Билет 28

1. Многопоточность. Интерфейсы Callable, Runnable, Future. Класс Thread.

2. HikariCP – что это и для чего используется. Какие есть альтернативы. Как работать в Java с HikariCP.

3. Реализовать endpoint в TaskController, которому при запросе передаётся boostrapServers и название топика. В endpoint создаётся consumer Kafka, который читает топик, начиная с самых старых сообщений. Далее в Redis находим строку, по ключу первого сообщения из Kafka ("test") и возвращаем в endpoint значение из Redis, но без каждого второго символа

Многопоточность. Интерфейсы Callable, Runnable, Future. Класс Thread.

Многопоточность в Java — это способность программы выполнять несколько потоков одновременно. Поток (thread) — это легковесный процесс, выполняющий инструкции параллельно с другими потоками. Java предоставляет встроенные средства для создания и управления потоками.

Класс Thread

Thread — основной класс для работы с потоками, представляет отдельный поток исполнения. Поток можно создать двумя способами:

1. Наследовать класс Thread и переопределить метод run():

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Hello from thread!");
    }
}
```

2. Создать объект Thread, передав ему реализацию интерфейса Runnable:

```
Runnable task = () -> System.out.println("Runnable task");
Thread t = new Thread(task);
t.start();
```

Запуск потока осуществляется методом start(), который вызывает метод run() в новом потоке. Прямой вызов run() выполнит код в текущем потоке, а не создаст новый.

Интерфейс Runnable

Runnable — функциональный интерфейс, содержащий один метод void run(). Используется для задания задачи, которую должен выполнять поток.

```
Runnable task = () -> {
    // код, выполняемый в потоке
};
```

```
new Thread(task).start();
```

Runnable не возвращает результат и не может выбрасывать проверяемые исключения.

Интерфейс Callable<V>

Callable — более мощный интерфейс, введённый в Java 5, предназначен для выполнения задач с возвращаемым результатом и возможностью выбрасывать исключения.

```
Callable<Integer> task = () -> {
    return 42;
};
```

Задачи Callable запускаются с использованием ExecutorService и возвращают объект типа Future.

Интерфейс Future<V>

Future — интерфейс, представляющий результат асинхронной операции. Позволяет:

- получить результат методом get(),
- проверить готовность isDone(),
- отменить выполнение cancel().

```
ExecutorService executor = Executors.newSingleThreadExecutor();
Callable<Integer> task = () -> 2 + 2;
Future<Integer> future = executor.submit(task);

int result = future.get(); // ожидает завершения задачи и получает
результат
executor.shutdown();
```

HikariCP – что это и для чего используется. Какие есть альтернативы. Как работать в Java с HikariCP.

HikariCP — это высокопроизводительный и лёгкий пул соединений с базой данных для Java-приложений. Его задача — эффективно управлять набором готовых соединений с базой, чтобы приложение могло быстро брать готовое соединение без затрат на установку нового. Это значительно улучшает производительность приложений, работающих с базами данных, особенно в условиях высокой нагрузки.

Основное назначение HikariCP — обеспечить быстрый и стабильный доступ к базе данных, минимизируя время ожидания и расходы на создание и закрытие соединений. Он отличается простотой настройки, низкой задержкой и высокой пропускной способностью.

Среди альтернатив HikariCP часто упоминаются пул соединений Apache Commons DBCP, C3P0 и Tomcat JDBC Connection Pool. По сравнению с ними HikariCP обычно показывает лучшие результаты по скорости и стабильности.

Для работы с HikariCP в Java нужно добавить соответствующую зависимость в проект (например, через Maven или Gradle), создать и настроить объект HikariDataSource, указав

параметры подключения (URL, логин, пароль, максимальное количество соединений и др.). Затем этот источник данных используется так же, как обычный DataSource: из него берутся соединения для работы с базой через JDBC.

Пример настройки

```
HikariConfig config = new HikariConfig();
config.setJdbcUrl("jdbc:postgresql://localhost:5432/mydb");
config.setUsername("user");
config.setPassword("password");
config.setMaximumPoolSize(10);

HikariDataSource dataSource = new HikariDataSource(config);

try (Connection conn = dataSource.getConnection()) {
    // Работа с базой через conn
}
```