Class Descriptions for CRM System

March 3, 2025

Class: DatabaseConnection

This class is responsible for establishing a connection to the MySQL database. It uses the JDBC API to interact with the database.

Private Attributes

- `private static final String URL`: This is the URL for the MySQL database, which includes the hostname (localhost), the port (3306), and the database name (CRM). It's a constant because the database URL will remain the same throughout the application.
- `private static final String USER`: This stores the username (root) used to authenticate the connection to the database. It is also a constant.
- `private static final String PASSWORD`: This stores the password (1992) for the database connection. Like the others, it's a constant.

Public Methods

- `public static Connection getConnection() throws SQLException`: This method establishes the connection to the database using the `DriverManager.getConnect` method and returns the Connection object. It uses the URL, USER, and PASSWORD attributes to create the connection. If the connection cannot be established, it throws an `SQLException`.

Static Attributes/Methods

The URL, USER, and PASSWORD attributes are declared as static because they are shared across all instances of the class. They are also final, meaning their values cannot be changed once assigned. The `getConnection()` method is static, which means it can be called without creating an instance of the DatabaseConnection class. It is commonly used to get a single connection to the database from anywhere in the application.

Usage

This class is typically used to manage database connections across the application, ensuring that all database operations have access to a valid connection. The `getConnection()` method would be used in other parts of the application to obtain a connection object, which would be used for executing SQL queries and updates.

Class: Person

This class represents a person in the CRM system. It holds details such as the person's ID, name, contact information, and address. It's a fundamental class, as other entities like Employee and Customer reference the Person class through the person_id foreign key.

Private Attributes

- `private int personId`: This attribute stores the unique identifier for the person. It corresponds to the `person_id` in the database.
- `private String firstName`: This stores the first name of the person.
- `private String lastName`: This stores the last name of the person.
- `private String email`: This stores the email address of the person.
- `private String phone`: This stores the phone number of the person.
- `private String address`: This stores the address of the person.

Constructors

- `public Person()`: This is a no-argument constructor that allows for the creation of a Person object without initializing any attributes.
- `public Person(int personId, String firstName, String lastName, String email, String phone, String address)`: This is a parameterized constructor that initializes the Person object with the given values for personId, firstName, lastName, email, phone, and address.

Getters and Setters

- `getPersonId()` / `setPersonId(int personId)`: Get and set methods for personId.
- `getFirstName()` / `setFirstName(String firstName)`: Get and set methods for firstName.
- `getLastName()` / `setLastName(String lastName)`: Get and set methods for lastName.
- `getEmail()` / `setEmail(String email)`: Get and set methods for email.
- `getPhone()` / `setPhone(String phone)`: Get and set methods for phone.
- `getAddress()` / `setAddress(String address)`: Get and set methods for address.

These methods allow access and modification of the private attributes.

Override Method

- `toString()`: This method returns a string representation of the Person object, displaying all the attributes in a human-readable format. This is particularly useful for debugging and logging purposes.

Static and Public Methods

This class does not have any static methods or static attributes, as it represents individual Person objects. All methods (getters, setters, and constructor) are public so that other parts of the program can interact with the Person object.

Relationships

The Person class is used by both the Employee and Customer classes. In both cases, a Person object is referenced through the foreign key `person_id`. The attributes in the Person class directly correspond to columns in the Person table in the database.

Summary

This class is essential for the CRM system, serving as a base class for storing personal information. The other classes, like Employee and Customer, inherit this class by linking to the `person_id`.

Class: PersonDAO

This class provides data access functionality for the Person entity. It is responsible for performing CRUD operations (Create, Read, Update, Delete) on the Person table in the database. It uses SQL queries to interact with the database, handling Person objects.

Private Attributes

- `private Connection connection`: This is a Connection object used to interact with the database. It is passed to the constructor and used to execute SQL queries.

Constructor

- `public PersonDAO(Connection connection)`: This constructor initializes the connection attribute with the provided Connection object, which is used to communicate with the database.

