**JPQL** (Java Persistence Query Language) – это язык запросов персистентности Java, определенный в спецификации JPA. Он используется для создания запросов к сущностям для хранения в реляционной базе данных.

**JPQL** разработан на основе синтаксиса **SQL**. JPQL может извлекать информацию или данные, используя предложение SELECT, может выполнять массовые обновления, используя предложение UPDATE и предложение DELETE. SQL работает непосредственно против таблиц, записей и полей реляционной базы данных, тогда как JPQL работает с классами и экземплярами Java.

**nativeQuery** - это атрибут аннотации @Query в Spring Data JPA, который позволяет выполнять нативные SQL запросы напрямую к базе данных.

//JPQL

@Query("SELECT ch FROM ConversionHistory ch WHERE ch.user.id = :userId")

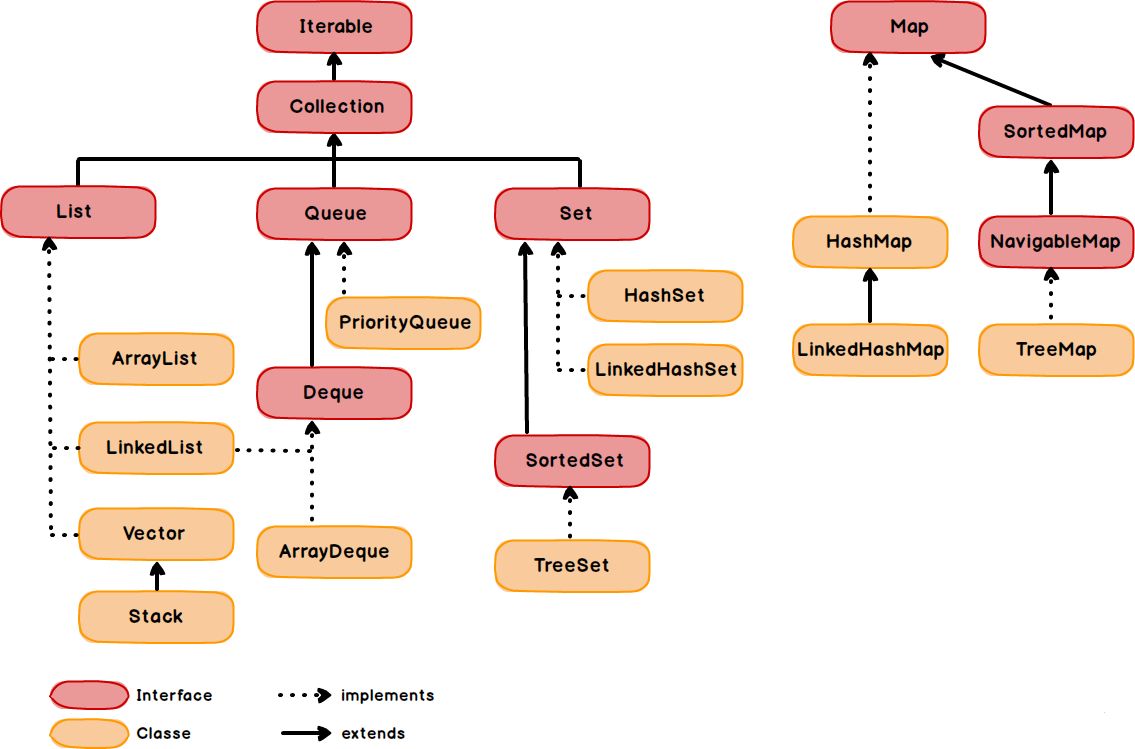
//native SQL

@Query(

value = "SELECT \* FROM conversion\_history ch WHERE ch.user\_id =:userId", nativeQuery = true

)

**Коллекции в Java**



**Класс java.util.Collection** является интерфейсом, определяющим базовый функционал для всех коллекций в Java. Он содержит основные методы для работы с коллекциями, такие как добавление, удаление и проверка наличия элемента, а также методы для получения размера коллекции и ее итерации.

**Класс java.util.Collections**, с другой стороны, является утилитарным классом, предоставляющим статические методы для работы с коллекциями. Он содержит методы для создания неизменяемых коллекций, синхронизации доступа к коллекции и сортировки элементов коллекции.

Таким образом, разница между двумя классами заключается в том, что Collection - это интерфейс, который определяет базовый функционал для всех коллекций в Java, а Collections - это утилитарный класс, который предоставляет набор статических методов для работы с коллекциями.

Все классы наследуются от Object. Это значит, что любой класс содержит методы, которые определены в классе Object. Методы **.equals()** и **.hashcode()** - одни из них.

**Главные правила для любых реализаций этих двух методов,** которые **нужно обязательно соблюдать**, запомнить как аксиому:

1). Если **x.equals(y) == true**, то обязательно **hashcode(x) == hashcode(y)**

2) Если **hashcode(x) == hashcode(y)**, то не обязательно **x.equals(y) == true**

Почему надо перегружать сразу два класса, если тебе нужен только equals()? Для сохранения логики hashcode.

**Кэш (cache)** - это временное хранилище данных, которое содержит копии информации, которые могут быть доступны более быстро и эффективно, чем оригинальные данные. Цель кэширования - уменьшить время доступа к данным и снизить нагрузку на источник данных.

В рамках программирования веб-приложений, кэш может использоваться для хранения результатов запросов к базе данных, ответов на веб-запросы, предварительно вычисленных значений и т. д. При запросе данных, сначала происходит поиск в кэше, и если данные уже есть, они возвращаются непосредственно из кэша без обращения к источнику данных, что ускоряет обработку запроса.

**Пагинация (pagination)** - это процесс разбиения большого объема данных на отдельные страницы для удобства навигации пользователем. Веб-страницы часто имеют ограниченный размер, и если данные превышают этот размер, то их необходимо разбить на несколько страниц. Пагинация позволяет пользователям просматривать данные постранично, что делает навигацию более удобной и эффективной.

**Пример пагинации:**

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.repository.PagingAndSortingRepository;

public interface UserRepository extends PagingAndSortingRepository<User, Long> {

Page<User> findAll(Pageable pageable);

}

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class UserController {

@Autowired

private UserRepository userRepository;

@GetMapping("/users")

public Page<User> getUsers(@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size) {

return userRepository.findAll(PageRequest.of(page, size));

}

}