**Administrator documentation**

Contents

[Introduction 3](#_Toc396940324)

[1 Web application structure 4](#_Toc396940325)

[2 Software management tools 5](#_Toc396940326)

[2.1 Build automation tool Maven 5](#_Toc396940327)

[2.2 Jetty server 5](#_Toc396940328)

[3 Design patterns 7](#_Toc396940329)

[3.1 Model-View-Controller 7](#_Toc396940330)

[3.2 Data Access Object 7](#_Toc396940331)

[4 Frameworks 9](#_Toc396940332)

[4.1 Spring 9](#_Toc396940333)

[4.2 Hibernate 13](#_Toc396940334)

[5 API 17](#_Toc396940335)

[5.1 Java Persistence API 17](#_Toc396940336)

[5.2 JDBC 17](#_Toc396940337)

[6 Entity classes 18](#_Toc396940338)

[7 Data Access Object classes 27](#_Toc396940339)

[7.1 JDBC approach 27](#_Toc396940340)

[7.2 Hibernate approach 36](#_Toc396940341)

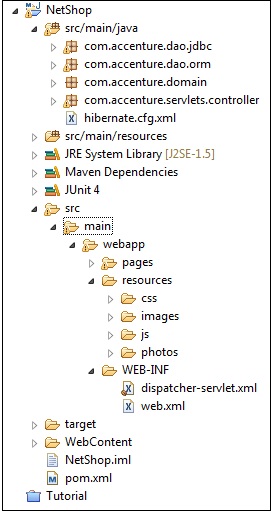
[8 Controllers 40](#_Toc396940342)

[9 JavaServer Pages 43](#_Toc396940343)

Introduction

Project “NetShop” is a web store dedicated for cars selling. In project development are used tools (Eclipse (IDE), Maven, Jetty), frameworks (Spring Framework, Hibernate ORM), API (JPA, JDBC), design patterns (MVC, DAO). For tests are used frameworks Selenium IDE and Mockito. In first part of development was used JDBC for database accessing, but after refactoring now is used Hibernate ORM. Website has 12 pages: main (manufactures brands), models (brand models), products (model cars), product (car description, button “Buy”), registration (user registration), order (car description, button “Confirm & Buy”), ordercompleted (order status), feedback, about (information), stores, support.

# Web application structure



img.1 Project structure

In this project are:

1. 5 POJO classes which are equal to database entities (src/main/java/com.accenture.domain)
2. 5 JDBC DAO classes with 1 connector class (src/main/java/com.accenture.dao.jdbc)
3. 5 Hibernate ORM DAO classes with 1 connector class (src/main/java/com.accenture.dao.orm
4. 5 controllers (src/main/java/com.accenture.servlets.controller)
5. 4 configuration files: hibernate.cfg (src/main/java/), applicationContext (src/main/resources/), dispatcher-servlet (src/main/webapp/WEB-INF), web (src/main/webapp/WEB-INF)
6. 12 JSP pages (src/main/webapp/pages)
7. 3 CSS (src/main/webapp/resources/css)
8. 10 images (src/main/webapp/resources/images)
9. 3 JS (src/main/webapp/resources/js)
10. 184 photos (src/main/webapp/resources/photos)
11. 1 Mockito test (src/test/java/mokito\_tests)
12. 9 Selenium tests (src/test/java/com/Accenture/tests)

# Software management tools

## Build automation tool Maven

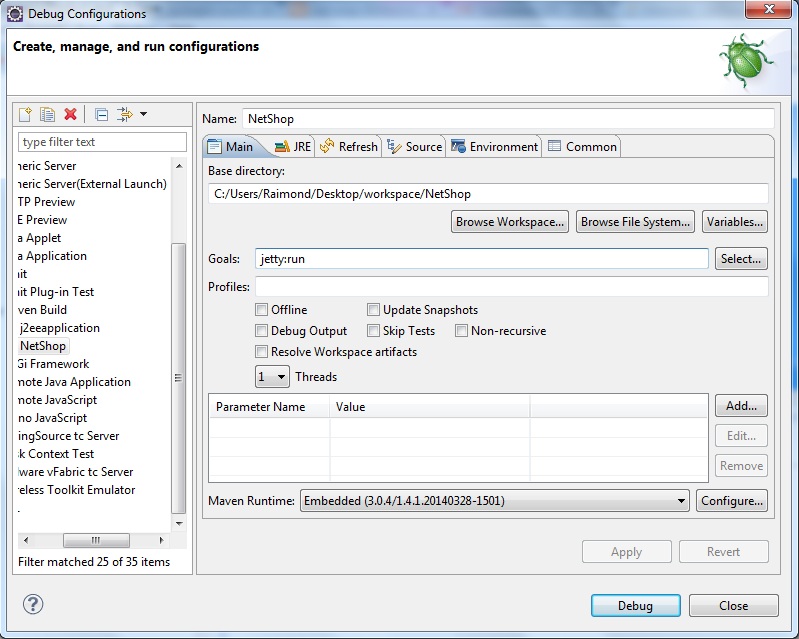
Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: First, it describes how software is built, and second, it describes its dependencies

Maven will generate all the Java’s standard folders structure. Maven project structure and contents are declared in an xml file, pom.xml referred as Project Object Model (POM), which is the fundamental unit of the entire Maven system. It includes: project dependencies, plugins, goals, build profiles.

Maven will download all Hibernate, Spring and MySQL libraries automatically and put into Maven’s local repository. At the same time, Maven will add the downloaded libraries into Eclipse “.classpath” for dependency purpose.

## Jetty server

Jetty is a web (HTTP) server and Servlet container that can be easily configured to serve static and dynamic content. Jetty can be easily embedded into your ownapplications by running Jetty directly from Maven.

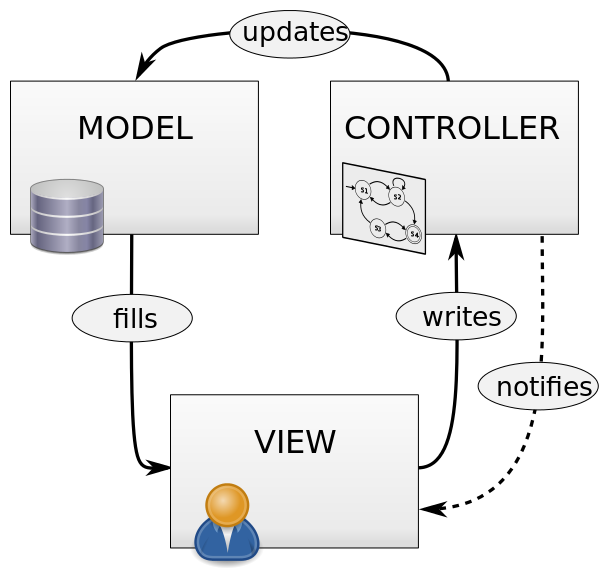


Running Jetty directly from Maven

# Design patterns

## Model-View-Controller

In this web application we used design pattern named MVC which stands for Model-View-Controller. MVC design pattern consists three parts: model, view and controller.



Model-View-Controller diagram

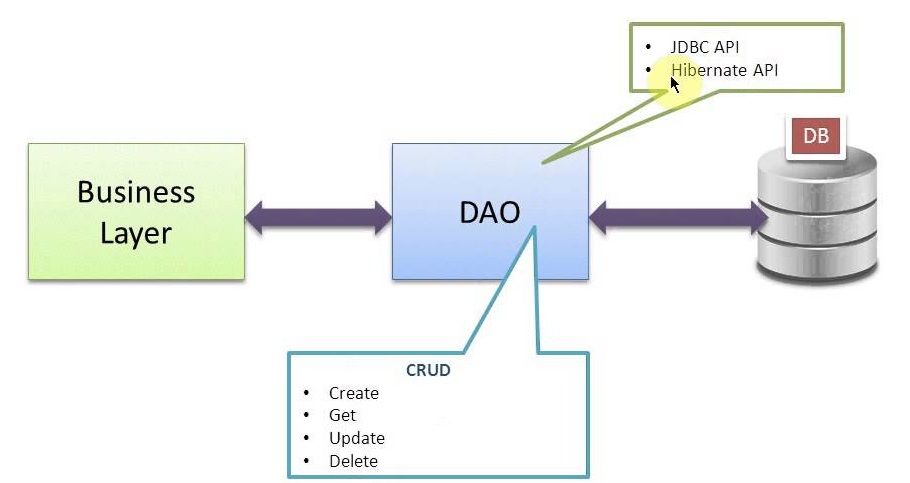
**Model** - The model represents data and the rules that govern access to and updates of this data.