Public Methods

- `public void insertPerson(Person person) throws SQLException`: This method inserts a new Person into the Person table in the database. It uses a PreparedStatement to securely execute the INSERT SQL query with the person's data.
- `public List<Person> getAllPersons() throws SQLException`: This method retrieves all Person records from the database. It uses a Statement to execute a SELECT query and returns a list of Person objects. Each Person is created using the data retrieved from the database.
- `public Person getPersonById(int personId) throws SQLException`: This method retrieves a Person by their person_id. It uses a PreparedStatement to execute a query with the provided ID and returns a Person object if found, otherwise returns null.
- `public List<Person> getPersonsByLastName(String lastName) throws SQLException`: This method retrieves all Person records with the specified last_name. It returns a list of Person objects matching the last name from the database.
- `public void updatePerson(Person person) throws SQLException`: This method updates an existing Person record in the database based on the person_id. It uses a PreparedStatement to execute an UPDATE query that modifies the attributes of the Person record.
- `public void deletePerson(int personId) throws SQLException`: This method deletes a Person record from the database. It first deletes the associated records in the Customer table (if any), and then deletes the Person record itself. The PreparedStatement is used to execute the deletion queries.

Static and Public Methods

The PersonDAO class does not have any static methods. All methods are instance methods, meaning they operate on a specific instance of the PersonDAO class. All methods are public so that other parts of the application can interact with the Person data stored in the database.

Database Operations

- `PreparedStatement`: This is used for all SQL queries to prevent SQL injection and to securely handle user input.
- `Statement`: This is used for the SELECT queries that do not require any user input.
- `SQLException`: This exception is thrown if there is an issue with any database operations, ensuring that database errors are handled.

Relationships

`Person` and `Customer`: The `deletePerson()` method demonstrates the relationship between the Person and Customer tables. Since the Person table has a foreign key reference in the Customer table (via `person_id`), the method first deletes any associated customer records before deleting the person.

Summary

This DAO (Data Access Object) class abstracts the operations related to the Person table, allowing other classes to interact with the database through high-level methods. It follows the basic CRUD pattern and ensures that the operations are executed securely and efficiently.

Class: Employee

This class contains information about an employee, including personal and job-related data. It is designed to be used in conjunction with the `Person` class, with a `personId` linking the `Employee` to a corresponding `Person` record in the database.

Attributes:

- `private int employeeId`: A unique identifier for each employee.
- `private int personId`: This links the employee to a corresponding `Person` record in the database (foreign key).

- `private String jobTitle`: The employee's job title (e.g., Manager, Developer, etc.).
- `private String department`: The department the employee belongs to (e.g., HR, Sales, IT, etc.).
- `private double salary`: The employee's salary.
- `private Date hireDate`: The date the employee was hired.

Constructor:

- `public Employee()`: Default constructor for creating an empty `Employee` object.
- `public Employee(int employeeId, int personId, String jobTitle, String department, double salary, Date hireDate)`: A constructor to initialize all attributes of the `Employee` object.

Getters and Setters:

These methods allow access to the private attributes and provide ways to set or get their values.

- `getEmployeeId(), setEmployeeId(int employeeId)`
- `getPersonId(), setPersonId(int personId)`
- `getJobTitle(), setJobTitle(String jobTitle)`
- `getDepartment(), setDepartment(String department)`
- `getSalary(), setSalary(double salary)`
- `getHireDate(), setHireDate(Date hireDate)`

toString() Method:

The `toString()` method provides a string representation of the `Employee` object. This helps in printing out the employee's details in a human-readable format.

Summary:

The `Employee` class contains the essential information about an employee. The `personId` links the `Employee` to the `Person` class, which may store personal details.

EmployeeDAO Class

The `EmployeeDAO` class handles database operations related to the Employee entity. It performs CRUD operations (Create, Read, Update, and Delete) for the Employee records in the database.

Class: EmployeeDAO

Attributes:

- `private Connection connection`: This represents the database connection, which is passed into the constructor.

