**Salesforce Developers**

**Basic Knowledge Test: 1**

**Name: Anton Kaliev**

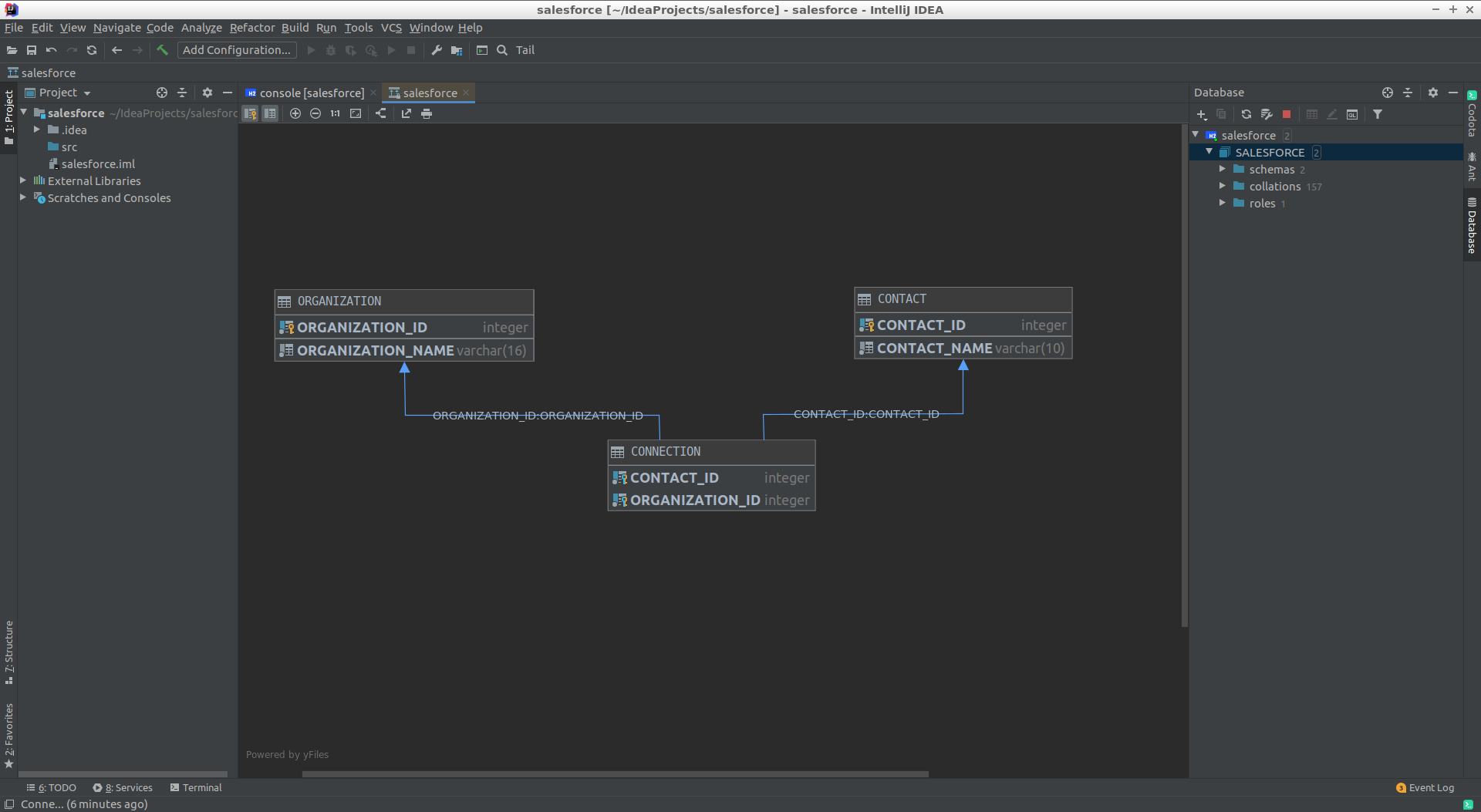
**Skype:**

**Phone:093 7 888 99 5**

**Note:** All test tasks should be completed in well-structured code/ scripts applying Java, Vanilla JS, SQL best practices and Code Conventions.

1. Your DB needs to hold two main objects – **Contacts** and **Organizations**. Each Contact can work in several organizations and each Organization has a lot of contacts.   
     
   - What type of relationship would you use?   
   - Draw the scheme of all tables with relationships.

