**Паттерн Посетитель(Visitor)**

**Версия без паттерна**

При построении объектной модели company и employee, которая должна взаимодействовать с реляционной базой данных, возникают определенные сложности из-за различий в механизмах структурирования данных. Реляционные базы данных не сохраняют некоторые характеристики объектов, такие как коллекции и наследование, что может привести к несоответствию между объектной моделью и структурой базы данных.

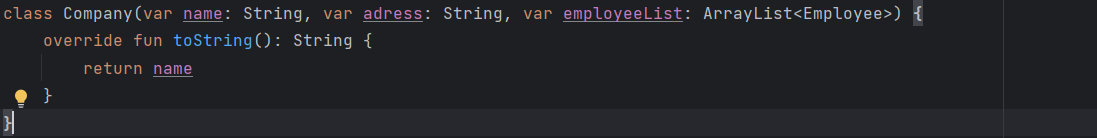
Для обмена данными между объектами и реляционной базой данных необходимо учитывать это несовпадение схем. Изменения в одной из структур могут потребовать соответствующих изменений в другой, что усложняет задачу обмена данными.

Чтобы решить эту проблему, можно использовать слой программного обеспечения, который будет отделять объекты в оперативной памяти от базы данных. Этот слой, называется преобразователем данных (Data Mapper), отвечает за передачу данных между объектами и базой данных, обеспечивая изоляцию между ними. Объекты в оперативной памяти не нуждаются в знании о структуре базы данных, так как вся работа с базой данных выполняется через преобразователь данных.

Преобразователь данных позволяет скрыть детали работы с базой данных от объектов в оперативной памяти, что упрощает процесс обмена данными и поддержку системы. Такой подход позволяет объектам работать с данными без необходимости знания SQL-запросов или деталей структуры базы данных, делая систему более гибкой и легко расширяемой.

