Переменная представляет именованную область памяти, которая хранит значение определенного типа. Каждая переменная имеет тип, имя и значение. Тип определяет, какую информацию может хранить переменная или диапазон допустимых значений.

Переменные объявляются следующим образом:

|  |  |
| --- | --- |
| 1 | тип\_данных имя\_переменной; |

**инициализация**: присвоение значения переменной при ее объявлении, и выделение памяти.

Слово **var** ставится вместо типа данных, а сам тип переменной выводится из того значения, которое ей присваивается.

В отличие от переменных **константам** можно присвоить значение только один раз. Объявляются с помощью ключевого слова final.

Тип данных определяет диапазон значений, которые может хранить переменная или константа.

**boolean**: хранит значение true или false. Размер занимаемой памяти зависит от реализации jvm.

**byte**: хранит целое число от -128 до 127 и занимает 1 байт.

**short**: хранит целое число от -32768 до 32767 и занимает 2 байта.

**int**: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта

**long**: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт

**double**: хранит число с плавающей точкой от ±4.9\*10-324 до ±1.8\*10308 и занимает 8 байт

**float**: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта

**char**: хранит одиночный символ в кодировке UTF-16 и занимает 2 байта, поэтому диапазон хранимых значений от 0 до 65535. (Ведут себя как целочисленные значения, с ними можно выполнять арифметические операции)

Цифры в числовых литералах можно разделять знаком подчеркивания, для упрощения чтения.

**Область видимости** переменной, метода или класса – это их свойство быть доступными из тех или иных частей кода, т. е. различных областей программы.

Переменная, объявленная в теле метода, в цикле или условной конструкции, то она доступан только внутри этого блока кода. То же самое касается и аргументов метода.

Переменные объекта существуют только в период жизни этого объекта. Их область видимости регулируется модификаторами.

**Public –** класс, переменная или метод видна в любом месте программы (как в текущем пакете, так и за его пределами).

**Private –** к переменным и методам можно обращаться только внутри класса, в которым они объявлены.

**Package-private(по умолчанию) –** классы, поля и методы видны только внутри своего пакета.

**Protected –** поля и методы доступны в пределах пакета, а также в классах наследниках(которые могут находится в другом пакете).

**boxing –** автоматическое преобразование примитивного типа в эквивалентную ему класс-обертку.

Происходит при:

1. Присвоении примитивного типа переменной соответствующего класса обертки.
2. При передаче примитивного типа в параметр метода, ожидающего класс-обертку.

**Unboxing –** преобразование класса обертки в соответствующий примитивный тип.

Происходит:

1. При присвоении экземпляра класса обертки переменной соответствующего примитивного типа.
2. В выражениях, в которых один или оба аргумента являются экземплярами класса оберток (кроме == и !=).
3. При передаче объекта класса-обертки в метод, ожидающий соответствующий примитивный тип.

**Примитивные типы –** хранят значения.

**Ссылочные –** ссылку на объект.

Примитивы имеют значение по умолчанию — 0 для чисел, false для boolean,‘\u0000’ для char. Они присваиваются при инициализации пустого массива или в объекте по умолчанию.

Ссылочные типы  в языке Java включают:

* + Массивы
  + Классы
  + Интерфейсы
  + Перечисления

Ссылочные типы данных по умолчанию хранят null.

**Неизменяемые элементы**

**Иммутабельный** (неизменяемый, immutable) класс — это класс, который после инициализации не может изменить свое состояние. То есть если в коде есть ссылка на экземпляр иммутабельного класса, то любые изменения в нем приводят к созданию нового экземпляра.

Чтобы класс был иммутабельным, он должен соответствовать следующим требованиям:

* Должен быть объявлен как final, чтобы от него нельзя было наследоваться. Иначе дочерние классы могут нарушить иммутабельность.
* Все поля класса должны быть приватными в соответствии с принципами инкапсуляции.
* Для корректного создания экземпляра в нем должны быть параметризованные конструкторы, через которые осуществляется первоначальная инициализация полей класса.
* Для исключения возможности изменения состояния после инстанцирования, в классе не должно быть сеттеров.
* Для полей-коллекций необходимо делать глубокие копии, чтобы гарантировать их неизменность.

**Иммутабельность строк** дает следующие преимущества:

• Строки потокобезопасны.

• Для строк можно использовать специальную область памяти, называемую "пул строк". Благодаря которой две разные переменные типа String с одинаковым значением будут указывать на одну и ту же область памяти.

• Строки отличный кандидат для ключей в коллекциях, поскольку они не могут быть изменены по ошибке.

• Класс String кэширует хэш-код, что улучшает производительность хеш-коллекций, использующих String.