**View** - requests information from the model that it uses to generate an output representation to the user.

**Controller** can send commands to the model to update the model's state. It can also send commands to its associated view to change the view's presentation of the model.

## Data Access Object

Data Access Object (DAO) design pattern to separate low-level data access logic from high-level business logic. The concrete DAO class contains logic for accessing data from a specific data source.



Data Access Object diagram

# Frameworks

## Spring

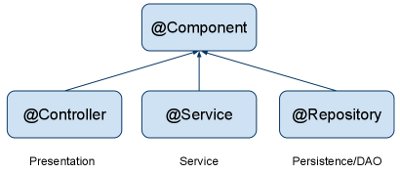
The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform.

To configure web-app to use Spring add the following servlet to web.xml file. Spring configuration file is located in **/WEB-INF/dispatcher-servlet.xml**

In a multitier application, we will have different layers like presentation, service, business, data access etc. When a class is to be annotated for auto-detection by Spring, then we should use the respective stereotype as below.

Annotations denoting the roles of types or methods in the overall architecture

Spring will then automatically detect classes annotated with @Controller, @Component, @Repository, @Service.



Annotation diagram

**@Component** – generic and can be used across application. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

**@Controller** – annotate classes at presentation layers level, mainly used in Spring MVC. This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning. It is typically used in combination with annotated handler methods based on the RequestMapping annotation.

**@Repository** – annotate classes at persistence layer, which will act as database repository. A mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects" implementing pattern Data Access Object.

**@Service**

Annotate all your service classes with @Service. All your business logic should be in Service classes.

|  |
| --- |
| @Service  public class CompanyServiceImpl implements CompanyService {  ...  } |

**@Repository**

Annotate all your DAO classes with @Repository. All your database access logic should be in DAO classes.

@Repository

public class CompanyDAOImpl implements CompanyDAO {

...

}

**@Component**

Annotate your other components (for example REST resource classes) with @Component.

@Component

public class ContactResource {

...

}

**@Controller**

Annotate your controller classes with @Controller.

@Controller

public class CompanyController {

...

}

@Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

Lib-s were used in Contro/aller. java classes ./dao/orm/servlets/controller

and \*Repository.java classes in ….. direfctory

**@Autowired**

Let Spring auto-wire other beans into your classes using @Autowired annotation.

\*@Autowire by default is a type driven injection. @Qualifier spring annotation can be used to further fine-tune autowiring.

@Service

public class CompanyServiceImpl implements CompanyService {

@Autowired

private CompanyDAO companyDAO;

...

}

**@Transactional**

Configure your transactions with @Transactional spring annotation.

@Service

public class CompanyServiceImpl implements CompanyService {

@Autowired

private CompanyDAO companyDAO;

@Transactional

public Company findByName(String name) {

Company company = companyDAO.findByName(name);

return company;

}

...

}

**@RequestMapping**

You use the @RequestMapping spring annotation to map URLs onto an entire class or a particular handler method. Typically the class-level annotation maps a specific request path (or path pattern) onto a form controller, with additional method-level annotations narrowing the primary mapping.

@RequestMethod

Intended for use with the RequestMapping.method() attribute of the RequestMapping annotation.

@Controller

@RequestMapping("/company")

public class CompanyController {

@Autowired

private CompanyService companyService;

...

}

**@PathVariable**

You can use the @PathVariable spring annotation on a method argument to bind it to the value of a URI template variable. In our example below, a request path of /company/techferry will bind companyName variable with 'techferry' value.

@Controller

@RequestMapping("/company")

public class CompanyController {

@Autowired

private CompanyService companyService;

@RequestMapping("{companyName}")

public String getCompany(Map<String, Object> map, @PathVariable String companyName) {

Company company = companyService.findByName(companyName);

map.put("company", company);

return "company";

}

...

}

**@RequestParam**

You can bind request parameters to method variables using spring annotation @RequestParam.

@Controller

@RequestMapping("/company")

public class CompanyController {

@Autowired

private CompanyService companyService;

@RequestMapping("/companyList")

public String listCompanies(Map<String, Object> map, @RequestParam int pageNum) {

map.put("pageNum", pageNum);

map.put("companyList", companyService.listCompanies(pageNum));

return "companyList";

}

...

}

Similarly, you can use spring annotation @RequestHeader to bind request headers.

**@SessionAttributes**

@SessionAttributes spring annotation declares session attributes. This will typically list the names of model attributes which should be transparently stored in the session, serving as form-backing beans between subsequent requests.

@Controller

@RequestMapping("/company")

@SessionAttributes("company")

public class CompanyController {

@Autowired

private CompanyService companyService;

...

}

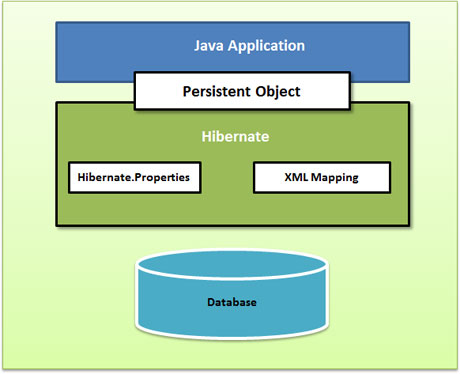
## Hibernate

Hibernate ORM is an object-relational mapping library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.

Hibernate's primary feature is mapping from Java classes to database tables. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from manual result set handling and object conversion. Applications using Hibernate are portable to supported SQL databases with little performance overhead.

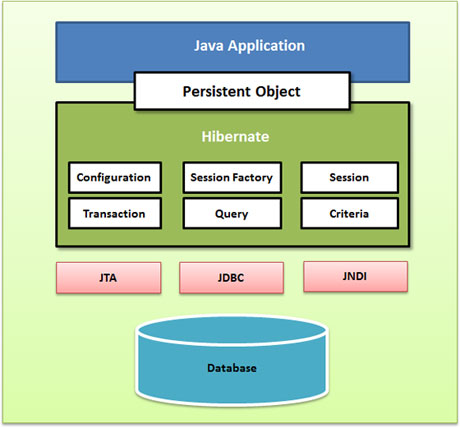
The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

Following is a very high level view of the Hibernate Application Architecture.



Hibernate High Level View

Following is a detailed view of the Hibernate Application Architecture with few important core classes.



Hibernate Architecture

Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Following section gives brief description of each of the class objects involved in Hibernate Application Architecture.

**Configuration Object:**

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys components:

* Database Connection: This is handled through one or more configuration files supported by Hibernate. These files are hibernate.properties and hibernate.cfg.xml.
* Class Mapping Setup

This component creates the connection between the Java classes and database tables..

**SessionFactory Object:**

Configuration object is used to create a SessionFactory object which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

**Session Object:**

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

**Transaction Object:**

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

**Query Object:**

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

**Criteria Object:**

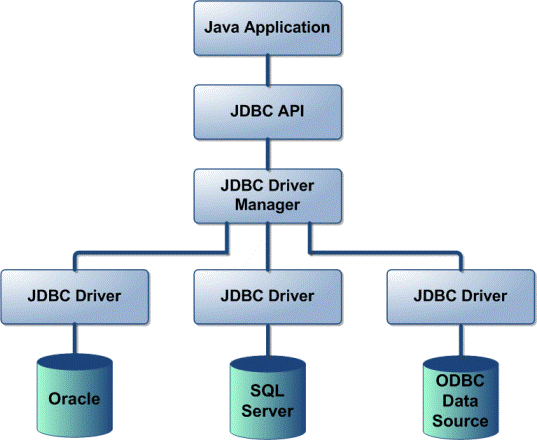
Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

# API

## Java Persistence API

Java Persistence API (JPA) is a Java programming language application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

## JDBC



JDBC (Java Database Connectivity) is a Java-based data access technology (Java Standard Edition platform) which is an API for the Java programming language that defines how a client may access a database. JDBC is oriented towards relational databases.

