p.getDiscount() + "\n"

);

}

System.out.println(sb.toString());

}

}

public static void addProduct() {

String code = Validator.getString(

sc, "Enter product code: ");

String id = Validator.getString(

sc, "Enter product ID: ");

String description = Validator.getLine(

sc, "Enter product description: ");

int amount = Validator.getInt(

sc, "Enter product amount: ");

double price = Validator.getDouble(

sc, "Enter price: ");

char discount = Validator.getChar(

sc, "Enter does the item has discount: ");

Product product = new Product();

product.setCode(code);

product.setId(id);

product.setDescription(description);

product.setAmount(amount);

product.setPrice(price);

product.setDiscount(discount);

boolean success = productDAO.addProduct(product);

System.out.println();

if (success) {

System.out.println(description

+ " was added to the database.\n");

} else {

System.out.println("Error! Unable to add product\n");

}

}

public static void deleteProduct() {

String code = Validator.getString(sc,

"Enter product code to delete: ");

Product p = productDAO.getProduct(code);

System.out.println();

if (p != null) {

boolean success = productDAO.deleteProduct(p);

if (success) {

System.out.println(p.getDescription()

+ " was deleted from the database.\n");

} else {

System.out.println("Error! Unable to add product\n");

}

} else {

System.out.println("No product matches that code.\n");

}

}

public static void updateProduct() {

String code = Validator.getString(

sc, "Enter product code: ");

String id = Validator.getString(

sc, "Enter product ID: ");

String description = Validator.getLine(

sc, "Enter product description: ");

int amount = Validator.getInt(

sc, "Enter product amount: ");

double price = Validator.getDouble(

sc, "Enter price: ");

char discount = Validator.getChar(

sc, "Enter does the item has discount: ");

Product product = new Product();

product.setCode(code);

product.setId(id);

product.setDescription(description);

product.setAmount(amount);

product.setPrice(price);

product.setDiscount(discount);

boolean success = productDAO.updateProduct(product);

System.out.println();

if (success) {

System.out.println(description

+ " was added to the database.\n");

} else {

System.out.println("Error! Unable to add product\n");

}

}

}

**DAOFactory.java**

public class DAOFactory

{

// this method maps the ProductDAO interface

// to the appropriate data storage mechanism

public static ProductDAO getProductDAO()

{

ProductDAO pDAO = new ProductXMLFile();

return pDAO;

}

}

**ProductXMLFile.java**

import java.util.\*;

import java.io.\*;

import javax.xml.stream.\*; // StAX API

public class ProductXMLFile implements ProductDAO {

private String productsFilename = "products.xml";

private File productsFile = null;

public ProductXMLFile() {

productsFile = new File(productsFilename);

}

private void checkFile() throws IOException {

// if the file doesn't exist, create it

if (!productsFile.exists()) {

productsFile.createNewFile();

}

}

private boolean saveProducts(ArrayList<Product> products) {

// create the XMLOutputFactory object

XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();

try {

// check the file to make sure it exists

this.checkFile();

// create XMLStreamWriter object

FileWriter fileWriter

= new FileWriter(productsFilename);

XMLStreamWriter writer

= outputFactory.createXMLStreamWriter(fileWriter);

//write the products to the file

writer.writeStartDocument("1.0");

writer.writeStartElement("Products");

for (Product p : products) {

writer.writeStartElement("Product");

writer.writeAttribute("Code", p.getCode());

writer.writeStartElement("ID");

writer.writeCharacters(p.getId());

writer.writeEndElement();

writer.writeStartElement("Description");

writer.writeCharacters(p.getDescription());

writer.writeEndElement();

writer.writeStartElement("Amount");

int amount = (int) p.getAmount();

writer.writeCharacters(Integer.toString(amount));

writer.writeEndElement();

writer.writeStartElement("Price");

double price = p.getPrice();

writer.writeCharacters(Double.toString(price));

writer.writeEndElement();

writer.writeStartElement("Discount");

char discount = p.getDiscount();

writer.writeCharacters(Character.toString(discount));

writer.writeEndElement();

writer.writeEndElement();

}

writer.writeEndElement();

writer.flush();

writer.close();

} catch (IOException e) {

e.printStackTrace();

return false;

} catch (XMLStreamException e) {

e.printStackTrace();

return false;

}

return true;

}

public ArrayList<Product> getProducts() {

ArrayList<Product> products = new ArrayList<Product>();

Product p = null;