Constructor:

- `public EmployeeDAO(Connection connection)`: This constructor initializes the DAO with a `Connection` object, which is used for executing SQL queries.

Methods:

1. `insertEmployee(Employee employee)`:
    - Inserts a new Employee record into the Employee table.
    - SQL query: `INSERT INTO Employee (person_id, job_title, department, salary, hire_date) VALUES (?, ?, ?, ?, ?)`
    - `PreparedStatement` is used to insert the Employee attributes.
2. `getAllEmployees()`:
    - Retrieves all Employee records from the Employee table.
    - Executes a `SELECT` query and maps each row to an Employee object, which is added to a list.

- Returns a list of all employees.

3. `getEmployeeById(int employeeId)`:
   - Retrieves an Employee record by its `employeeId`.
   - Executes a `SELECT` query with a parameterized ID and returns the corresponding Employee object if found, or null if no employee is found.

4. `updateEmployee(Employee employee)`:
   - Updates an existing Employee record.
   - SQL query: `UPDATE Employee SET person_id = ?, job_title = ?, department = ?, salary = ?, hire_date = ?  WHERE employee_id = ?`
   - `PreparedStatement` is used to update the employee's attributes based on their `employeeId`.

5. `deleteEmployee(int employeeId)`:
   - Deletes an Employee record from the Employee table based on the `employeeId`.
   - SQL query: `DELETE FROM Employee WHERE employee_id = ?`

General Approach:

- The methods use `PreparedStatements` to prevent SQL injection and execute database queries.
- `ResultSets` are used to fetch data and map it to the Employee object.
- Transactions (in case of updates/deletes) are executed with proper exception handling to ensure data consistency.

Summary:

- `Insert:` Adds a new employee record to the database.
- `Read:` Retrieves employee records (all or by ID).
- `Update:` Updates an existing employee's details.

- `Delete`: Removes an employee record.

The DAO class allows for cleaner and more maintainable code, especially as the number of entities and operations grows.

Leads Class

The `Leads` class represents a lead in a CRM system. A lead typically refers to a potential customer or prospect, which has various attributes such as contact details, source, status, and the employee associated with the lead.

Class: Leads

Attributes:

- `leadId`: A unique identifier for the lead.
- `name`: The name of the lead.
- `email`: The lead's email address.
- `phone`: The lead's phone number.
- `source`: The origin/source of the lead (e.g., referral, website, etc.).
- `status`: The current status of the lead (e.g., New, Contacted, Qualified, etc.).
- `employeeId`: The ID of the employee who is handling or responsible for this lead.

Constructors:

- `public Leads()`: Default constructor for creating an empty lead object.
- `public Leads(int leadId, String name, String email, String phone, String source, String status, int employeeId)`: A parameterized constructor to initialize the lead with values.

Getters and Setters:

- Standard getter and setter methods are provided for each attribute to access and modify them.

toString() Method:

- Provides a string representation of the `Leads` object, which is useful for debugging and logging.

LeadsDAO Class

The `LeadsDAO` class provides data access functionality for managing leads in your CRM system. It performs operations such as adding, retrieving, updating, and deleting leads in the database. The class is designed to handle these tasks using JDBC, and it works directly with SQL queries and prepared statements.

Class: LeadsDAO

Attributes:

- `connection`: The `Connection` object for interacting with the database.

Constructor:

- `LeadsDAO(Connection connection)`: Initializes the `LeadsDAO` with the provided database connection.

Methods:

1. `addLead(Leads lead)`:
    - Inserts a new lead into the Leads table.
    - Uses a `PreparedStatement` to safely insert the data, avoiding SQL injection.
2. `getAllLeads()`:
    - Retrieves all the leads from the Leads table.
    - Uses a `Statement` to execute the query and then maps the result set to Leads objects, which are added to a list.