# Entity classes

An entity is a lightweight persistence domain object. Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The primary programming artifact of an entity is the entity class, although entities can use helper classes.

Entity (POJO) classes: Client, Product, Order, OrderedProduct, Photos

@Entity

@Table(name = "Client")

public class Client {

@Id

@GeneratedValue

private int id;

private String name;

private String surname;

private String login;

private String password;

public Client() {

}

public Client(String name, String surname, String login, String password) {

super();

this.id = 0;

this.name = name;

this.surname = surname;

this.login = login;

this.password = password;

}

public Client(int id, String name, String surname, String login,

String password) {

super();

this.id = id;

this.name = name;

this.surname = surname;

this.login = login;

this.password = password;

}

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 String getSurname() {

return surname;

}

public void setSurname(String surname) {

this.surname = surname;

}

public String getLogin() {

return login;

}

public void setLogin(String login) {

this.login = login;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + id;

result = prime \* result + ((login == null) ? 0 : login.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

result = prime \* result

+ ((password == null) ? 0 : password.hashCode());

result = prime \* result + ((surname == null) ? 0 : surname.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Client other = (Client) obj;

if (id != other.id)

return false;

if (login == null) {

if (other.login != null)

return false;

} else if (!login.equals(other.login))

return false;

if (name == null) {

if (other.name != null)

return false;

} else if (!name.equals(other.name))

return false;

if (password == null) {

if (other.password != null)

return false;

} else if (!password.equals(other.password))

return false;

if (surname == null) {

if (other.surname != null)

return false;

} else if (!surname.equals(other.surname))

return false;

return true;

}

@Override

public String toString() {

return "Client [id=" + id + ", name=" + name + ", surname=" + surname

+ ", login=" + login + ", description=" + password + "]";

}

}

@Entity

@Table(name = "Orders")

public class Order {

@Id

@GeneratedValue

private int id;

private int clientId;

private int status;

private Date date;

public Order() {

}

public Order(int id, int clientId, int status, Date date) {

super();

this.id = id;

this.clientId = clientId;

this.status = status;

this.date = date;

}

public int getClientId() {

return clientId;

}

public void setClientId(int clientId) {

this.clientId = clientId;

}

public int getStatus() {

return status;

}

public void setStatus(int status) {

this.status = status;

}

public Date getDate() {

return date;

}

public void setDate(Date date) {

this.date = date;

}

@Override

public String toString() {

return "Product [clientId=" + clientId + ", status=" + status

+ ", date=" + date + " ]";

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + clientId;

result = prime \* result + ((date == null) ? 0 : date.hashCode());

result = prime \* result + id;

result = prime \* result + status;

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Order other = (Order) obj;

if (clientId != other.clientId)

return false;

if (date == null) {

if (other.date != null)

return false;

} else if (!date.equals(other.date))

return false;

if (id != other.id)

return false;

if (status != other.status)

return false;

return true;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

}

@Entity

@Table(name = "OrderedProducts")

public class OrderedProducts {

@Id

@GeneratedValue

private int id;

private String amaunt;

private String price;

private int orderID;

private int productID;

public OrderedProducts() {

}

public OrderedProducts(int id, String amaunt, String price, int orderID,

int productID) {

super();

this.id = id;

this.amaunt = amaunt;

this.price = price;

this.orderID = orderID;

this.productID = productID;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getAmaunt() {

return amaunt;

}

public void setAmaunt(String amaunt) {

this.amaunt = amaunt;

}

public String getPrice() {

return price;

}

public void setPRice(String price) {

this.price = price;

}

public int getOrderID() {

return orderID;

}

public void setOrderID(int orderID) {

this.orderID = orderID;

}

public void setProductID(int productID) {

this.productID = productID;

}

public int getProductID() {

return productID;

}

@Override

public String toString() {

return "OrderedProducts [id=" + id + ", amaunt=" + amaunt + ", price="

+ price + ", orderID=" + orderID + ", productID=" + productID

+ "]";

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((amaunt == null) ? 0 : amaunt.hashCode());

result = prime \* result + id;

result = prime \* result + orderID;

result = prime \* result + ((price == null) ? 0 : price.hashCode());

result = prime \* result + productID;

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

OrderedProducts other = (OrderedProducts) obj;

if (amaunt == null) {

if (other.amaunt != null)

return false;

} else if (!amaunt.equals(other.amaunt))

return false;

if (id != other.id)

return false;

if (orderID != other.orderID)

return false;

if (price == null) {

if (other.price != null)

return false;

} else if (!price.equals(other.price))

return false;

if (productID != other.productID)

return false;

return true;

}

}

@Entity

@Table(name = "Photos")

**public** **class** Photos {

@Id

@GeneratedValue

**private** **int** id;

**private** String fileName;

**private** **int** productId;

**public** Photos() {

}

**public** Photos(**int** id, String fileName, **int** productId) {

**super**();

**this**.id = id;

**this**.fileName = fileName;

**this**.productId = productId;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getFileName() {

**return** fileName;

}

**public** **void** setFileName(String fileName) {

**this**.fileName = fileName;

}

**public** **int** getProductId() {

**return** productId;

}

**public** **void** setProductId(**int** productId) {

**this**.productId = productId;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result

+ ((fileName == **null**) ? 0 : fileName.hashCode());

result = prime \* result + id;

result = prime \* result + productId;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Photos other = (Photos) obj;

**if** (fileName == **null**) {

**if** (other.fileName != **null**)

**return** **false**;

} **else** **if** (!fileName.equals(other.fileName))

**return** **false**;

**if** (id != other.id)

**return** **false**;

**if** (productId != other.productId)

**return** **false**;

**return** **true**;

}

@Override

**public** String toString() {

**return** "Photos [id=" + id + ", fileName=" + fileName + ", productId="

+ productId + "]";

}

}

@Entity

@Table(name = "Product")

**public** **class** Product {

@Id

@GeneratedValue

**private** **int** id;

**private** String name;

**private** **int** year;

**private** **float** price;

**private** String description;

**private** String producer;

@OneToMany(mappedBy = "productId", fetch = FetchType.*EAGER*)

**private** List<Photos> photos;

**public** Product() {

}

**public** Product(**int** id, String name, **int** year, **float** price,

String description, String producer) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.year = year;

**this**.price = price;

**this**.description = description;

**this**.producer = producer;

}

**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** **int** getYear() {

**return** year;

}

**public** **void** setYear(**int** year) {

**this**.year = year;

}

**public** **float** getPrice() {

**return** price;

}

**public** **void** setPrice(**float** price) {

**this**.price = price;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** String getProducer() {

**return** producer;

}

**public** **void** setProducer(String producer) {

**this**.producer = producer;

}

**public** List<Photos> getPhotos() {

**return** photos;

}

**public** **void** setPhotos(List<Photos> photos) {

**this**.photos = photos;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result

+ ((description == **null**) ? 0 : description.hashCode());

result = prime \* result + id;

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

result = prime \* result + Float.*floatToIntBits*(price);

result = prime \* result

+ ((producer == **null**) ? 0 : producer.hashCode());

result = prime \* result + year;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Product other = (Product) obj;

**if** (description == **null**) {

**if** (other.description != **null**)

**return** **false**;

} **else** **if** (!description.equals(other.description))

**return** **false**;

**if** (id != other.id)

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**if** (Float.*floatToIntBits*(price) != Float.*floatToIntBits*(other.price))

**return** **false**;

**if** (producer == **null**) {

**if** (other.producer != **null**)

**return** **false**;

} **else** **if** (!producer.equals(other.producer))

**return** **false**;

**if** (year != other.year)

**return** **false**;

**return** **true**;

}

@Override

**public** String toString() {

**return** "Product [id=" + id + ", name=" + name + ", year=" + year

+ ", price=" + price + ", description=" + description

+ ", producer=" + producer + "]";

}

}

The persistent state of an entity is represented either through persistent fields or persistent properties. These fields or properties use object/relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.

