

**Set** — это неупорядоченное множество уникальных элементов.

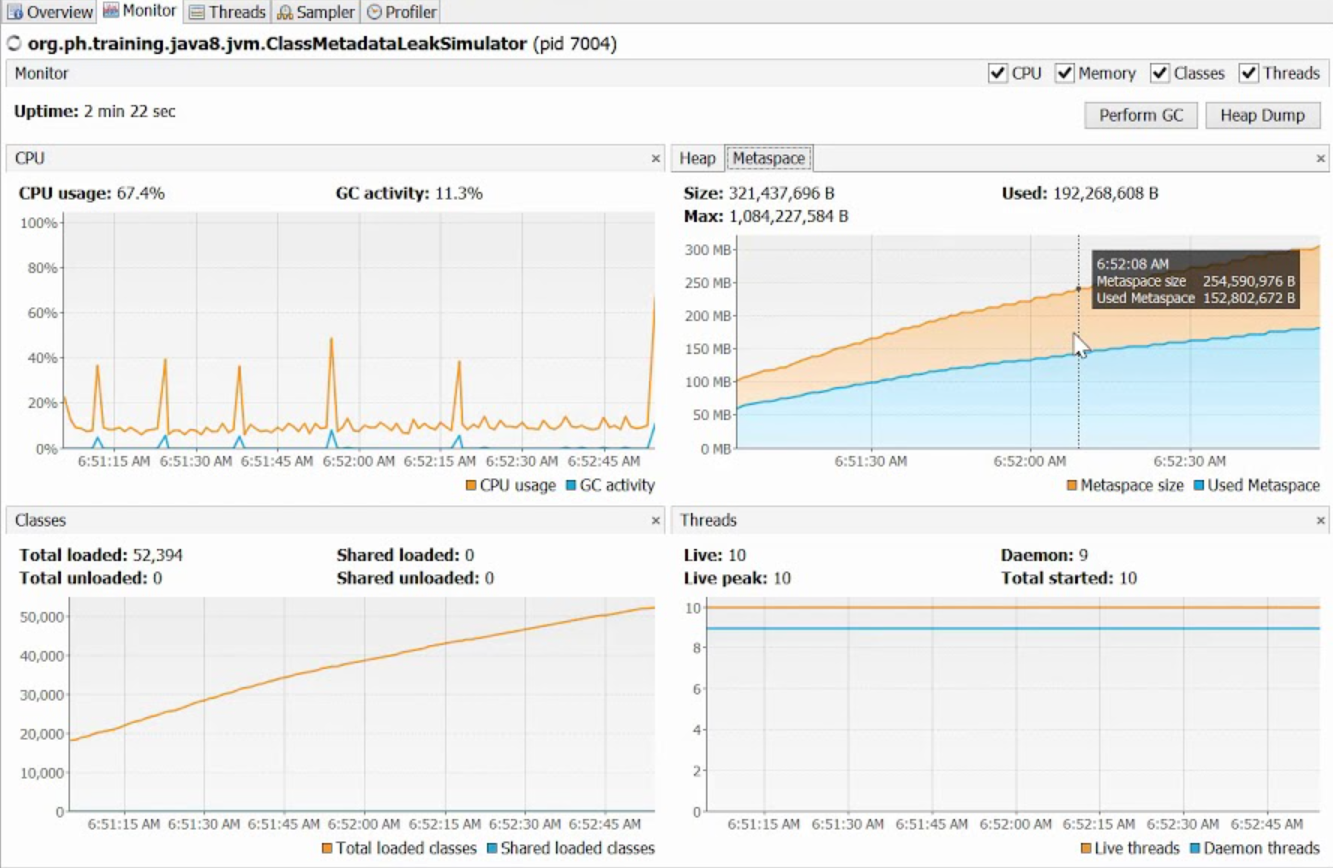
Например, мешочек с бочонками для игры в лото: каждый номер от 1 до 90 встречается в нём ровно один раз, и заранее неизвестно, в каком порядке бочонки вынут при игре.

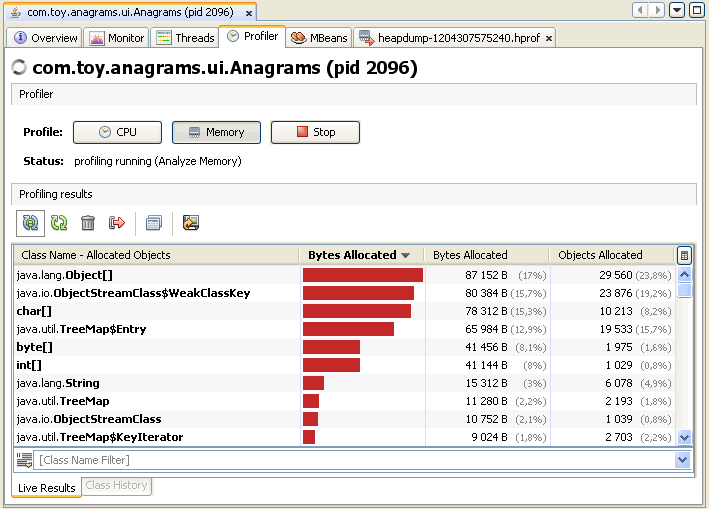
**List** — упорядоченный список, в котором у каждого элемента есть индекс. Дубликаты значений допускаются.