// create the XMLInputFactory object

XMLInputFactory inputFactory = XMLInputFactory.newInstance();

try {

// check the file to make sure it exists

this.checkFile();

// create a XMLStreamReader object

FileReader fileReader

= new FileReader(productsFilename);

XMLStreamReader reader

= inputFactory.createXMLStreamReader(fileReader);

// read the products from the file

while (reader.hasNext()) {

int eventType = reader.getEventType();

switch (eventType) {

case XMLStreamConstants.START\_ELEMENT:

String elementName = reader.getLocalName();

if (elementName.equals("Product")) {

p = new Product();

String code = reader.getAttributeValue(0);

p.setCode(code);

}

if (elementName.equals("ID")) {

String id = reader.getElementText();

p.setId(id);

}

if (elementName.equals("Description")) {

String description = reader.getElementText();

p.setDescription(description);

}

if (elementName.equals("Amount")) {

String AmountString = reader.getElementText();

int amount = Integer.parseInt(AmountString);

p.setAmount(amount);

}

if (elementName.equals("Price")) {

String priceString = reader.getElementText();

double price = Double.parseDouble(priceString);

p.setPrice(price);

}

if (elementName.equals("Discount")) {

String discountString = reader.getElementText();

char discount = discountString.charAt(0);

p.setDiscount(discount);

}

break;

case XMLStreamConstants.END\_ELEMENT:

elementName = reader.getLocalName();

if (elementName.equals("Product")) {

products.add(p);

}

break;

default:

break;

}

reader.next();

}

} catch (IOException e) {

e.printStackTrace();

return null;

} catch (XMLStreamException e) {

e.printStackTrace();

return null;

}

return products;

}

public Product getProduct(String code) {

ArrayList<Product> products = this.getProducts();

for (Product p : products) {

if (p.getCode().equals(code)) {

return p;

}

}

return null;

}

public boolean addProduct(Product p) {

ArrayList<Product> products = this.getProducts();

products.add(p);

return this.saveProducts(products);

}

public boolean deleteProduct(Product p) {

ArrayList<Product> products = this.getProducts();

products.remove(p);

return this.saveProducts(products);

}

public boolean updateProduct(Product newProduct) {

ArrayList<Product> products = this.getProducts();

// get the old product and remove it

Product oldProduct = this.getProduct(newProduct.getCode());

int i = products.indexOf(oldProduct);

products.remove(i);

// add the updated product

products.add(i, newProduct);

return this.saveProducts(products);

}

}

**StringUtils.java**

public class StringUtils

{

public static String padWithSpaces(String s, int length)

{

if (s.length() < length)

{

StringBuilder sb = new StringBuilder(s);

while(sb.length() < length)

{

sb.append(" ");

}

return sb.toString();

}

else

{

return s.substring(0, length);