# Data Access Object classes

DAO classes:

1. Using JDBC:

2. Using Hibernate ORM:

*ClientRepository* class (DAO) is working with Client class responsible to get/set data from/to a datasource . Using this class we will perform simple operations: find a client by ID, find a client by client name, find a client by client surname, find all clients, create new client in database.

## JDBC approach

public class DatabaseConnector {

private String url;

private String userName;

private String password;

private String filePath = "./resources/dbConfig";

private String driverName = "com.mysql.jdbc.Driver";

private static DatabaseConnector instance = null;

public static DatabaseConnector getInstance() {

if (instance == null) {

instance = new DatabaseConnector();

}

return instance;

}

private DatabaseConnector() {

url = "jdbc:mysql://localhost:3306/shop";

userName = "root";

password = "abcd1234";

}

public Connection getConnection() throws SQLException {

try {

Class.forName(driverName);

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

return DriverManager.getConnection(url, userName, password);

}

public void closeConnection(Connection connection) throws SQLException {

connection.close();

}

}

**public** **class** ClientRepository {

/\*Finds a client by client ID\*/

**public** Client findID(Integer clientID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Client client = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Client where id = ?");

statement.setInt(1, clientID);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

client = **new** Client(resultSet.getInt("id"),

resultSet.getString("name"), resultSet.getString("surname"),

resultSet.getString("login"),

resultSet.getString("password"));

}

} **finally** {

connection.close();

}

**return** client;

}

/\*Finds a client by client name\*/

**public** Client findName(String clientName) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Client client = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Client where name = ?");

ResultSet resultSet = statement.executeQuery();

statement.setString(1, clientName);

**if** (resultSet.next()) {

client = **new** Client(resultSet.getInt("id"),

resultSet.getString("name"), resultSet.getString("surname"),

resultSet.getString("login"),

resultSet.getString("password"));

}

} **finally** {

connection.close();

}

**return** client;

}

/\* Finds a client by client surname\*/

**public** Client findSurname(String clientSurname) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Client client = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Client where surname = ?");

ResultSet resultSet = statement.executeQuery();

statement.setString(1, clientSurname);

**if** (resultSet.next()) {

client = **new** Client(resultSet.getInt("id"),

resultSet.getString("name"), resultSet.getString("surname"),

resultSet.getString("login"),

resultSet.getString("password"));

}

} **finally** {

connection.close();

}

**return** client;

}

/\*Finds all clients\*/

**public** List<Client> findAll() **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

List<Client> clients = **new** ArrayList<Client>();

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Client");

ResultSet resultSet = statement.executeQuery();

**while** (resultSet.next()) {

clients.add(**new** Client(resultSet.getInt("id"), resultSet

.getString("name"), resultSet.getString("surname"), resultSet

.getString("login"), resultSet.getString("password")));

}

} **finally** {

connection.close();

}

**return** clients;

}

/\*Creates new client in Data Base\*/

**public** **void** create(Client client) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("INSERT INTO Contact VALUES (default,?,?,?,?,?);");

statement.setString(1, client.getName());

statement.setString(2, client.getSurname());

statement.setString(3, client.getLogin());

statement.setString(4, client.getPassword());

statement.executeUpdate();

} **finally** {

connection.close();

}

}

/\*Deletes client from Data base\*/

**public** **void** delete(Long clientID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("DELETE Client WHERE id = ? ");

statement.setLong(1, clientID);

statement.executeUpdate();

} **finally** {

connection.close();

}

}

}

**public** **class** OrderedProductsRepository {

/\*Finds ordered product by Ordered ID\*/

**public** OrderedProducts findID(Integer OrderedID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

OrderedProducts ordProd = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from OrderedProducts where id = ?");

statement.setInt(1, OrderedID);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

ordProd = **new** OrderedProducts(resultSet.getInt("id"),

resultSet.getString("amaunt"),

resultSet.getString("price"),

resultSet.getInt("orderID"),

resultSet.getInt("productID"));

}

} **finally** {

connection.close();

}

**return** ordProd;

}

/\*Finds ordered product by amount\*/

**public** OrderedProducts findAmaunt(String Amaunt) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

OrderedProducts ordProd = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from OrderedProducts where amaut = ?");

statement.setString(1, Amaunt);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

ordProd = **new** OrderedProducts(resultSet.getInt("id"),

resultSet.getString("amaunt"),

resultSet.getString("price"),

resultSet.getInt("orderID"),

resultSet.getInt("productID"));

}

} **finally** {

connection.close();

}

**return** ordProd;

}

/\*Finds ordered product by price\*/

**public** OrderedProducts findPrice(String Price) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

OrderedProducts ordProd = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from OrderedProducts where price = ?");

statement.setString(1, Price);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

ordProd = **new** OrderedProducts(resultSet.getInt("id"),

resultSet.getString("amaunt"),

resultSet.getString("price"),

resultSet.getInt("orderID"),

resultSet.getInt("productID"));

}

} **finally** {

connection.close();

}

**return** ordProd;

}

/\*Finds ordered products by Order ID\*/

**public** OrderedProducts findOrderID(Integer OrderID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

OrderedProducts ordProd = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from OrderedProducts where orderId = ?");

statement.setInt(1, OrderID);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

ordProd = **new** OrderedProducts(resultSet.getInt("id"),

resultSet.getString("amaunt"),

resultSet.getString("price"),

resultSet.getInt("orderID"),

resultSet.getInt("productID"));

}

} **finally** {

connection.close();

}

**return** ordProd;

}

/\*Finds a ordered product by product id\*/

**public** OrderedProducts findProductID(Integer ProductID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

OrderedProducts ordProd = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from OrderedProducts where productId = ?");

statement.setInt(1, ProductID);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

ordProd = **new** OrderedProducts(resultSet.getInt("id"),

resultSet.getString("amaunt"),

resultSet.getString("price"),

resultSet.getInt("orderID"),

resultSet.getInt("productID"));

}

} **finally** {

connection.close();

}

**return** ordProd;

}

/\*Finds all ordered products\*/

**public** List<OrderedProducts> findAll() **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

List<OrderedProducts> ordered = **new** ArrayList<OrderedProducts>();

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from OrderedProducts");

ResultSet resultSet = statement.executeQuery();

**while** (resultSet.next()) {

ordered.add(**new** OrderedProducts(resultSet.getInt("id"),

resultSet.getString("amaunt"),

resultSet.getString("price"),

resultSet.getInt("orderID"),

resultSet.getInt("productID")));

}

} **finally** {

connection.close();

}

**return** ordered;

}

/\*Creates new ordered product in Data Base\*/

**public** **void** create(OrderedProducts ordProd) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("INSERT INTO OrderedProducts VALUES (default,?,?,?,?);");

statement.setString(1, ordProd.getAmaunt());

statement.setString(2, ordProd.getPrice());

statement.setInt(3, ordProd.getOrderID());

statement.setInt(4, ordProd.getProductID());

statement.executeUpdate();

} **finally** {

connection.close();

}

}

/\*Deletes ordered product from Data Base\*/

**public** **void** delete(Integer productID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("DELETE OrderedProducts WHERE id = ? ");

statement.setInt(1, productID);

statement.executeUpdate();

} **finally** {

connection.close();

}

}

}

**public** **class** OrderRepository {

/\*Finds order by orders id\*/

**public** Order findID(**int** orderID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Order order = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from `Order` where id = ?");

statement.setInt(1, orderID);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

order = **new** Order(resultSet.getInt("id"),

resultSet.getInt("clientId"),

resultSet.getInt("status"), resultSet.getDate("date"));

}

} **finally** {

connection.close();

}

**return** order;

}

/\*Finds order by client id\*/

**public** Order findClientID(**int** clientId) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Order order = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from `Order` where clientId = ?");

statement.setInt(1, clientId);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

order = **new** Order(resultSet.getInt("id"),

resultSet.getInt("clientId"),

resultSet.getInt("status"), resultSet.getDate("date"));

}

} **finally** {

connection.close();

}

**return** order;

}

/\*Finds orders date\*/

**public** Order findDate(Date date) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Order order = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from `Order` where date = ?");

statement.setDate(1, (java.sql.Date) date);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

order = **new** Order(resultSet.getInt("id"),

resultSet.getInt("clientId"),

resultSet.getInt("status"), resultSet.getDate("date"));