3. `deleteLead(int leadId)`:
    - Deletes a lead based on the `leadId`.
    - A `PreparedStatement` is used to safely delete the record.
4. `getLeadById(int leadId)`:
    - Retrieves a lead based on the given `leadId`.
    - A `PreparedStatement` is used to fetch the lead's details from the database, and the result is returned as a `Leads` object.
5. `updateLead(Leads lead)`:
    - Updates an existing lead in the database.
    - It modifies the attributes of the lead (e.g., name, email, status, etc.) and uses a `PreparedStatement` to update the corresponding record in the Leads table.

Campaigns Class

Attributes:

- `campaignId`: The unique identifier for each campaign.
- `name`: The name of the campaign.
- `startDate`: The start date of the campaign.
- `endDate`: The end date of the campaign.
- `budget`: The budget allocated for the campaign.
- `employeeId`: The ID of the employee managing the campaign.

Constructor:

- `Campaigns(int campaignId, String name, Date startDate, Date endDate, double budget, int employeeId)`: A constructor to initialize the campaign object with the provided attributes.

Getters and Setters: Standard getters and setters are provided for all attributes.

Method:

- `toString()`: This method provides a string representation of the `Campaigns` object, making it easier to print the object's details for debugging or logging purposes.

CampaignsDAO Class

Attributes:

- `connection`: A `Connection` object used to interact with the database.

Methods:

1. `addCampaign(Campaigns campaign)`:
   - Inserts a new campaign into the database using the INSERT SQL query.
   - Uses a PreparedStatement to set the campaign attributes and execute the query.
2. `getAllCampaigns()`:
   - Retrieves all campaigns from the Campaigns table using a SELECT query.
   - Iterates through the ResultSet and creates a list of `Campaigns` objects.
3. `deleteCampaign(int campaignId)`:
   - Deletes a specific campaign from the database based on the given `campaignId`.
   - Uses the DELETE SQL query to remove the campaign.
4. `getCampaignById(int campaignId)`:
   - Retrieves a campaign by its ID from the database.

- Uses the SELECT SQL query to fetch the campaign and return it as a `Campaigns` object.

5. `updateCampaign(Campaigns campaign)`:
   - Updates an existing campaign in the database.
   - Uses the UPDATE SQL query to modify the campaign's attributes based on the provided `Campaigns` object.

Accounts Class

Attributes:

- `accountId`: The unique identifier for the account.
- `customerId`: The ID of the customer who owns the account.
- `accountNumber`: The account number associated with the account.
- `creationDate`: The date the account was created.
- `accountType`: The type of the account (e.g., savings, checking, business).
- `balance`: The current balance of the account.

Constructors:

- Default constructor: Initializes the object with default values.
- Parameterized constructor: Initializes the object with values provided for each attribute.

Getters and Setters: The class provides getter and setter methods for each attribute.

Method:

- `toString()`: This method provides a string representation of the `Accounts` object, including all of its attributes in a readable format.

AccountsDAO Class

Methods:

1. `addAccount(Accounts account)`:
    - Adds a new account to the Accounts table by inserting the provided `Accounts` object into the database.
    - SQL Query: `INSERT INTO Accounts (customer_id, account_number, creation_date, account_type, balance) VALUES (?, ?, ?, ?, ?)`
2. `getAllAccounts()`:
    - Retrieves all accounts from the Accounts table and returns them as a list of `Accounts` objects.
    - SQL Query: `SELECT * FROM Accounts`
3. `deleteAccount(int accountId)`:
    - Deletes an account from the Accounts table using the provided `accountId`.
    - SQL Query: `DELETE FROM Accounts WHERE account_id = ?`
4. `getAccountById(int accountId)`:
    - Retrieves an account from the Accounts table based on the provided `accountId` and returns it as an `Accounts` object.
    - SQL Query: `SELECT * FROM Accounts WHERE account_id = ?`
5. `updateAccount(Accounts account)`:
    - Updates an existing account in the Accounts table with the provided `Accounts` object.
    - SQL Query: `UPDATE Accounts SET customer_id = ?, account_number = ?, creation_date = ?, account_type = ?, balance = ?  WHERE account_id = ?`

Tickets Class

Fields:

- `ticketId`: The unique ID for the ticket.
- `customerId`: The ID of the customer who raised the ticket.
- `issueDescription`: A description of the issue or problem the customer reported.
- `status`: The current status of the ticket (e.g., "Open", "Closed", "In Progress").
- `createdDate`: The date when the ticket was created.