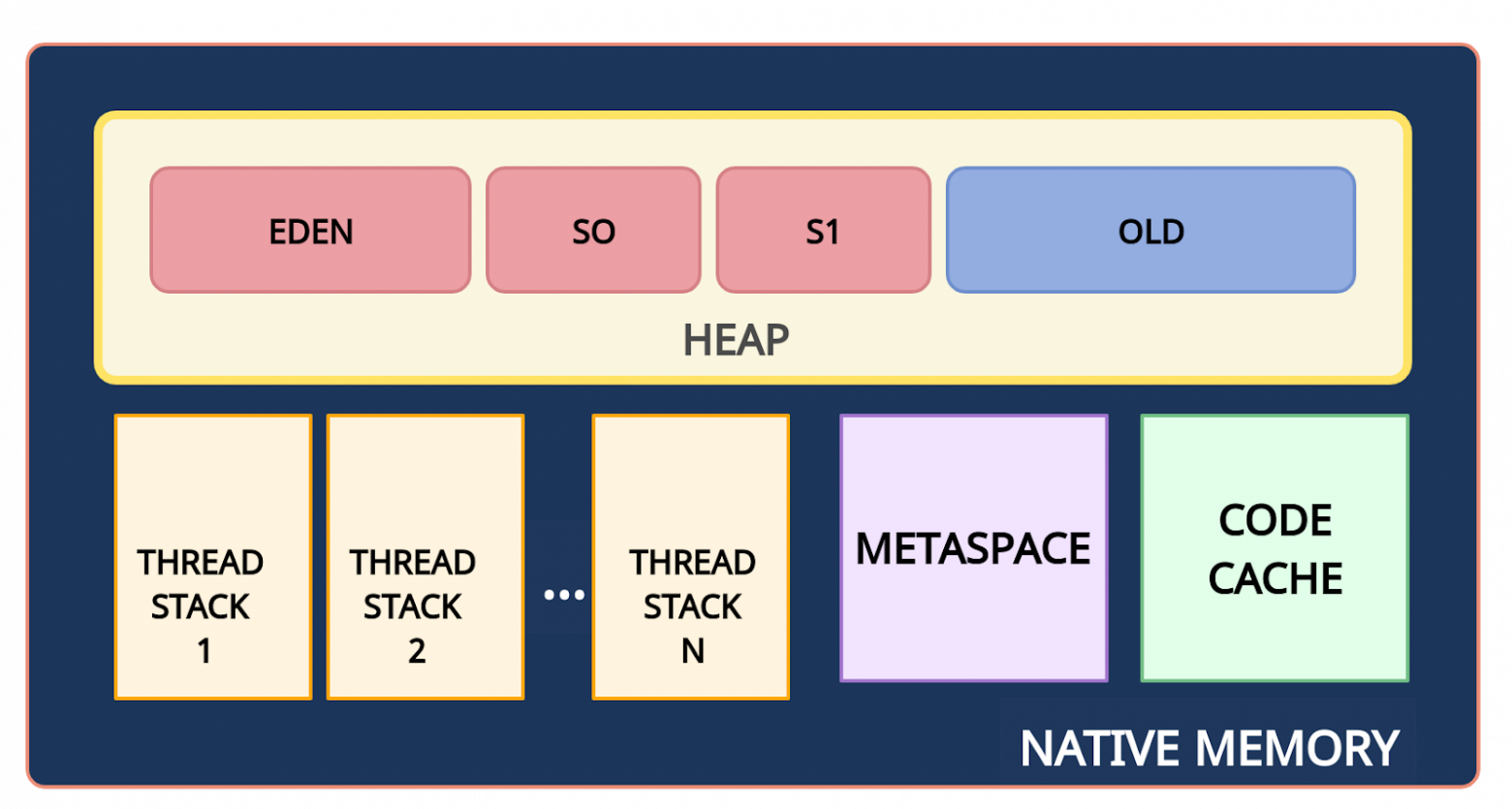
• Чувствительные данные, такие как имена пользователей и пароли, нельзя изменить по ошибке во время выполнения, даже при передаче ссылок на них между разными методами.

**Порядок действий при создании объектов**

1. Сначала для хранения объекта выделяется память.
2. Далее JVM создает ссылку на этот объект.
3. Инициализируются статические поля базового класса (Один раз во время загрузки класса). Потом выполняется статический блок инициализации.
4. Инициализируются статические поля класса потомка (Один раз во время загрузки класса). Потом выполняется статический блок инициализации.
5. Инициализируются нестатичексие переменные класса предка. Выполняется нестатический блок инициализации предка.
6. Выполняется конструктор базового класса.
7. Инициализируются нестатические поля класса потомка.

Выполняется нестатический блок инициализации потомка.

1. Вызывается конструктор класса потомка.