}

} **finally** {

connection.close();

}

**return** order;

}

/\*Finds all orders\*/

**public** List<Order> findAll() **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

List<Order> orders = **new** ArrayList<Order>();

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from `Order`");

ResultSet resultSet = statement.executeQuery();

**while** (resultSet.next()) {

orders.add(**new** Order(resultSet.getInt("id"), resultSet

.getInt("clientId"), resultSet.getInt("status"),

resultSet.getDate("date")));

}

} **finally** {

connection.close();

}

**return** orders;

}

/\*Creates new order\*/

**public** **void** create(Order order) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("INSERT INTO `Order` VALUES (default,?,?,?);");

statement.setInt(1, order.getClientId());

statement.setInt(2, order.getStatus());

statement.setDate(3, order.getDate());

;

statement.executeUpdate();

} **finally** {

connection.close();

}

}

/\*Deletes order from Data Base\*/

**public** **void** delete(**int** orderID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("DELETE FROM `Order` WHERE id = ? ");

statement.setInt(1, orderID);

statement.executeUpdate();

} **finally** {

connection.close();

}

}

}

**public** **class** PhotosRepository {

/\*Finds photo by photo id\*/

**public** Photos findID(Integer photoID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Photos photo = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Photos where id = ?");

statement.setInt(1, photoID);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

photo = **new** Photos(resultSet.getInt("id"),

resultSet.getString("fileName"),

resultSet.getInt("productId"));

}

} **finally** {

connection.close();

}

**return** photo;

}

/\*Finds photo by product id\*/

**public** Photos findProduct(Integer productID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Photos photo = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Photos where productId = ?");

statement.setInt(1, productID);

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

photo = **new** Photos(resultSet.getInt("id"),

resultSet.getString("fileName"),

resultSet.getInt("productId"));

}

} **finally** {

connection.close();

}

**return** photo;

}

/\*Finds all photos\*/

**public** List<Photos> findAll() **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

List<Photos> photos = **new** ArrayList<Photos>();

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Photos");

ResultSet resultSet = statement.executeQuery();

**while** (resultSet.next()) {

photos.add(**new** Photos(resultSet.getInt("id"), resultSet

.getString("fileName"), resultSet.getInt("productId")));

}

} **finally** {

connection.close();

}

**return** photos;

}

/\*Creates new photo\*/

**public** **void** create(Photos photo) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("INSERT INTO Photos VALUES (default,?,?);");

statement.setString(1, photo.getFileName());

statement.setInt(2, photo.getProductId());

statement.executeUpdate();

} **finally** {

connection.close();

}

}

/\*Deletes photo from data base\*/

**public** **void** delete(Integer photoID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("DELETE Photos WHERE id = ? ");

statement.setInt(1, photoID);

statement.executeUpdate();

} **finally** {

connection.close();

}

}

**public** **class** ProductRepository {

/\*Finds product by contact ID\*/

**public** Product find(Long contactID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

Product product = **null**;

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Product where id = ?");

ResultSet resultSet = statement.executeQuery();

**if** (resultSet.next()) {

product = **new** Product(resultSet.getInt("id"),

resultSet.getString("name"), resultSet.getInt("year"),

resultSet.getFloat("price"),

resultSet.getString("description"),

resultSet.getString("producer"));

}

} **finally** {

connection.close();

}

**return** product;

}

/\*Finds all products\*/

**public** List<Product> findAll() **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

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

**try** {

PreparedStatement statement = connection

.prepareStatement("select \* from Product");

ResultSet resultSet = statement.executeQuery();

**while** (resultSet.next()) {

products.add(**new** Product(resultSet.getInt("id"), resultSet

.getString("name"), resultSet.getInt("year"), resultSet

.getFloat("price"), resultSet.getString("description"),

resultSet.getString("producer")));

}

} **finally** {

connection.close();

}

**return** products;

}

/\*Creates new product\*/

**public** **void** create(Product product) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("INSERT INTO Contact VALUES (default,?,?,?,?,?);");

statement.setString(1, product.getName());

statement.setLong(2, product.getYear());

statement.setFloat(3, product.getPrice());

statement.setString(4, product.getDescription());

statement.setString(5, product.getProducer());

statement.executeUpdate();

} **finally** {

connection.close();

}

}

/\*Deletes new product\*/

**public** **void** delete(Long productID) **throws** SQLException {

Connection connection = DatabaseConnector.*getInstance*().getConnection();

**try** {

PreparedStatement statement = connection

.prepareStatement("DELETE Product WHERE id = ? ");

statement.setLong(1, productID);

statement.executeUpdate();

} **finally** {

connection.close();

}

}

}

## Hibernate approach

@Component

public class OrmDaoImpl {

@Autowired

@Qualifier("sessionFactory")

private SessionFactory sessionFactory;

protected Session getCurrentSession() {

return sessionFactory.getCurrentSession();

}

}

|  |
| --- |
| @Repository  **public** **class** ClientRepository **extends** OrmDaoImpl {  /\* Creating criteria. getCurrentSession() obtains the current session \*/  @Transactional  **public** List<Client> find() **throws** HibernateException {  **return** getCurrentSession().createCriteria(Client.**class**).list();  }  /\* Finds client in database by id \*/  @Transactional  **public** Client findById(Integer id) **throws** HibernateException {  **return** (Client) getCurrentSession().get(Client.**class**, id);  }  /\*Criteria object is used to define how we will take data from database. In this case it is used to find client by name. Returns list of objects that have name in property "name"\*/  @Transactional  **public** List<Client> findByName(String name) **throws** HibernateException {  Criteria criteria = getCurrentSession().createCriteria(Client.**class**);  **return** criteria.add(Restrictions.*eq*("name", name)).list();  }  /\*In this case criteria is used to find client by username. Returns list of objects that have username in property "login" \*/  @Transactional  **public** List<Client> findByUsername(String username)  **throws** HibernateException {  Criteria criteria = getCurrentSession().createCriteria(Client.**class**);  **return** criteria.add(Restrictions.*eq*("login", username)).list();  }  /\*Method for saving and updating client in database\*/  @Transactional  **public** Client create(Client client) {  getCurrentSession().saveOrUpdate(client);  **return** client;  }  } |

@Component

**public** **class** OrderedProductsRepository **extends** OrmDaoImpl {

/\* Creating criteria. getCurrentSession() obtains the current session \*/

@Transactional

**public** List<OrderedProducts> find() **throws** HibernateException {

**return** getCurrentSession().createCriteria(OrderedProducts.**class**).list();

}

/\* Finds ordered product in database by id \*/

@Transactional

**public** OrderedProducts findById(Integer id) **throws** HibernateException {

**return** (OrderedProducts) getCurrentSession().get(OrderedProducts.**class**,

id);

}

/\*Criteria object is used to define how we will take data from database. In this case it is used to find ordered product by amount. Returns list of objects that have amount in property "amount"\*/

@Transactional

**public** List<OrderedProducts> findByName(Integer amaunt)

**throws** HibernateException {

Criteria criteria = getCurrentSession().createCriteria(

OrderedProducts.**class**);

**return** criteria.add(Restrictions.*eq*("amount", amount)).list();

}

/\*Method for saving and updating ordered products in database\*/

@Transactional

**public** OrderedProducts create(OrderedProducts orderedProducts) {

getCurrentSession().saveOrUpdate(orderedProducts);

**return** orderedProducts;

}

}

@Component

**public** **class** OrderRepository **extends** OrmDaoImpl {

/\* Creating criteria. getCurrentSession() obtains the current session.\*/

@Transactional

**public** List<Order> find() **throws** HibernateException {

**return** getCurrentSession().createCriteria(Order.**class**).list();

}

/\* Finds order in database by id\*/

@Transactional

**public** Order findById(Integer id) **throws** HibernateException {

**return** (Order) getCurrentSession().get(Order.**class**, id);

}

/\*Criteria object is used to define how we will take data from database. In this case it is used to find order by client id. Returns list of objects that have clientId in property "clientId"\*/

@Transactional

**public** List<Order> findByName(Integer clientId) **throws** HibernateException {

Criteria criteria = getCurrentSession().createCriteria(Order.**class**);