Relationship Name:   
Many-to-many. We use a table of relationships to bring it to a one-to-many relationship, because many-to-many relationships are not possible in relational databases  
  
Schema:



create table Contacts

(

contact\_ID integer not null auto\_increment,

contact\_name varchar(10) not null,

primary key (contact\_ID)

);

create table Organization

(

organization\_ID integer not null auto\_increment,

organization\_name varchar(16) not null,

primary key (organization\_ID)

);

create table Connection

(

contact\_ID integer not null,

organization\_ID integer not null,

primary key (contact\_ID, organization\_ID),

foreign key (contact\_ID) references Contacts,

foreign key (organization\_ID) references Organization,

);

1. Now we have DB with each **Contact** has the only one related **Organization**.   
     
   - How can you find all contacts with **Name** starts with *“A”* in organization named *“Cats”*?   
   - How can you count number of contacts for each organization?   
   - Write SQL query for each question above.

1:

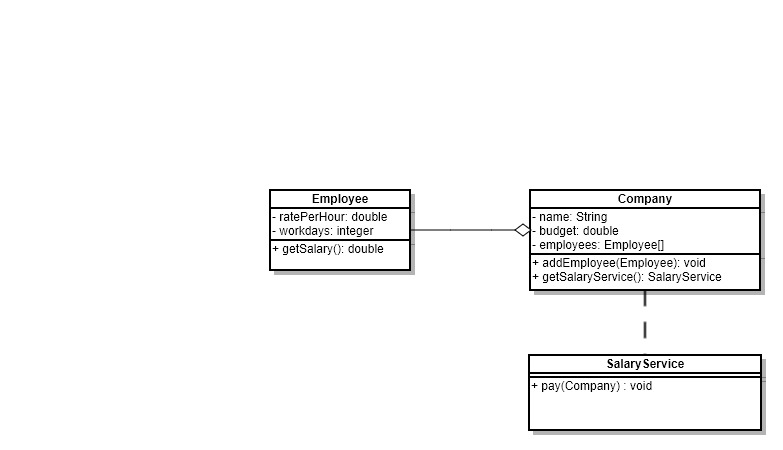
select Contacts.contact\_name  
from Connection  
 inner join Contacts on Connection.contact\_ID = Contacts.contact\_ID  
 inner join Organization on Connection.organization\_ID = Organization.organization\_ID  
where organization\_name = 'Cat'  
 and contact\_name like 'A%';

2:

select Organization.organization\_ID, Organization.organization\_name, *count*(Organization.organization\_name)  
from Connection  
 join Organization on Connection.organization\_ID = Organization.organization\_ID  
group by Organization.organization\_name;

1. As a part of project, you need to implement functionality of calculating salaries for employees in different companies. Each company have its own salary budget. Every employee has his own hourly rate and work hours. Basic formula for salary calculation is ratePerHour \* workdays \* 8

Each company should have ability to pay salaries to its workers. We should take in greater than company wage budget. UML diagram listed below.



1) Write classes that implement this diagram.   
2) You should have at least 3 types of employees  
e.g.

Manager

Developer

Scrum master

Developer may have bonuses that should be add to the basic salary, it should be taken into consideration during calculation of his wage.

package Main.Task3;  
  
import Main.Task3.Employee.Developer;  
import Main.Task3.Employee.Employee;  
import lombok.Getter;  
import org.jetbrains.annotations.NotNull;  
  
import java.util.ArrayList;  
import java.util.List;  
import static Main.Task3.ColorANCII.*RED*;  
  
public class Company implements ColorANCII{  
 @Getter  
 String name;  
 @Getter  
 double budget;  
 @Getter  
 private double availableBudget;  
 @Getter  
 private List<Employee> employees;  
 private SalaryService salaryService;  
  
 public Company(String name, double budget) {  
 this.name = name;  
 this.budget = budget;  
 this.availableBudget = budget;  
 this.employees = new ArrayList<>();  
 }  
  
 */\*\**  
 *\* Add an employee to the company*  
 *\**  
 *\** ***@param*** *employee*  
 *\*/*  
public synchronized void addEmployee(@NotNull Employee employee) {  
 //do we have enough budget?  
 if (employee.getSalary() > availableBudget)  
 throw new IllegalArgumentException("a current employee isn't in budget with total salary at " + employee.getSalary() +  
 " , but available budget is " + availableBudget);  
 if (employee.getSalary() <= 0)  
 System.*out*.println(*RED*("I hope you know what you do. Employee salary less than zero"));  
 employees.add(employee);  
 availableBudget -= employee.getSalary();  
 }  
  