Constructors:

- Default Constructor: Initializes a new `Tickets` object with default values.
- Parameterized Constructor: Initializes a new `Tickets` object with specific values for each field.

Getters and Setters: Provides methods for getting and setting the values of each field.

Method:

- `toString()`: Returns a string representation of the `Tickets` object in a human-readable format.

TicketsDAO Class

Constructor:

- `TicketsDAO(Connection connection)`: Initializes the DAO with a given `Connection` object.

Methods:

1. `addTicket(Tickets ticket)`:

- Adds a new ticket to the Tickets table by inserting the ticket's details (customer ID, issue description, status, and created date).

2. `getAllTickets()`:
    - Retrieves all tickets from the Tickets table. It returns a list of `Tickets` objects.
3. `deleteTicket(int ticketId)`:
    - Deletes a ticket based on the provided `ticketId`.
4. `getTicketById(int ticketId)`:
    - Retrieves a specific ticket based on its ID. If the ticket is found, it returns the corresponding `Tickets` object.
5. `updateTicket(Tickets ticket)`:
    - Updates an existing ticket in the Tickets table based on its ID, modifying its customer ID, issue description, status, and created date.

Class: Opportunities

Fields:

- `opportunityId`: The unique identifier for the opportunity.
- `customerId`: The ID of the customer associated with the opportunity.
- `description`: A description of the opportunity (e.g., potential sale).
- `estimatedValue`: The estimated value of the opportunity.
- `stage`: The current stage of the opportunity (e.g., "Initial", "Negotiation", "Closed").
- `createdDate`: The date the opportunity was created.

Constructor:

- `Opportunities()`: Default constructor.

- `Opportunities(int opportunityId, int customerId, String description, double estimatedValue, String stage, Date createdDate)`: A constructor that initializes the class fields.

Getters and Setters:

- `getOpportunityId()`: Returns the opportunity ID.
- `setOpportunityId(int opportunityId)`: Sets the opportunity ID.
- `getCustomerId()`: Returns the customer ID.
- `setCustomerId(int customerId)`: Sets the customer ID.
- `getDescription()`: Returns the opportunity description.
- `setDescription(String description)`: Sets the opportunity description.
- `getEstimatedValue()`: Returns the estimated value of the opportunity.
- `setEstimatedValue(double estimatedValue)`: Sets the estimated value.
- `getStage()`: Returns the stage of the opportunity.
- `setStage(String stage)`: Sets the stage of the opportunity.
- `getCreatedDate()`: Returns the created date of the opportunity.
- `setCreatedDate(Date createdDate)`: Sets the created date of the opportunity.

toString():

- Returns a string representation of the opportunity's details.

Class: OpportunitiesDAO

Fields:

- `connection`: A Connection object for connecting to the database.

Constructor:

- `OpportunitiesDAO(Connection connection)`: Initializes the DAO with a database connection.

Methods:

- `addOpportunity(Opportunities opportunity)`:
  - Inserts a new opportunity into the database.
  - Uses a PreparedStatement to safely insert values into the Opportunities table.
- `getAllOpportunities()`:
  - Retrieves all opportunities from the database.
  - Executes a SELECT query and returns a list of Opportunities objects.
- `deleteOpportunity(int opportunityId)`:
  - Deletes an opportunity from the database based on its ID.
  - Uses a PreparedStatement to execute the delete query.
- `getOpportunityById(int opportunityId)`:
  - Retrieves a specific opportunity by its ID.
  - Executes a SELECT query and returns an Opportunities object if found.
- `updateOpportunity(Opportunities opportunity)`:
  - Updates an existing opportunity in the database.
  - Uses a PreparedStatement to execute the UPDATE query based on the opportunity's ID.