**return** criteria.add(Restrictions.*eq*("clientId", clientId)).list();

}

/\*Method for saving and updating order in database\*/

@Transactional

**public** Order create(Order order) {

getCurrentSession().saveOrUpdate(order);

**return** order;

}

}

@Component

**public** **class** PhotosRepository **extends** OrmDaoImpl {

/\* Creating criteria. getCurrentSession() obtains the current session.\*/

@Transactional

**public** List<Photos> find()**throws** HibernateException {

**return** getCurrentSession().createCriteria(Photos.**class**).list();

}

/\* Finds photo in database by id\*/

@Transactional

**public** Photos findById(Integer id)**throws** HibernateException {

**return** (Photos) getCurrentSession().get(Photos.**class**, id);

}

/\*Criteria object is used to define how we will take data from database. In this case it is used to find photo by product id. Returns list of objects that have productId in property "productId"\*/

@Transactional

**public** List<Photos> findByName(Integer productId)**throws** HibernateException {

Criteria criteria = getCurrentSession().createCriteria(Photos.**class**);

**return** criteria.add(Restrictions.*eq*("productId", productId)).list();

}

}

@Repository

**public** **class** ProductRepository **extends** OrmDaoImpl {

/\* Creating criteria. getCurrentSession() obtains the current session. setResultTransformer(Criteria.DISTINCT\_ROOT\_ENTITY) will make the results distinct\*/

@Transactional

**public** List<Product> find() **throws** HibernateException {

**return** getCurrentSession().createCriteria(Product.**class**).setResultTransformer(Criteria.*DISTINCT\_ROOT\_ENTITY*).list();

}

/\* Finds product in database by id\*/

@Transactional

**public** Product findById(Integer id) **throws** HibernateException {

**return** (Product) getCurrentSession().get(Product.**class**, id);

}

/\*Criteria object is used to define how we will take data from database. In this case it is used to find product by producer Model. Returns list of objects that have producerModel in property "name"\*/

@Transactional

**public** List<Product> findCarsByModel(String producerModel) **throws** HibernateException {

Criteria criteria = getCurrentSession().createCriteria(Product.**class**);

**return** criteria.add(Restrictions.*eq*("name", producerModel)).setResultTransformer(Criteria.*DISTINCT\_ROOT\_ENTITY*).list();

}

/\*Criteria object is used to define how we will take data from database. In this case it is used to find product by producer. Returns list of objects that have producer in property "producer"\*/

@Transactional

**public** List<Product> findByProducer(String producer)

**throws** HibernateException {

Criteria criteria = getCurrentSession().createCriteria(Product.**class**);

**return** criteria.add(Restrictions.*eq*("producer", producer)).setResultTransformer(Criteria.*DISTINCT\_ROOT\_ENTITY*).list();

}

@Transactional

**public** List<String> findModelsByProducer(String producer)

**throws** HibernateException {

Criteria criteria = getCurrentSession().createCriteria(Product.**class**);

List<Product> a = **new** ArrayList<Product>();

List<String> b = **new** ArrayList<String>();

a = criteria.add(Restrictions.*eq*("producer", producer)).list();

**if** (criteria.list().size() > 0)

**for** (**int** i = 0; i < criteria.list().size(); i++) {

**if** (!b.contains(a.get(i).getName())) {

b.add(a.get(i).getName());

}

}

**return** b;

}

/\*Method for saving and updating product in database\*/

@Transactional

**public** Product create(Product product) {

getCurrentSession().saveOrUpdate(product);

**return** product;

}

}

# Controllers

**Controller classes:** MainContraller.java, ProducerContraller.java, ProductController.java, ClientContraller.java, OrderContraller.java.

@Controller

public class **MainContraller** {

**/\*\*When main page is requested with “/” or “/main”, then a list with producers names is created from all products. main.jsp is opened. \*\*/**

@Autowired

ProductRepository repo;

@RequestMapping("/")

public String mainPage0(HttpServletResponse response,

HttpServletRequest request) {

List<String> producers = new ArrayList<>();

for (Product product : repo.find()) {

if (!producers.contains(product.getProducer())) {

producers.add(product.getProducer());

}

}

request.setAttribute("producers", producers);

return "main";

}

@RequestMapping("/main")

public String mainPage(HttpServletResponse response,

HttpServletRequest request) {

List<String> producers = new ArrayList<>();

for (Product product : repo.find()) {

if (!producers.contains(product.getProducer())) {

producers.add(product.getProducer());

}

}

request.setAttribute("producers", producers);

return "main";

}

}

@Controller

public class **ProducerContraller** {

@Autowired

ProductRepository repo;

**/\*\*When a specific producer page is requested with “/producer/{producer}”, then a list with specific producer models names is created from all products. models.jsp is opened.\*\*/**

@RequestMapping(value = "/producer/{producer}", method = RequestMethod.GET)

public String getMyData1(@PathVariable String producer,

HttpServletResponse response, HttpServletRequest request) {

request.setAttribute("item", repo.findModelsByProducer(producer));

return "models";

}

**/\*\*When a specific producer model page is requested with "/producer/model/{producerModel}", then a list with specific producer model cars is created from all products. products.jsp is opened.\*\*/**

@RequestMapping(value = "/producer/model/{producerModel}", method = RequestMethod.GET)

public String getMyData2(@PathVariable String producerModel,

HttpServletResponse response, HttpServletRequest request) {

request.setAttribute("item", repo.findCarsByModel(producerModel));

return "products";

}

}

@Controller

public class **ProductController** {

@Autowired

ProductRepository repoProduct;

**/\*\*When a specific producer model page is requested with "/products?name=", then a list with specific producer model cars is created from all products. products.jsp is opened. \*\*/**

@RequestMapping("/products")

public String allProducts(

@RequestParam(value = "name", required = false, defaultValue = "World") String name,

HttpServletResponse response, HttpServletRequest request) {

List<Product> products = repoProduct.find();

request.setAttribute("item", products.toString());

return "products";

}

**/\*\*When a specific product page is requested with "/products/{productId}", then a specific product is searched from all products. Product is saved in session. product.jsp is opened. \*\*/**

@RequestMapping(value = "/products/{productId}", method = RequestMethod.GET)

public String getProduct(@PathVariable Integer productId,

HttpServletResponse response, HttpServletRequest request) {

Product product = repoProduct.findById(productId);

request.getSession().setAttribute("item", product);

return "product";

}

}

@Controller

@SessionAttributes("item")

public class **ClientContraller** {

@Autowired

ClientRepository repoClient;

@Autowired

ProductRepository repoProduct;

**/\*\*When a client registration page is requested with "/client ", then a client registration page (registration.jsp) is opened.\*\*/**

@RequestMapping(value = "/client", method = RequestMethod.GET)

public String client(HttpServletResponse response, HttpServletRequest request) {

return "registration";

}

**/\*\*When a client finished registration page is requested with "/client/add", then a new client account is created. Client is saved in session. registration.jsp is opened. \*\*/**

@RequestMapping(value = "/client/add", method = RequestMethod.POST)

public String addClient(

@RequestParam(value = "name", required = false) String name,

@RequestParam(value = "surname", required = false) String surname,

@RequestParam(value = "login", required = false) String login,

@RequestParam(value = "password", required = false) String password,

HttpServletResponse response, HttpServletRequest request) {

Client client = new Client(name, surname, login, password);

client = repoClient.create(client);

request.setAttribute("message", "Client wass added to database");

request.getSession().setAttribute("client", client);

return "registration";

}

}

@Controller

public class **OrderContraller** {

@Autowired

OrderRepository repoOrder;

@Autowired

OrderedProductsRepository repoOrderedproducts;

@Autowired

ProductRepository repoProduct;

**/\*\*When a specific order page with a product ID is requested with "/order/{productId}”, then order.jsp is opened. \*\*/**

@RequestMapping(value = "/order/{productId}", method = RequestMethod.GET)

public String getOrderNID(HttpServletResponse response, HttpServletRequest request) {

return "order";

}

**/\*\*When a finished order page is requested with "/ordercompleted", then a new order and ordered product information are created. Client and product information is taken from session. ordercompleted.jsp is opened. \*\*/**

@RequestMapping(value = "/ordercompleted", method = RequestMethod.GET)

public String getMyData1(HttpServletResponse response,

HttpServletRequest request) {

Product product = (Product) request.getSession().getAttribute("item");

Client client = (Client) request.getSession().getAttribute("client");

java.util.Date now = new java.util.Date();

java.sql.Date date = new java.sql.Date(now.getTime());

Order order = new Order(0, client.getId(), 0, date);

order = repoOrder.create(order);

OrderedProducts orderedProducts = new OrderedProducts(0, "1",

String.valueOf(product.getPrice()), order.getId(),

product.getId());

repoOrderedproducts.create(orderedProducts);

return "ordercompleted";

}

}

# JavaServer Pages

**JSP pages:** index, main, models, products, product, registration, order, ordercompleted, about, feedback, stores, support.