 */\*\**  
 *\** ***@return*** *SalaryService, associated with this company*  
 *\*/*  
public SalaryService getSalaryService() {  
 if (null == salaryService)  
 salaryService = new SalaryService(this);  
 return salaryService;  
 }  
  
 */\*\**  
 *\* calculates and returns the total salary, including bonuses*  
 *\**  
 *\** ***@return*** *salary*  
 *\*/*  
double getTotalSalary() {  
 double result = 0.0;  
 for (Employee employee : employees)  
 result += employee.getSalary();  
 return result;  
 }  
  
 */\*\**  
 *\* try to give bonuses to the employee*  
 *\** ***@param*** *employee*  
 *\** ***@param*** *bonus*  
 *\*/*  
public void setBonus(Employee employee, double bonus) {  
 if(bonus > availableBudget)  
 throw new IllegalArgumentException("Too many bonuses, not enough budget");  
  
 try {  
 Developer dev = (Developer) employee;  
 dev.setBonus(bonus);  
 availableBudget -= bonus;  
 } catch (ClassCastException e) {  
 System.*err*.println();  
 throw new IllegalArgumentException("Only developer can have bonus, but current employee is " + employee.getClass().getSimpleName());  
 }  
 }  
  
  
}

package Main.Task3;  
  
import org.jetbrains.annotations.NotNull;  
  
public class SalaryService {  
 private Company company;  
  
 public SalaryService(@NotNull Company company) {  
 this.company = company;  
 }  
  
 */\*\**  
 *\* Check, if company has the ability to pay salaries*  
 *\** ***@param*** *company company to check*  
 *\** ***@throws*** *Exception will throw, if company have no ability to pay salaries*  
 *\*/*  
public static void pay(@NotNull Company company) throws Exception {  
 if (company.getTotalSalary() > company.getBudget())  
 throw new Exception(String.*format*("Companies %s need %f , but there are only %f . This is the reason for this exception",  
 company.getName(), company.getTotalSalary(), company.getBudget()));  
 System.*out*.println("The company has the ability to pay salaries. Budget balance "+ company.getAvailableBudget());  
 }  
 */\*\**  
 *\* Check, if associated to service company has the ability to pay salaries*  
 *\** ***@throws*** *Exception will throw, if company have no ability to pay salaries*  
 *\*/*  
public void pay() throws Exception {  
 *pay*(company);  
 }  
}

package Main.Task3.Employee;  
  
import Main.Task3.Main;  
  
public abstract class Employee {  
 double ratePerHour;  
 int workdays;  
 static int *id* = 0;  
 String name;  
  
 public Employee(double ratePerHour, int workdays) {  
 this.ratePerHour = ratePerHour;  
 this.workdays = workdays;  
 }  
  
// public Employee(double ratePerHour, int workdays, String name) {  
// this.ratePerHour = ratePerHour;  
// this.workdays = workdays;  
// this.name = name + " id:" + id;  
// id++;  
// }  
//  
// public Employee(double ratePerHour, int workdays, boolean addId) {  
// this.ratePerHour = ratePerHour;  
// this.workdays = workdays;  
// if (addId) {  
// this.name = "id:" + id;  
// this.id++;  
// }  
// }  
  
  
 public double getSalary() {  
 return ratePerHour \* workdays \* 8;  
 }  
}

package Main.Task3.Employee;  
  
import lombok.Getter;  
import lombok.Setter;  
  
public class Developer extends Employee {  
 @Getter  
 @Setter  
 private double bonus;  
  
 public Developer(double ratePerHour, int workdays) {  
 super(ratePerHour, workdays);  
 System.*out*.println("Developer war born");  
 }  
  
 @Override  
 public double getSalary() {  
 return super.getSalary() + bonus;  
 }  
  
}

package Main.Task3.Employee;  
  
public class Manager extends Employee {  
 public Manager(double ratePerHour, int workdays) {  
 super(ratePerHour, workdays);  
 System.*out*.println("Manager was hired");  
 }  
}

package Main.Task3.Employee;  
  
public class ScrumMaster extends Employee {  
 public ScrumMaster(double ratePerHour, int workdays) {  
 super(ratePerHour, workdays);  
 System.*out*.println("ScrumMaster war hired");  
 }  
}