}

}

}

**Validator.java**

**import java.util.Scanner;**

**public class Validator {**

**public static String getLine(Scanner sc, String prompt) {**

**System.out.print(prompt);**

**String s = sc.nextLine(); // read the whole line**

**return s;**

**}**

**public static String getString(Scanner sc, String prompt) {**

**System.out.print(prompt);**

**String s = sc.next(); // read the first string on the line**

**sc.nextLine(); // discard the rest of the line**

**return s;**

**}**

**public static int getInt(Scanner sc, String prompt) {**

**boolean isValid = false;**

**int i = 0;**

**while (isValid == false) {**

**System.out.print(prompt);**

**if (sc.hasNextInt()) {**

**i = sc.nextInt();**

**isValid = true;**

**} else {**

**System.out.println("Error! Invalid integer value. Try again.");**

**}**

**sc.nextLine(); // discard any other data entered on the line**

**}**

**return i;**

**}**

**public static int getInt(Scanner sc, String prompt,**

**int min, int max) {**

**int i = 0;**

**boolean isValid = false;**

**while (isValid == false) {**

**i = getInt(sc, prompt);**

**if (i <= min) {**

**System.out.println(**

**"Error! Number must be greater than " + min);**

**} else if (i >= max) {**

**System.out.println(**

**"Error! Number must be less than " + max);**

**} else {**

**isValid = true;**

**}**

**}**

**return i;**

**}**

**public static double getDouble(Scanner sc, String prompt) {**

**boolean isValid = false;**

**double d = 0;**

**while (isValid == false) {**

**System.out.print(prompt);**

**if (sc.hasNextDouble()) {**

**d = sc.nextDouble();**

**isValid = true;**

**} else {**

**System.out.println("Error! Invalid decimal value. Try again.");**

**}**

**sc.nextLine(); // discard any other data entered on the line**

**}**

**return d;**

**}**

**public static double getDouble(Scanner sc, String prompt,**

**double min, double max) {**

**double d = 0;**

**boolean isValid = false;**

**while (isValid == false) {**

**d = getDouble(sc, prompt);**

**if (d <= min) {**

**System.out.println(**

**"Error! Number must be greater than " + min);**

**} else if (d >= max) {**

**System.out.println(**

**"Error! Number must be less than " + max);**

**} else {**

**isValid = true;**

**}**

**}**

**return d;**

**}**

**public static char getChar(Scanner sc, String prompt) {**

**char c = ' ';**

**boolean isValid = false;**

**while (isValid == false) {**

**System.out.print(prompt);**

**if (sc.hasNext()) {**

**c = sc.findInLine(".").charAt(0);**

**if (c == 'y' || c == 'n') {**

**isValid = true;**

**} else {**

**System.out.println("Error! Invalid decimal value. Try again.");**

**}**

**}**

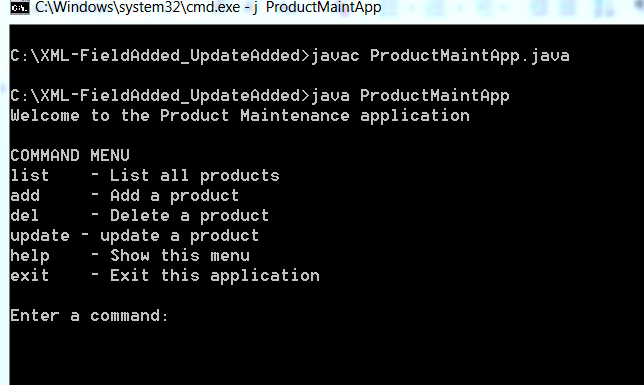
**sc.nextLine(); // discard any other data entered on the line**

