**Билет №13**

**1. Java Memory Model (JMM) - основные аспекты. Garbage Collector. Garbage Collector G1. CMS.**

**Java Memory Model** - это абстракция, формальная спецификация, как потоки взаимодействуют через общую память. Её основная цель - определить, когда изменения данных, сделанные одним потоком, становятся видимыми для другого потока. Это необходимо, потому что в многопоточных приложениях из-за оптимизаций процессора поведение программы может быть неочевидным.

**Ключевые аспекты JMM:**

1. **Happens-Before:** Это фундаментальное отношение в JMM. Если действие A *«происходит-до»* действия B, то все результаты A гарантированно видны для B. JMM определяет набор правил для этих отношений:

**Правило монитора:** Разблокировка монитора *«происходит-до»* последующей блокировки того же монитора

**Правило volatile:** Запись в volatile-переменную *«происходит-до»* последующего чтения той же переменной

**Правило запуска и завершения потока:** Вызов thread.start() *«происходит-до»* любого действия в запущенном потоке. Любое действие в потоке *«происходит-до»* того, как другой поток успешно завершит ожидание на thread.join()

**Правило транзитивности:** Если A *«происходит-до»* B, а B *«происходит-до»* C, то A *«происходит-до»* C.

1. **volatile переменные:** Ключевое слово volatile обеспечивает видимость. Когда поток записывает значение в volatile-переменную, это значение становится видимым для всех других потоков.
2. **Синхронизация:** Использование синхронизации создает отношения *«происходит-до»*, что гарантирует, что изменения, сделанные в синхронизированном блоке одним потоком, будут видны следующему потоку, который войдет в синхронизированный блок по тому же монитору.

**Garbage Collector** - это механизм в составе JVM, который автоматически управляет памятью. Его основная задача освобождать память, занятую объектами, которые больше не используются приложением.

**Принцип работы:** Объект считается "живым", если на него существует цепочка ссылок, начинающаяся от "корня". Корни - это ссылки в стеке потоков, статические переменные и т.д. Если на объект нельзя добраться ни из одного корня, он считается мусором и может быть удален.

**G1 (Garbage-First)** - сборщик для серверных приложений. Его основная цель - замена CMS, обеспечивающая высокую производительность с предсказуемыми паузами.

**Ключевые особенности G1:**

**Разделение кучи на регионы:** В отличие от старых сборщиков, G1 делит кучу на множество регионов фиксированного размера (например, 1, 2, 4, 8, 16, 32 МБ). Это позволяет ему управлять памятью более гибко.

**Сборка на основе прогнозирования пауз:** G1 оценивает, сколько регионов можно очистить за заданный временной интервал паузы

CMS был разработан как сборщик для приложений, чувствительных к задержкам, с целью минимизировать время пауз.

**Ключевые фазы CMS:**

1. **Initial Mark (Начальная пометка):** Короткая пауза Stop-The-World. Помечаются объекты, достижимые напрямую из корней GC.
2. **Concurrent Mark (Параллельная пометка):** Длинная фаза, выполняемая параллельно с работой приложения. Помечаются все живые объекты, достижимые из начально помеченных.
3. **Remark (Перепометка):** Короткая пауза Stop-The-World. Нужна для пометки объектов, которые изменились во время фазы Concurrent Mark.
4. **Concurrent Sweep (Параллельная очистка):** Фаза, выполняемая параллельно с приложением, в которой память, занятая мусором, освобождается.

**2. Gradle – что это и для чего используется. Рассказать про разные области видимости в dependency, основные таски – clean, build, jar и другие.**

**Gradle** - это система автоматизации сборки, написанная на Java и Kotlin. Она используется для управления жизненным циклом проекта: компиляции исходного кода, упаковки в архив (JAR, WAR), управления зависимостями, запуска тестов, развертывания и т.д..

**Ключевые особенности:**

**Основан на Kotlin:** В отличие от Apache Maven, который использует XML, Gradle использует языки программирования (Kotlin) для написания скриптов сборки (build.gradle или build.gradle.kts). Это делает скрипты более гибким и лаконичными.

**последовательность:** Gradle старается выполнять только ту работу, которая необходима. Он отслеживает, какие задачи уже выполнены и какие входные данные изменились, и перезапускает только те задачи, которые зависят от этих изменений. Это значительно ускоряет сборку.

**Поддержка многомодульных проектов:** Отлично справляется с большими проектами, разбитыми на множество модулей.

**Обширная экосистема плагинов:** Существуют плагины для сборки Java и многих других технологий.

Область видимости (конфигурация) зависимости определяет, на каком этапе жизненного цикла проекта эта зависимость нужна. Основные конфигурации в плагине java:

1. implementation : Зависимость необходима для компиляции основного кода, но не передается транзитивно вашей библиотеки. Это означает, что если проект A зависит от библиотеки X с конфигурацией implementation, то проект B, который зависит от вашего проекта A, не увидит библиотеку X.
2. api : Зависимость необходима для компиляции и является частью API вашей библиотеки. Она передается транзитивно потребителям.
3. compileOnly : Зависимость требуется только во время компиляции, но не должна быть включена в runtime (например, в конечный JAR). Типичный пример - аннотации из библиотеки, которые нужны только компилятору.
4. runtimeOnly : Зависимость не нужна для компиляции, но необходима для выполнения программы. Например, драйвер базы данных JDBC.
5. testImplementation : Зависимость, используемая для компиляции и запуска тестов
6. testCompileOnly и testRuntimeOnly : Аналоги compileOnly и runtimeOnly, но для тестов.

Задача (Task) в Gradle — это атомарная операция сборки (компиляция, копирование файлов, создание JAR и т.д.).

Основные задачи, которые добавляет плагин java:

**clean:** Удаляет каталог build (и все его содержимое), в который Gradle помещает результаты своей работы. Это позволяет начать сборку "с чистого листа".

Команда: gradle clean

compileJava**:** Компилирует исходный Java-код из каталога src/main/java в классы, которые помещаются в build/classes/java/main.

**jar:** Берет скомпилированные классы и ресурсы и упаковывает их в JAR-архив, который по умолчанию создается в build/libs/.

Команда: gradle jar

compileTestJava**:** Компилирует тестовый Java-код из src/test/java. Зависит от classes, так как для компиляции тестов нужны основные классы.

test**:** Запускает модульные. Перед запуском тестов она автоматически выполняет все необходимые подготовительные задачи.

Команда: gradle test

**build:** Основная сборная задача. Выполняет полный цикл сборки: проверку, компиляцию, запуск тестов и упаковку в артефакт. Она зависит от check (что включает test) и assemble.

Команда: gradle build