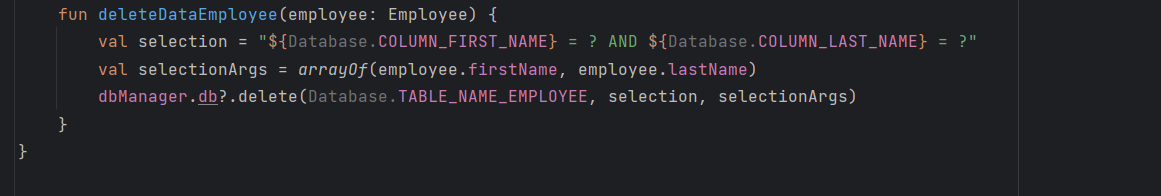
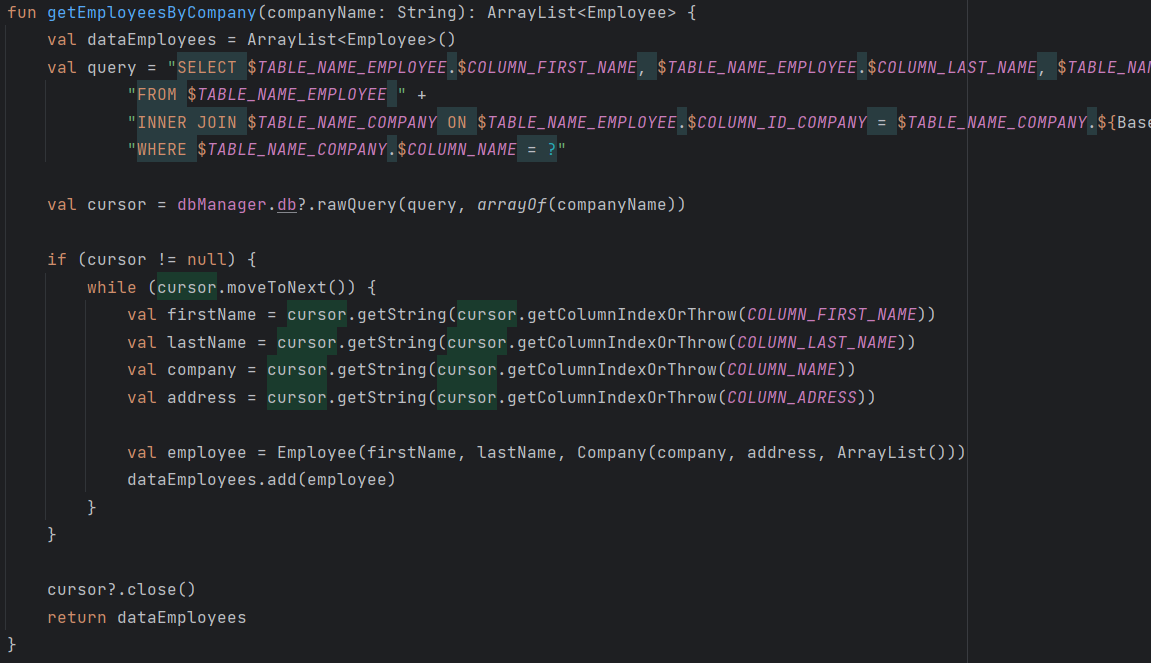
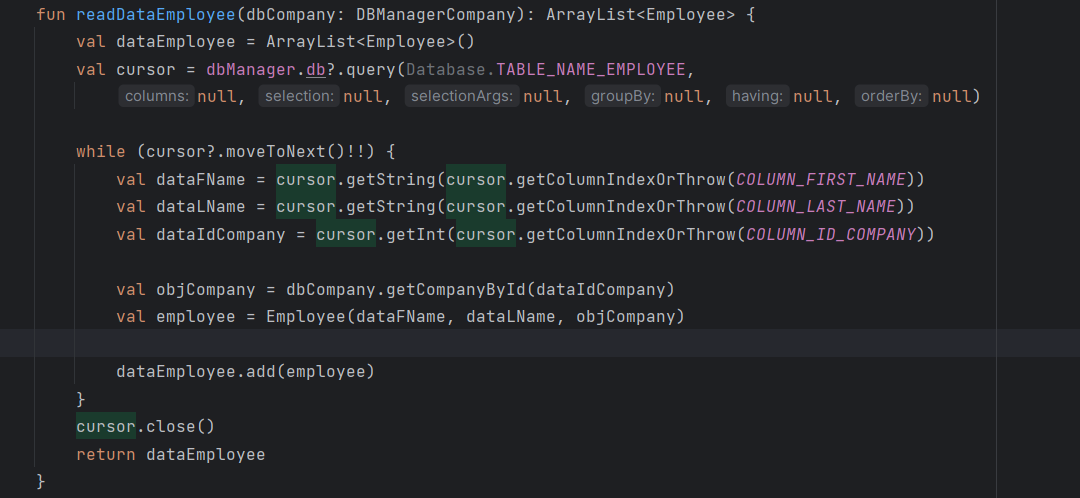
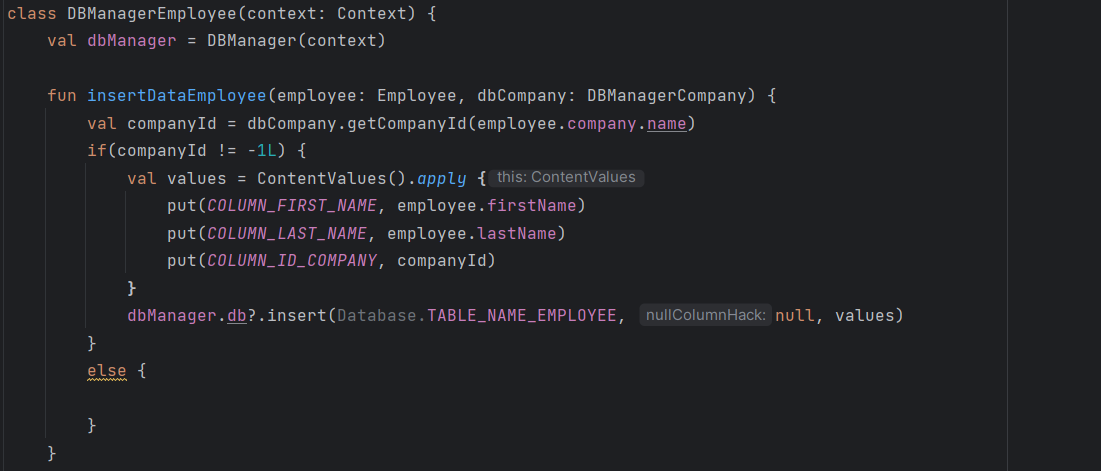
**Версия с паттерном**

**Диаграмма классов  
**

****

В данном случае у нас есть два класса: Company и Employee, которые представляют компанию и сотрудника соответственно. Компания имеет коллекцию сотрудников, а каждый сотрудник принадлежит одной компании.

**Employee Mapper**



Концепция маппера DBManagerCompany для сущности Company:

1. Метод insertDataCompany(company: Company) добавляет данные о компании в базу данных.