Native Memory — вся доступная системная память.

Heap (куча) — часть native memory, выделенная для кучи. Здесь JVM хранит объекты (В том числе и массивы). Это общее пространство для всех потоков приложения. Размер этой области памяти настраивается с помощью параметра -Xms (минимальный размер) и -Xmx (максимальный размер).

Stack (стек) — используется для хранения локальных переменных и стека вызовов метода. Для каждого потока выделяется свой стек.

Metaspace (метаданные) — в этой памяти хранятся метаданные классов (экземпляры класса Class<T>) и статические переменные. Это пространство также является общими для всех. Так как metaspace является частью native memory, то его размер зависит от платформы. Верхний предел объема памяти, используемой для metaspace, можно настроить с помощью флага MaxMetaspaceSize.

CodeCache (кэш кода) — JIT-компилятор компилирует часто исполняемый код, преобразует его в нативный машинный код и кеширует для более быстрого выполнения. Это тоже часть native memory.

В Java используются разные типы ссылок: сильные, слабые, мягкие и фантомные ссылки.

---------------------------------------------------------------------------------------------------

Любой объект что имеет strong ссылку запрещен для удаления сборщиком мусора. Разумеется что это объекты которые нужны Java программе.

мягкая(soft) удаляетс ясборщиком мусора когда возникает нехватка памяти.

Weak ссылка удаляется сборщиком мусора. Она используется в WeakHashMap: когда не остается strong ссылок значение из мапа удаляется.

#### 4. Фантомная ссылка

Используется для планирования посмертных действий по очистке, поскольку мы точно знаем, что объекты больше не живы. Используется только с очередью ссылок, поскольку .get()метод таких ссылок всегда будет возвращаться null. Эти типы ссылок считаются предпочтительными для финализаторов.

Строки хранятся в пуле строк.

-------------------------------------------------------

HotSpot JVM — сборка мусора

Это дорогостоящий процесс.

Главная идея подхода Tracing (трассировка) состоит в утверждении, что живыми могут считаться только те объекты, до которых мы можем добраться из корневых точек (GC Root) и те объекты, которые доступны с живого объекта. Всё остальное - мусор.

Существует 4 типа корневых точки:

* Локальные переменные и параметры методов;
* Потоки;
* Статические переменные;
* Ссылки из JNI.

Mark (маркировка). На первом этапе GC сканирует все объекты и помечает живые (объекты, которые все еще используются). На этом шаге выполнение программы приостанавливается. Поэтому этот шаг также называется "Stop the World" .

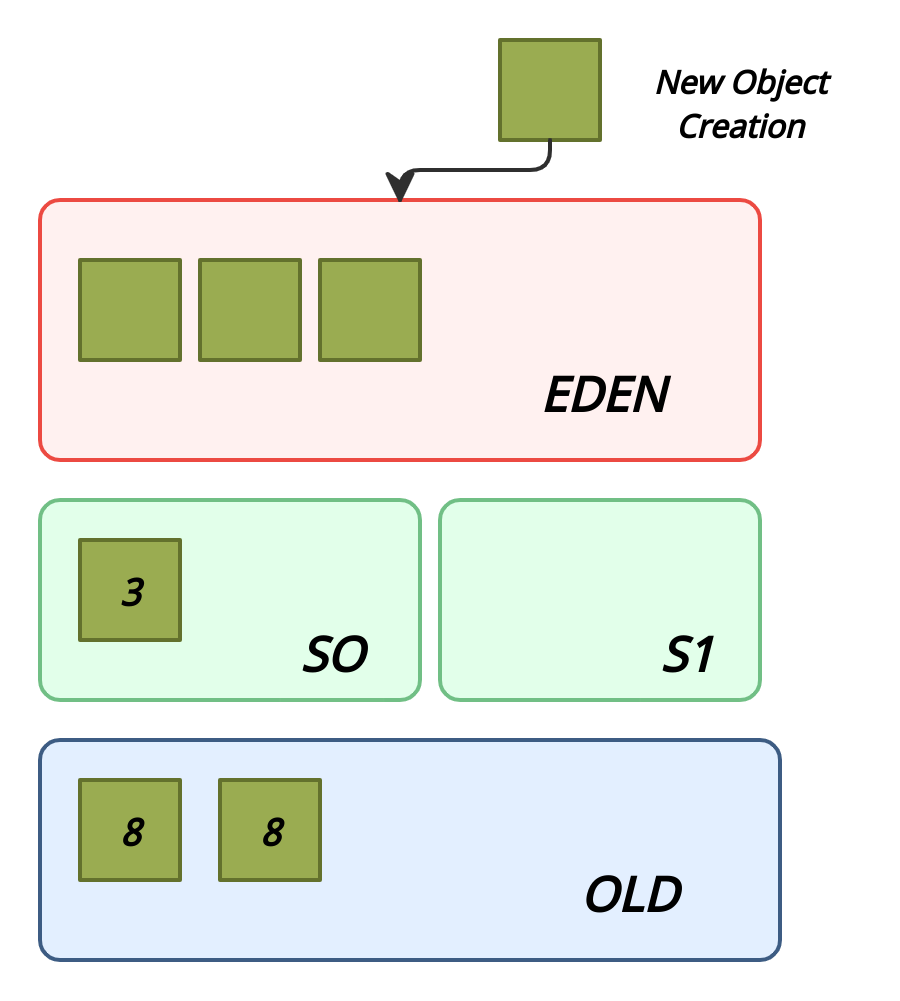
1. Sweep (очистка). На этом шаге освобождается память, занятая объектами, не отмеченными на предыдущем шаге.
2. Compact (уплотнение). Объекты, пережившие очистку, перемещаются в единый  непрерывный блок памяти. Это уменьшает фрагментацию кучи и позволяет проще и быстрее размещать новые объекты.