All pages, except “index” have registered user control (checking if it is a guest or registered user).

<div class='head-login'>

<%

Client client = (Client) request.getSession()

.getAttribute("client");

%>

<%

if (client == null) {

%>

Welcome,<a href='<%=request.getContextPath()%>/main'><b>

Guest</b></a>

<%

} else {

%>

<a href=''> Welcome, <b><%=client.getName()%> <%=client.getSurname()%></b></a>

<%

}

%>

</div>

“Index” page forward to “main” page:

<jsp:forward page="/pages/main.jsp" />

All pages, except “index”, have links to “feedback”, “support”, “stores”, “about” pages.

<div id="bottomBlocks">

<div class="block">

<a title="Feedback"

href="<%=request.getContextPath()%>/pages/feedback.jsp">Feedback</a>

</div>

<div class="block">

<a title="Support"

href="<%=request.getContextPath()%>/pages/support.jsp">Support

</a>

</div>

<div class="block">

<a title="Stores" href="<%=request.getContextPath()%>/pages/stores.jsp">Stores</a>

</div>

<div class="block">

<a title="About NetShop company"

href="<%=request.getContextPath()%>/pages/about.jsp">About NetShop company</a>

</div>

</div>

All pages, except “index”, have a logo with link to main page.

<a href="<%=request.getContextPath()%>/main" title="NETSHOP" class="logo"><img

src="<%=request.getContextPath()%>/resources/images/NetShop\_logo.jpg"

width="158" height="35" alt="NETSHOP" /></a>

All pages, except “index”, have a copyright.

<div id="copyright">&copy; 2014 NETSHOP. All Rights Reserved.</div>

“Main”, “products” pages has a photo banner.

<div style='background: url("<%=request.getContextPath()%>/resources/images/Mainbanner.jpg") ; height:356px;'> </div>

“Main” page has a table with manufactures brands and brands logo with links to models pages.

<table id="fixedheight" border="1" width="100%">

<%List<String> producers = (List<String>)request.getAttribute("producers"); %>

<%int i = 0; %>

<% while (i< producers.size()){ %>

<tr>

<% do{ %>

<% if (i < producers.size()){ %>

<th><a href="/producer/<%=producers.get(i)%>"> <img

src="<%=request.getContextPath()%>/resources/images/<%=producers.get(i) + ".jpg"%>" />

<h3><%= producers.get(i) %></h3>

</a></th>

<% } i++; %>

<% }while (i%4!=0); %>

</tr>

<%} %>

</table>

“Models” page has models names of specific manufacture brand with links to specific model cars.

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 300px; top: 50px; left: 0px;">

<div class="grid-cell-new"

style="width: 976px; height: 300px; vertical-align: middle;">

<%List<String> models = (List<String>)request.getAttribute("item"); %>

<% for (String model : models ){ %>

<p class="banner\_title-new"><a href="/producer/model/<%=model%>"> <%= model%> </a></p>

<% } %>

</div>

</div>

“Products” page has a table of specific model cars (name, year, price, description) with links to specific car in name and description.

<table class="attributes" id="fixedheight" border="1" rules="all">

<tr>

<th class="name" bgcolor="#CCCCCC" height="30">Name</th>

<th class="year" bgcolor="#CCCCCC" height="30">Year</th>

<th class="price" bgcolor="#CCCCCC" height="30">Price</th>

<th class="description" bgcolor="#CCCCCC" height="30">Description</th>

</tr>

<%List<Product> producers = (List<Product>)request.getAttribute("item"); %>

<% for (Product product : producers ){ %>

<tr>

<td class="attribute" height="30">

<a href="/products/<%=product.getId()%>"> <%=product.getName()%></a>

</td>

<td class="attribute" height="30"><%=product.getYear()%></td>

<td class="attribute" height="30"><%=NumberFormat.getCurrencyInstance(new Locale("en", "US")).format(product.getPrice())%></td>

<td height="30"><a href="/products/<%=product.getId()%>">

<%=product.getDescription().length() > 112 ? product.getDescription().substring(0,112) : product.getDescription() + "..."%></a></td>

</tr>

<% } %>

</table>

“Product” page has specific car description. Model name in title and description, manufacture brand name, year, price, description in description. Also photo slider. 2 buttons: “back” (return back using history) and “buy” (forward to client registration).

<%

Product producer = (Product) request.getSession().getAttribute("item");

%>

<div class="grid-wrapper-new">

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 50px; top: 0px; left: 0px;">

<div class="grid-cell-new" style="width: 976px; height: 50px;"><hr>

<p class="banner\_title-new"><%=producer.getName()%></p><br><hr>

</div>

</div>

<div class="grid-cell-wrapper-new"

style="width: 488px; height: 400px; top: 50px; left: 0px;">

<div class="grid-cell-new"

style="width: 488px; height: 400px; vertical-align: middle;">

<br></br>

<div class="slider">

<% for (Photos photo : producer.getPhotos()) { %>

<img src="/resources/photos/<%= photo.getFileName() %>" />

<% } %>

</div>

</div>

</div>

<div class="grid-cell-wrapper-new"

style="width: 488px; height: 400px; top: 50px; left: 488px;">

<div class="grid-cell-new" style="width: 488px; height: 400px;"><br>

<p class="banner\_title-new"><b>Description</b></p>

<p class="banner\_attribute">

Name: <span class="banner\_description-new"><%=producer.getName()%></span>

</p>

<p class="banner\_attribute">

Year: <span class="banner\_description-new"><%=producer.getYear()%></span>

</p>

<p class="banner\_attribute">

Price: <span class="banner\_description-new"><%=NumberFormat.getCurrencyInstance(new Locale("en", "US")).format(producer.getPrice())%></span>

</p>

<p class="banner\_attribute">

Description: <span class="banner\_description-new"><%=producer.getDescription()%></span>

</p>

<p class="banner\_attribute">

Producer: <span class="banner\_description-new"><%=producer.getProducer()%></span>

</p>

</div>

</div>

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 50px; top: 450px; left: 0px;">

<div class="grid-cell-new" style="width: 976px; height: 120px;"><br>

<p align="center">

<button type="button" name="buttonBack"

style="font-size: small; width: 74px; height: 35px;"

onclick="history.go(-1);return true;"><b>Back</b></button>

<button type="button" name="buttonBack" autofocus

style="font-size: small; width: 74px; height: 35px;"

onclick="location.href='http://localhost:8081/client'"><b>Buy</b></button>

</p><br><hr>

</div>

</div>

</div>

“Registration” page has a registration form with textboxes (name, surname, login, password). Validates if all fields are filled: if yes, redirect to order page, if no, reload registration page. 2 buttons: “back” (return back using history) and “sign up” (forward to order page using redirect after validation with record in database about client).