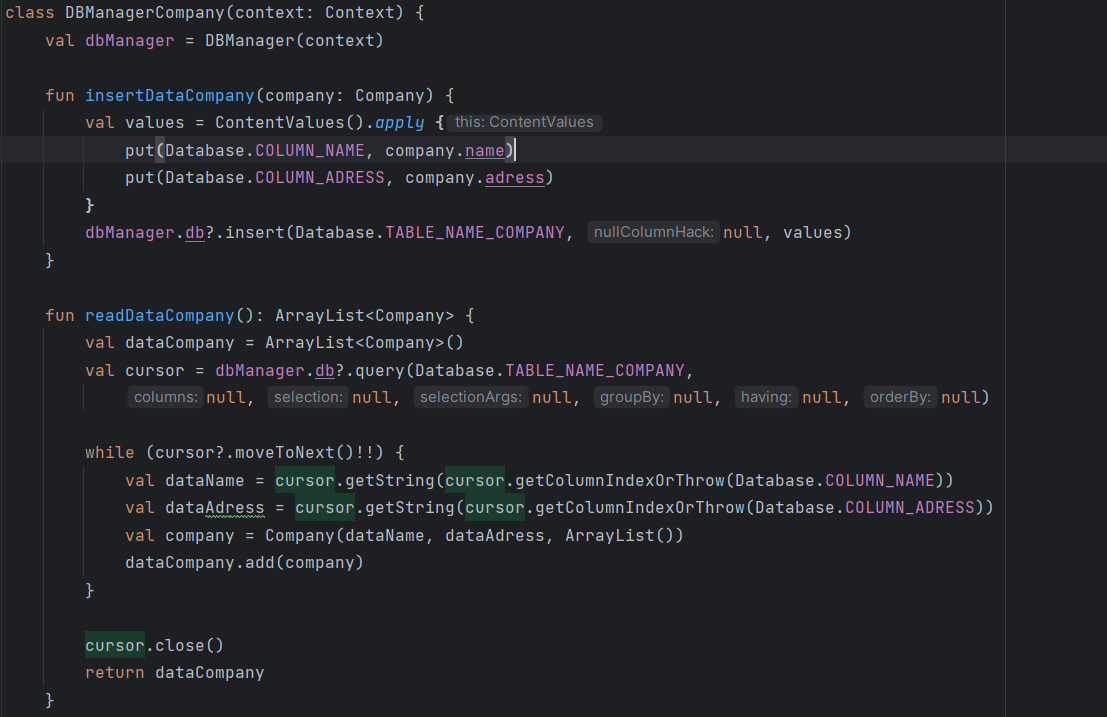
2. Метод readDataCompany(): ArrayList<Company> считывает данные о компаниях из базы данных.

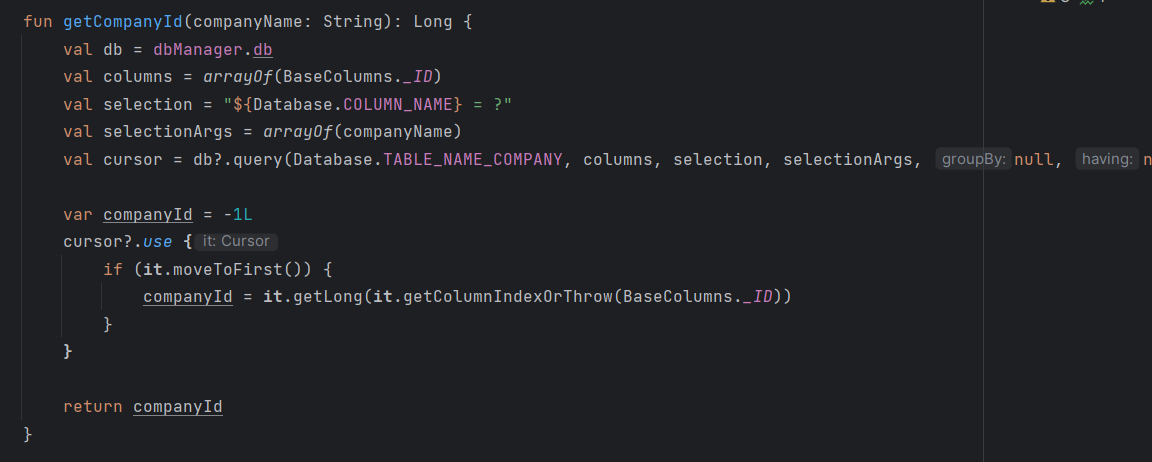
3. Метод getCompanyId(companyName: String): Long возвращает идентификатор компании по имени.