Для оптимизации сборки мусора память кучи дополнительно разделена на четыре области. В эти области объекты помещаются в зависимости от их возраста (как долго они используются в приложении).

* Young Generation (молодое поколение). Здесь создаются новые объекты. Область young generation разделена на три части раздела: Eden (Эдем), S0 и S1 (Survivor Space — область для выживших).
* Old Generation (старое поколение). Здесь хранятся давно живущие объекты.

Когда запускается этап mark, работа приложения останавливается. После завершения mark приложение возобновляет свою работу. Любая сборка мусора — это "Stop the World".

Гипотеза о поколениях говорит о следующем:

1. Большинство объектов живут недолго.
2. Если объект выживает, то он, скорее всего, будет жить вечно.
3. Этапы mark и sweep занимают меньше времени при большом количестве мусора. То есть маркировка будет происходить быстрее, если анализируемая область небольшая и в ней много мертвых объектов.
4. Новые объекты создаются в области Eden. Области Survivor (S0, S1) на данный момент пустые.
5. Когда область Eden заполняется, происходит минорная сборка мусора (Minor GC). Minor GC — это процесс, при котором операции mark и sweep выполняются для young generation (молодого поколения).
6. После Minor GC живые объекты перемещаются в одну из областей Survivor (например, S0). Мертвые объекты полностью удаляются.
7. По мере работы приложения пространство Eden заполняется новыми объектами. При очередном Minor GC области young generation и S0 очищаются. На этот раз выжившие объекты перемещаются в область S1, и их возраст увеличивается (отметка о том, что они пережили сборку мусора).
8. При следующем Minor GC процесс повторяется. Однако на этот раз области Survivor меняются местами. Живые объекты перемещаются в S0 и у них увеличивается возраст. Области Eden и S1 очищаются.
9. Объекты между областями Survivor копируются определенное количество раз (пока не переживут определенное количество Minor GC) или пока там достаточно места. Затем эти объекты копируются в область Old.
10. Major GC. При Major GC этапы mark и sweep выполняются для Old Generation. Major GC работает медленнее по сравнению с Minor GC, поскольку старое поколение в основном состоит из живых объектов.

**BigInteger и BigDecimal**

Примитивные типы имеют ограниченный диапазон значений. BigInteger и BigDecimal могут работать с числами любой точности. Являются immutable.

BigInteger является потомком класса Number. Содержит методы, дублирующие стандартные арифметические и побитовые операции Java, метод compareTo. Числа хранятся в двоичной форме с дополнительным кодом. Не приводят к переполнению или приведению по модулю, а увеличивают количество разрядов. Чтобы преобразовать обычное число используется статический метод valueOf().

BigDecimal хранит два целочисленных значения: мантиссу вещественного числа в виде объекта BigInteger и неотрицательный десятичный порядок числа типа int. Содержит методы, дублирующие математические операции, но методов побитовых операций нет.

Поскольку в этом классе можно работать с числами произвольной точности и явно указывать точность и вид округления, его используют для точных вычислений.

**Выполнение Java кода**

Сначала исходный код из файла .java компилируется в байт код компилятором javac. Этот байт код хранится в файле .class.

Интерпретатор jvm построчно преобразует байт-код в машинный код, и исполняет его. Однако чистая интерпретация байт-кода медленная. Ускорить интерпретацию байт-кода позволяет механизм (JITC) – Just-in-time compilation.

Если в программе присутствуют части кода, которые выполняются много раз, то их можно скомпилировать один раз в машинный код, чтобы в будущем ускорить выполнение. При следующем вызове этой части jvm будет сразу выполнять скомпилированный машинный код.