<script>

function validateForm() {

var x = document.forms["myForm"]["name"].value;

if (x==null || x=="") {

alert("All fields must be filled out");

setTimeout(function() {

javascript: location.reload(true)

}, 2000)

return false;

}

var x1 = document.forms["myForm"]["surname"].value;

if (x1==null || x1=="") {

alert("All fields must be filled out");

setTimeout(function() {

javascript: location.reload(true)

}, 2000)

return false;

}

var x2 = document.forms["myForm"]["login"].value;

if (x2==null || x2=="") {

alert("All fields must be filled out");

setTimeout(function() {

javascript: location.reload(true)

}, 2000)

return false;

}

var x3 = document.forms["myForm"]["password"].value;

if (x3==null || x3=="") {

alert("All fields must be filled out");

setTimeout(function() {

javascript: location.reload(true)

}, 2000)

return false;

}

}

</script>

<div class="grid-wrapper-new">

<form name="myForm" action="/client/add"

onsubmit="return validateForm();" method="post">

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 50px; top: 450px; left: 0px;">

<div class="grid-cell-new" style="width: 976px; height: 50px;">

</div>

</div>

<br>

<center>

<%

if (request.getAttribute("message") != null) {

%>

<%=request.getAttribute("message")%>

<%

Product producer = (Product) request.getSession().getAttribute(

"item");

%>

<script type="text/javascript">

setTimeout(function () { window.location.href = '/order/<%=producer.getId()%>';}, 2000);

</script>

<%

}

%>

</center>

<br> <br> <br> <br>

<div style="text-align: center" class="input-box">

<label for="name" class="myboxlabel"> Name </label> <input

type="text" name="name" class="mytext" /><br> <br>

</div>

<div style="text-align: center" class="input-box">

<label for="surname" class="myboxlabel"> Surname </label> <input

type="text" name="surname" class="mytext" /><br> <br>

</div>

<div style="text-align: center" class="input-box">

<label for="login" class="myboxlabel">Login</label> <input

type="text" name="login" class="mytext" /><br> <br>

</div>

<div style="text-align: center" class="input-box">

<label for="password" class="myboxlabel">Password</label> <input

type="password" name="password" class="mytext" /><br> <br>

</div>

<p align="center">

<button type="button" name="buttonBack"

style="font-size: small; width: 64px; height: 25px;"

onclick="history.go(-1);return true;">Back</button>

<button type="submit" name="buttonSignUp" autofocus

style="font-size: small; width: 64px; height: 25px;">Sign

up</button>

</p>

</form>

</div>

“Order” page is similar to “product” page with few differences. “Order” page has specific car description. Model name in title and description, manufacture brand name, year, price, description in description. Also photo slider. 1 button: “confirm & buy” (forward to ordered completed page with records in database about order).

<%

Product producer = (Product) request.getSession().getAttribute("item");

%>

<div class="grid-wrapper-new">

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 50px; top: 0px; left: 0px;">

<div class="grid-cell-new" style="width: 976px; height: 50px;">

<p class="banner\_title-new"><%=producer.getName()%></p>

<br>

<hr>

</div>

</div>

<div class="grid-cell-wrapper-new"

style="width: 488px; height: 400px; top: 50px; left: 0px;">

<div class="grid-cell-new"

style="width: 488px; height: 400px; vertical-align: middle;">

<br></br>

<div class="slider">

<%

for (Photos photo : producer.getPhotos()) {

%>

<img src="/resources/photos/<%=photo.getFileName()%>" />

<%

}

%>

</div>

</div>

</div>

<div class="grid-cell-wrapper-new"

style="width: 488px; height: 400px; top: 50px; left: 488px;">

<div class="grid-cell-new" style="width: 488px; height: 400px;">

<br>

<p class="banner\_title-new">

<b>Description</b>

</p>

<p class="banner\_attribute">

Name: <span class="banner\_description-new"><%=producer.getName()%></span>

</p>

<p class="banner\_attribute">

Year: <span class="banner\_description-new"><%=producer.getYear()%></span>

</p>

<p class="banner\_attribute">

Price: <span class="banner\_description-new"><%=producer.getPrice()%></span>

</p>

<p class="banner\_attribute">

Description: <span class="banner\_description-new"><%=producer.getDescription()%></span>

</p>

<p class="banner\_attribute">

Producer: <span class="banner\_description-new"><%=producer.getProducer()%></span>

</p>

</div>

</div>

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 50px; top: 450px; left: 0px;">

<div class="grid-cell-new" style="width: 976px; height: 120px;">

<br> <p align="center"><a href="/ordercompleted">

<button type="button" name="buttonConfirmAndBuy" autofocus

style="font-size: x-large; width: 200px; height: 35px;">Confirm

& Buy</button>

</a></p>

<br>

<br>

<br>

</div>

</div>

</div>

“Order completed” page has model name in title, image and phrase about order status. 1 button: “return to main page” (forward to main page).

<%

Product producer = (Product) request.getSession().getAttribute("item");

%>

<div class="grid-wrapper-new">

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 50px; top: 0px; left: 0px;">

<div class="grid-cell-new" style="width: 976px; height: 50px;">

<p class="banner\_title-new">You have purchased <%=producer.getName()%>!</p><br>

</div>

</div>

<div class="grid-cell-wrapper-new"

style="width: 976px; height: 400px; top: 50px; left: 0px;">

<div class="grid-cell-new" style="width: 976px; height: 400px;">

<center><img src="<%=request.getContextPath()%>/resources/images/correct-12.jpg"/></center>

<center><h3>Thank you for your purchase! Your order will be completed in 24h.</h3></center><br>

<a href="/main">

<p align="center">

<button type="button" name="buttonConfirmAndBuy" autofocus

style="font-size: x-large; width: 300px; height: 50px;">Return to main page</button>

</p>

</a>

</div>

</div>

</div>

“Feedback” page has textboxes (first name, comment) and button “submit”.

<form>

<h2>First name: <input style="margin-left: 33px;" type="text" name="FirstName" value=" "><br></h2><br>

<h2>Comment: <input style="margin-left: 41px;" type="text" name="Comment" value=" "><br></h2><br>

<input type="submit" value="Submit">

</form>

“About” page has information about website.

<center><h1>Visited by more than 1 million car shoppers each month,<br><br>

NETSHOP is the leading destination for online car shoppers,<br><br>

offering credible and easy-to-understand information from<br><br>

consumers and experts to help buyers formulate opinions <br><br>

on what to buy, where to buy and how much to pay for a car.<br><br>

With comprehensive pricing information, side-by-side <br><br>

comparison tools, photo galleries, unbiased editorial<br><br>

content and a large selection of new- and used-car<br><br>

inventory.<br><br><br>

Launched in June 2007. <br>

</h1></center>

“Support” page has textboxes (first name, email, message) and button “submit”.

<br> <h1> Support </h1> </br>

<br> <h3> How we can help you? </h3></br>

<br> <h3> We will do our best to reply as soon as possible. </h3> </br> <br> </br>

<h3> First name: <input style="margin-left: 30px; type="text" name="FirstName" value=""><br> </h3><br>

<h3> Email: <input style="margin-left: 64px; type="text" name="FirstName" value=""><br> </h3><br>

<h3> Message: <input style="margin-left: 41px;" type="text" name="Comment" value=" "><br></h3><br>

<br> </br>

<input type="submit" value="Contact us">

“Stores” page has information about store and map.

<br> <h1> Contact information </h1> </br>

<h3> Address: Raina bulvaris 19, Riga </h3>

<h3> Phone: +26700061 </h3>

<br> </br>

<h3> Opening hours </h3>

<h3> Mon.-Fri. 9:00 - 17:00 </h3>

<br> </br>

<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script><div style="overflow:hidden;height:300px;width:700px;"><div id="gmap\_canvas" style="height:300px;width:700px;"></div><style>#gmap\_canvas img{max-width:none!important;background:none!important}</style><a class="google-map-code" href="http://www.goertz-gutscheiin.com" id="get-map-data">goertz-gutscheiin.com</a></div><script type="text/javascript"> function init\_map(){var myOptions = {zoom:15,center:new google.maps.LatLng(56.9508026,24.116318800000045),mapTypeId: google.maps.MapTypeId.ROADMAP};map = new google.maps.Map(document.getElementById("gmap\_canvas"), myOptions);marker = new google.maps.Marker({map: map,position: new google.maps.LatLng(56.9508026, 24.116318800000045)});infowindow = new google.maps.InfoWindow({content:"<b>NetShop</b><br/>Latvia Raina bulvaris 19<br/> Riga" });google.maps.event.addListener(marker, "click", function(){infowindow.open(map,marker);});infowindow.open(map,marker);}google.maps.event.addDomListener(window, 'load', init\_map);</script>