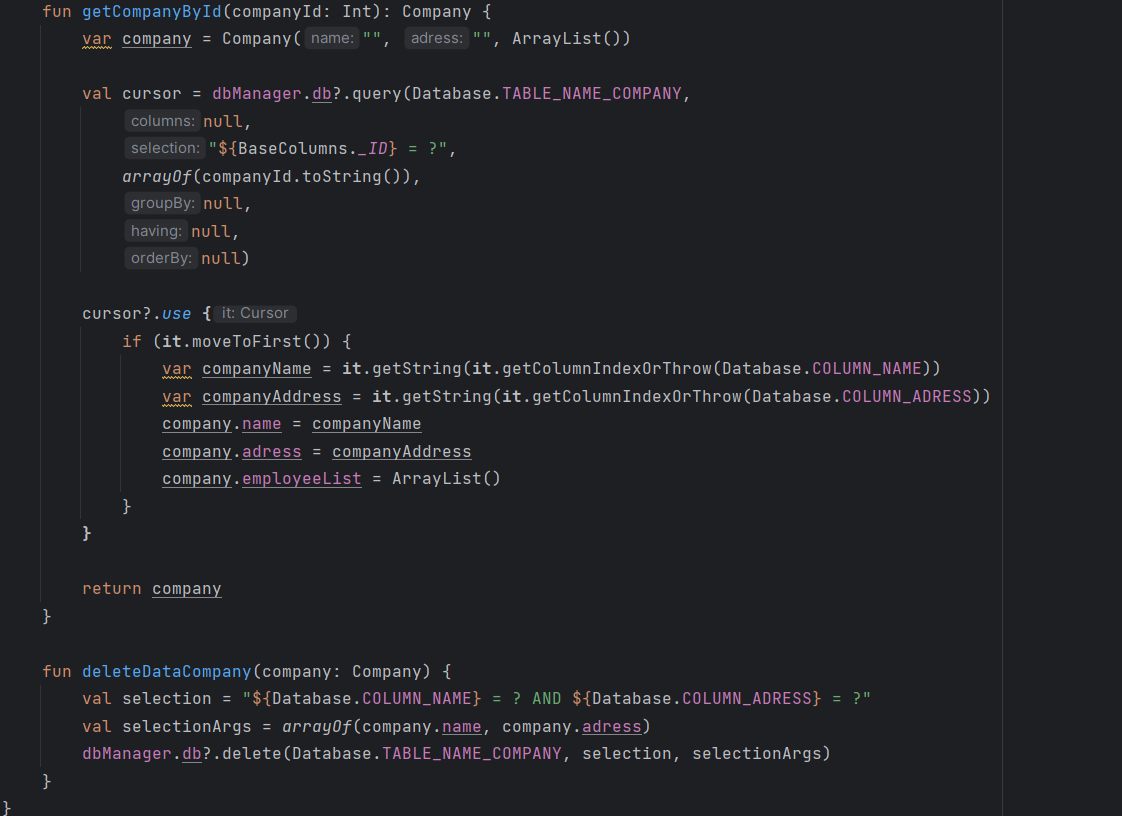
4. Метод getCompanyById(companyId: Int): Company получает данные о компании по идентификатору.

5. Метод deleteDataCompany(company: Company) удаляет данные о компании из базы данных.

**Company Mapper**







Пример маппера DBManagerEmployee для сущности Employee:

1. Метод insertDataEmployee(employee: Employee, dbCompany: DBManagerCompany) добавляет данные о сотруднике в базу данных.

2. Метод readDataEmployee(dbCompany: DBManagerCompany): ArrayList<Employee> считывает данные о сотрудниках из базы данных.

3. Метод getEmployeesByCompany(companyName: String): ArrayList<Employee> возвращает список сотрудников по имени компании.

4. Метод deleteDataEmployee(employee: Employee) удаляет данные о сотруднике из базы данных.  
  
Мапперы DBManagerCompany и DBManagerEmployee представляют собой классы, отвечающие за взаимодействие с базой данных для компаний и сотрудников соответственно. Они обеспечивают управление данными, сохраняют их в базе, читают из неё и обновляют при необходимости.

Преимущества:

1. Разделение ответственностей: Паттерн Mapper помогает разделить код, отвечающий за работу с данными, от кода бизнес-логики, что улучшает читаемость и поддерживаемость кода.

2. Гибкость: Использование маппера позволяет легко изменять структуру базы данных без влияния на бизнес-логику приложения.

3. Тестируемость: Благодаря разделению слоев кода тестирование становится проще.

Заключение:

Паттерн Mapper является полезным инструментом для работы с данными в приложениях. Он помогает упростить код, повысить его читаемость и поддерживаемость, а также обеспечивает гибкость при изменении структуры базы данных.