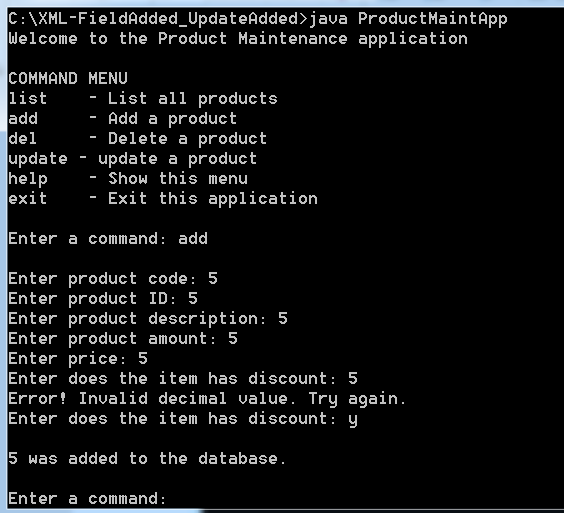
**}**

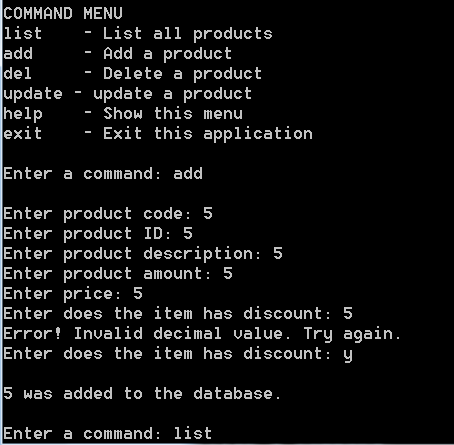
**return c;**

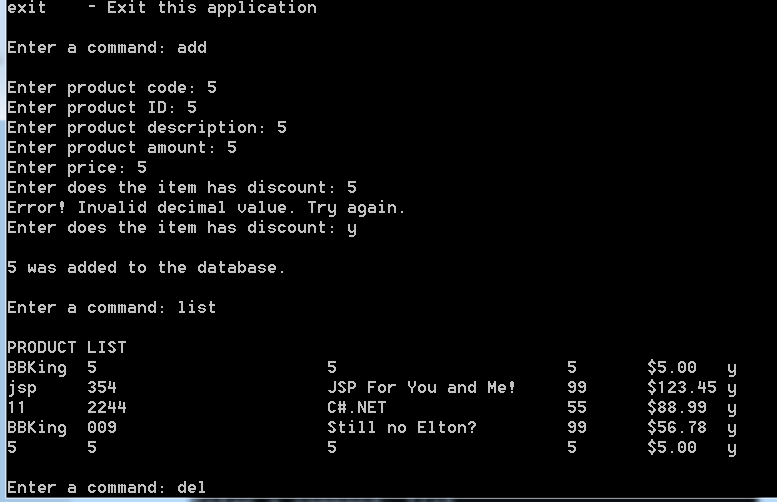
**}**

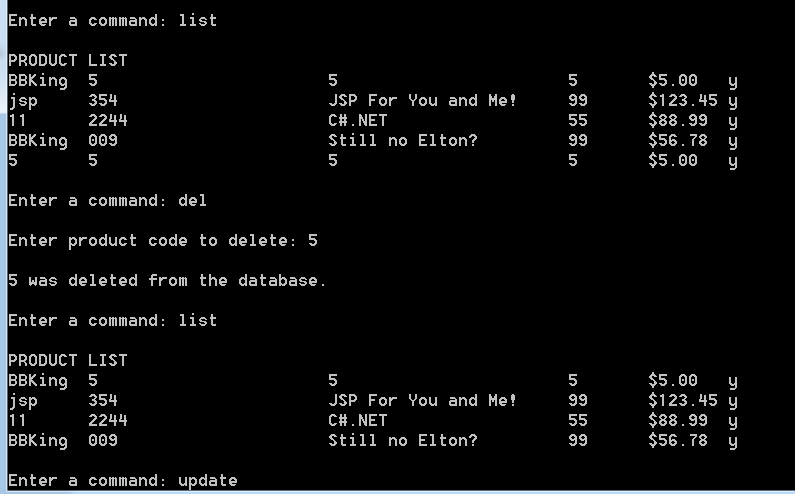
**}**

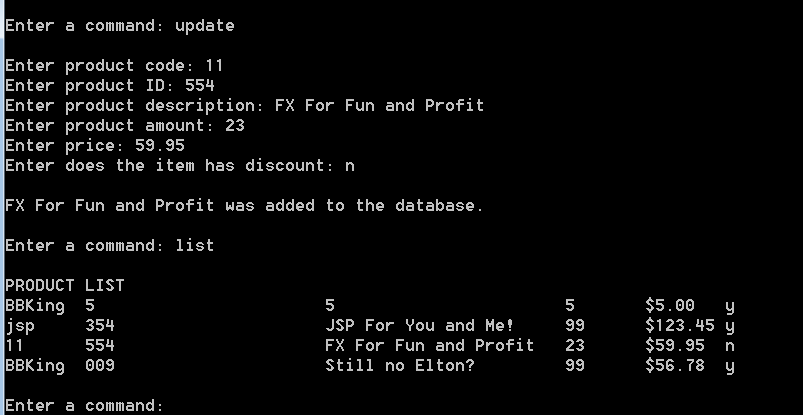


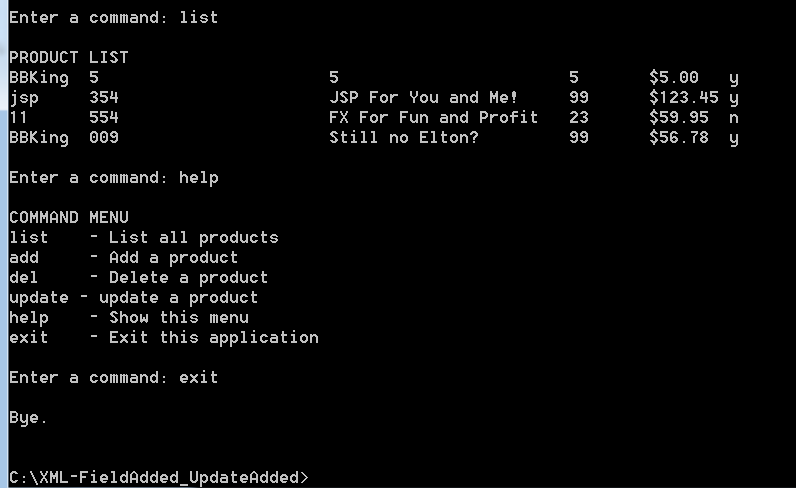












**C’est Finis**