Class: Activities

Fields:

- `activityId`: Unique identifier for the activity.
- `relatedType`: The type of the related entity (e.g., "Customer", "Opportunity").
- `relatedId`: The ID of the related entity (e.g., customer or opportunity ID).
- `activityType`: The type of activity (e.g., "Meeting", "Call", "Email").
- `activityDate`: The date when the activity took place.
- `notes`: Any additional notes or details about the activity.

Constructor:

- `Activities(int activityId, String relatedType, int relatedId, String activityType, Date activityDate, String notes)`: Initializes an activity with all its fields.
- `Activities()`: Default constructor for creating an empty activity.

Getters and Setters:

- These methods provide access to the activity's fields, allowing you to retrieve or modify each field.

toString() Method:

- This method provides a string representation of the Activities object, which is useful for debugging and logging.

Notes:

- The `relatedType` field is useful for linking activities to various types of entities (e.g., customers, opportunities).
- The `relatedId` is the foreign key that links the activity to the actual entity (like the customer or opportunity ID).

- `activityType` could be any action such as a "Call", "Email", "Meeting", or other types of interactions.
- The `activityDate` should be stored as a Date object, which can be converted into a java.sql.Date object when inserting or retrieving from the database.

Class: ActivitiesDAO

Fields:

- `connection`: A Connection object that represents the database connection.

Methods:

- `addActivity(Activities activity)`:
  - This method inserts a new activity into the database. It takes an Activities object and uses a PreparedStatement to insert its details into the Activities table.
- `getAllActivities()`:
  - This method retrieves all activities from the database and returns them as a list of Activities objects. It executes a SELECT query, iterates through the result set, and creates Activities objects.
- `deleteActivity(int activityId)`:
  - This method deletes an activity from the database based on its activityId. It uses a PreparedStatement to perform the DELETE query.
- `getActivityById(int activityId)`:
  - This method retrieves a single activity based on its activityId. It returns the corresponding Activities object if found, or null if not.
- `updateActivity(Activities activity)`:

  - This method updates an existing activity in the database using the activityId to identify the record. It updates the relevant fields (such as `relatedType`, `activityType`, `notes`, etc.) using a PreparedStatement.

CustomerDAO Class

The `CustomerDAO` (Data Access Object) class handles the interactions with the database for the `Customer` entity. It provides methods for performing CRUD operations on customer records in the database. The `CustomerDAO` class uses the `Connection` object passed to it to execute SQL queries.

Methods

- `insertCustomer()`:
  - This method inserts a new customer into the database.
  - It uses a `PreparedStatement` to insert the `personId`, `companyName`, `registrationDate`, and `totalSpent` values into the `Customer` table.
  - After executing the insert, it retrieves the generated `customerId` (the primary key) and sets it in the customer object.
- `getAllCustomers()`:
  - This method retrieves all customers from the database.
  - It executes a `SELECT` query and returns a list of `Customer` objects.
- `getCustomerById()`:
  - This method retrieves a specific customer by their `customerId`.
  - It uses a `PreparedStatement` to run a `SELECT` query with a `WHERE` clause and returns the customer object if found.
- `updateCustomer()`:
  - This method updates the details of an existing customer in the database.

  - It executes an `UPDATE` query to modify the `personId`, `companyName`, `registrationDate`, and `totalSpent` for the specified `customerId`.

- `deleteCustomer()`:
  - This method deletes a customer from the database using their `customerId`.
  - It uses a `PreparedStatement` with a `DELETE` query to remove the record.

Overall Overview

- `Customer Class`: Represents the structure of customer data, storing individual customer attributes and providing getters and setters.
- `CustomerDAO Class`: Manages the persistence of customer data in the database, providing methods to insert, retrieve, update, and delete customer records.

These classes form the basis for managing customer information in your CRM system. The `Customer` class acts as the data model, while the `CustomerDAO` class performs the database operations to manipulate customer data.