1. You have a list of numbers to fill the table of prices in the shown way

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Jan** | **Feb** | **Mar** | **Apr** | **May** | **Jun** | **Jul** | **Aug** | **Sep** | **Oct** | **Nov** | **Dec** |
| Prod. 1 | Num1 | Num2 | Num3 | Num4 | Num5 | Num6 | Num7 | Num8 | Num9 | Num10 | Num11 | Num12 |
| Prod. 2 | Num13 | Num14 | … |  |  |  |  |  |  |  |  |  |
| Prod. 3 |  |  |  |  |  |  |  |  |  |  |  |  |

- Implement Java method which receive this list and return the object which will be convenient to find price using Month and Product keys.

package Main.Task4;  
  
import lombok.Getter;  
import lombok.Setter;  
import org.jetbrains.annotations.NotNull;  
  
import java.time.Month;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class ProductList {  
 private List<Product> productList;  
  
 public ProductList() {  
 this.productList = new ArrayList<Product>();  
 }  
  
 public ProductList(List<String> productName, List<Double> productPrices) {  
 this();  
 addProducts(productName, productPrices);  
 }  
  
 */\*\**  
 *\* Add many products to the ProductList*  
 *\**  
 *\** ***@param*** *productName list of products names*  
 *\** ***@param*** *productPrices list of products prices*  
 *\*/*  
public void addProducts(List<String> productName, List<Double> productPrices) {  
 if (productName.size() \* 12 < productPrices.size())  
 throw new IllegalArgumentException(String.*format*("Too many prices, too few products. There are %d products and %d prices.",  
 productName.size(), productPrices.size()));  
 for (int i = 0; i < productName.size(); i++) {  
 Product product = new Product(productName.get(i));  
 int firstIndex = i + 12 \* i;  
 int lastIndex = i + 12 \* i + 12;  
  
 if (firstIndex > productPrices.size())  
 throw new IndexOutOfBoundsException("Too many prices");  
 //not prices noy for all year  
 if (lastIndex > productPrices.size())  
 product.addPrice(productPrices.subList(i + 12 \* i, productPrices.size()));  
 else  
 product.addPrice(productPrices.subList(i + 12 \* i, i + 12 \* i + 12));  
 productList.add(product);  
 }  
 }  
  
 */\*\**  
 *\* Add single product to the ProductList, if it not already there, or set the price of the currently added product*  
 *\**  
 *\** ***@param*** *productName product Name*  
 *\** ***@param*** *month month*  
 *\** ***@param*** *price price*  
 *\*/*  
  
public void addValue(String productName, Month month, double price) {  
 Product product = productList.stream().filter(x -> x.productName.equals(productName)).findAny().orElse(null);  
 if (null == product) {  
 product = new Product(productName, month, price);  
 productList.add(product);  
 return;  
 }  
 product.setPrice(month, price);  
 }  
  
 */\*\**  
 *\* get Price value for the selected product. It will throw exceptions if something wrong.*  
 *\**  
 *\** ***@param*** *productName product Name*  
 *\** ***@param*** *month month*  
 *\** ***@return*** *price of the product*  
 *\** ***@throws*** *NullPointerException if there is no expected product. or its value for a given month hasn't been established.*  
 *\*/*  
public double getValue(String productName, Month month) throws NullPointerException {  
 Product product = productList.stream().filter(x -> x.productName.equals(productName)).findAny().orElse(null);  
 if (null == product)  
 throw new NullPointerException(String.*format*("no %s product in current list", productName));  
 try {  
 return product.getPrice(month);  
 } catch (NullPointerException e) {  
// e.printStackTrace();  
 throw new NullPointerException(String.*format*("no price for %s product at %s", productName, month));  
 }  
 }  
  
  
 public void print() {  
 System.*out*.println(String.*format*("Name |%9s |%9s |%9s |%9s |%9s |%9s |%9s |%9s |%9s |%9s |%9s |%9s |", Month.*values*()));  
 System.*out*.println("-----------------------------------------------------------------------------------------------------------------------------------------------");  
 for (Product product : productList) {  
 System.*out*.print(String.*format*("%8s |", product.getProductName()));  
 for (Month month : Month.*values*()) {  
 String s;  
 if (null == product.getPrice(month))  
 s = String.*format*("%8s |", null);  
 else  
 s = String.*format*("%8.2f |", product.getPrice(month));  
 System.*out*.print(s);  
  
 }  
 System.*out*.println();  
 }  
 }  
  
}  
  
class Product {  
 @Getter  
 @Setter  
 String productName;  
 private Map<Month, Double> productPrice;  
  
 */\*\**  
 *\* Create a product*  
 *\**  
 *\** ***@param*** *productName product name*  
 *\*/*  
Product(String productName) {  
 this.productName = productName;  
 productPrice = new HashMap<>();  
 }  
  
 Product(String productName, Month month, double price) {  
 this.productName = productName;  
 productPrice = new HashMap<>();  
 setPrice(month, price);  
 }  
  
  
 */\*\**  
 *\* Add a list of prices*  
 *\**  
 *\** ***@param*** *price list of product prices. Must be less than 12 elements long(as a number of month)*  
 *\** ***@throws*** *IllegalArgumentException expected 12 prices for 12 months, if more than throw exception.*  
 *\*/*  
void addPrice(List<Double> price) throws IllegalArgumentException {  
 if (price.size() > 12)  
 throw new IllegalArgumentException("invalid number of items, expected 12 numbers for 12 Months, but was " + price.size());  
 for (int i = 0; i < price.size(); i++) {  
 productPrice.put(Month.*of*(1 + i), price.get(i));  
 }  
 }  
  
 Double getPrice(Month month) {  
 return productPrice.get(month);  
 }  
  
 void setPrice(Month month, Double price) {  
 productPrice.put(month, price);  
 }  
}

1. Given todo list in file ‘Test\_Backend\_4’.



Implement following tasks using javaSript:

* 1. creation of the toDo list item.

.contentList {  
 list-style-type: none;  
 padding: 0;  
 margin: 0;  
 background: aliceblue;  
 color: black;  
 cursor: pointer;  
 position: relative;  
 }  
  
 ul li:hover {  
 background: #ddd;  
 }  
  
 /\* Style the close button \*/  
 .close {  
 position: absolute;  
 right: 0;  
 padding: 1px 1px 1px 1px;  
 }  
  
 .close:hover {  
 background-color: #f44336;  
 color: white;  
 }  
  
 </style>  
  
</head>  
<body>  
  
<div class="todo">  
 <div class="todo-canvas">  
  
 <div class="todo-header">  
 <button class="todo-button todo-button-add" id="todo-button-add">+</button>  
 </div>  
 <div class="todo-list">  
 <div class="todo-card" id="card-1">  
 <label class="label">Content</label>  
 <input class="content" type="text" id="task-value"/>  
 </div>  
 <div>  
 <ul class="contentList" id="task-list">  
 <li>111111111111111</li>  
 <li>222222222222222</li>  
 <li>333333333333333</li>  
 <li>444444444444444</li>  
 <li>555555555555555</li>  
 <li>666</li>  
 </ul>  
 </div>  
 </div>  
 </div>  
  
</div>  
<script>  
  
 let ***addButton*** = ***document***.getElementById("todo-button-add");  
 let ***taskValue*** = ***document***.getElementById("task-value");  
 let ***taskList*** = ***document***.getElementById("task-list");  
 //task will be added by clicking on the Add-button, or by pressing ENTER  
 ***addButton***.addEventListener("click", addTask);  
 ***taskValue***.addEventListener("keypress", function (ev) {  
 if (ev.key === 'Enter') {  
 addTask();  
 }  
 });  
  
 //add task and two control buttons for it  
 function addTask() {  
 //check, if no message  
 if ("" === ***taskValue***.value) {  
 alert("no task value");  
 return;  
 }  
 let li = ***document***.createElement("li");  
 li.append(***taskValue***.value);  
 li.append(deleteButton());  
 ***taskList***.append(li);  
 ***taskValue***.value = "";  
 ***console***.log("added element");  
 }  
  
  
 function deleteButton() {  
 let span = ***document***.createElement("span");  
 span.className = "close";  
 span.append("\u00D7");  
 span.onclick = function () {  
 this.parentElement.remove();  
 };  
 return span;  
 }  
  
 //inin block  
 // Create a "close" button and append it to each list item  
 let ***taskListElements*** = ***document***.getElementsByTagName("LI");  
  
 for (let ***i*** = 0; ***i*** < ***taskListElements***.length; ***i***++) {  
 ***taskListElements***[***i***].append(deleteButton());  
 }  
  
</script>

1. Why have you chosen to be developer? Describe your main objectives and hopes about your future carrier.

The developer today is a profession, that give you a way to work by your mind, impruve your skills, and doesn't need a lot of health or many special equipment on it. I like to write code, and I like to create something new. Well, it’s easier for me to work with a computer than with people.

From my career, I expect to be able to improve my skills. That my projects will be interesting. And that I will meet people from whom I can learn something.