Например, последовательность букв в слове: буквы могут повторяться, при этом их порядок важен.

**Queue** — очередь. В таком списке элементы можно добавлять только в хвост, а удалять — только из начала. Так реализуется концепция **FIFO** (**first in, first out)** — «первым пришёл — первым ушёл». Вам обязательно напомнят это правило, если попробуете пролезть без очереди в магазине

**Map** <K, V> представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта Map. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта.





# Garbage Collector – сбощик мусора

# Модификаторы доступа

Public, private, protected(неизменяемый ,но вызываемый)

Static\no static

статическая переменная принадлежит самому классу, а нестатическая переменная принадлежит каждому экземпляру класса

# Spring Framework

По сути Spring Framework представляет собой просто [контейнер внедрения зависимостей](https://en.wikipedia.org/wiki/Dependency_injection), с несколькими удобными слоями (например: доступ к базе данных, прокси, аспектно-ориентированное программирование, RPC, веб-инфраструктура MVC). Это все позволяет вам быстрее и удобнее создавать Java-приложения.

Фраемворк для написания заглушек и бэкенда.

# Java метрики

**Память JVM – heap**

**Сборщики мусора**

* Heap – куча, вся managed-память, в которой хранятся все пользовательские объекты. Все следующие разделы – части кучи. Параметры -Xms, -Xmn и -Xmx устанавливают начальный, минимальный и максимальный размеры хипа соответственно.
* Eden, New Generation, Old Generation и другие – специфичные для сборщика мусора части кучи, поколения. Могут быть разные, но общий подход сохраняется: долго живущий объект постепенно двигается во всё более старое поколение; сборка мусора в разных поколениях происходит раздельно; чем поколение старше, тем сборка в нём реже, но и дороже.
* **Heap,цпу,оперативка**
* **Responds time**
* **Утилизация Memory**
* **Утилизация цпу**
* **Нагрузка системы**

# Отличие класса от абстрактного метода

Класс описывает переменные, свойства, процедуры и события объекта. Объекты являются экземплярами классов

Интерфейс описывает только поведение (методы), и у него нет полей

# отличие между абстрактным классом и интерфейсом

Основное отличие между абстрактным классом и интерфейсом в Java заключается в том, что абстрактный класс может содержать реализацию методов, в то время как интерфейс может содержать только объявления методов без какой-либо реализации.

Так же от интерфейса возможно массовое наследование,а с классом нет

Обращаться к классу такому

Extends\ implements

JVM (Java Virtual Machine) - жвм преобразует код джава в машинный код

# JVM чистка мусора

Что такое "мусор"? Мусором считается объект, который больше не может быть достигнут по ссылке из какого-либо объекта. Поскольку такие объекты больше не используются в приложении, то их можно удалить из памяти.

В Java сборка мусора происходит автоматически в течение всего времени работы программы

если после вызова объекта в коде он в коде больше не будет юзаться тогда делит

В поме прописываются зависимости

# Pom.xml

JVM состоит из трех отдельных компонентов:

#### загрузчик классов -

Когда вы компилируете исходный файл .java, он преобразуется в байт-код как файл .class. Когда вы вызываете этот класс у себя в программе, загрузчик классов загружает его в основную память.

* область памяти/данных среды выполнения;
* механизм выполнения.

#### Область кучи

Здесь хранятся все объекты и соответствующие им переменные экземпляра. Это область данных времени выполнения, из которой выделяется память для всех экземпляров классов и массивов.

### Система выполнения

Как только байт-код загружен в основную память, и подробная информация становится доступна в области данных среды выполнения, наступает следующий этап  —  запуск программы. Механизм выполнения делает это, выполняя код из каждого класса.

Однако перед выполнением программы байт-код необходимо преобразовать в инструкции машинного языка. В качестве механизма выполнения JVM может задействовать интерпретатор или JIT-компилятор.

Интерпретатор считывает и выполняет инструкции байт-кода строка за строкой.

JIT-компилятор преодолевает недостаток интерпретатора. Механизм выполнения сначала использует интерпретатор для выполнения байт-кода, но когда он находит какой-то повторяющийся код, то задействует JIT-компилятор.

Затем JIT-компилятор компилирует весь байт-код и изменяет его на собственный машинный код

Сборщик мусора (Garbage Collector, GC) собирает и удаляет объекты без ссылок из области кучи. Это процесс автоматического восстановления неиспользуемой памяти во время выполнения путем уничтожения мусорных объектов.

JVM содержит три различных типа сборщиков мусора.

* **Последовательная сборка мусора**. Это самая простая реализация GC. Она предназначена для небольших приложений, работающих в однопоточных средах.

**Параллельная сборка мусора Д**ля сборки мусора в нем используется несколько потоков, но работа приложения все равно приостанавливается при запуске.

**Garbage First (G1)**  Он разбивает кучу на набор областей одинакового размера и использует несколько потоков для их сканирования.

Нативные библиотеки методов  —  это библиотеки, написанные на других языках программирования

### Распространенные ошибки JVM

* **ClassNotFoundException**. Происходит, когда загрузчик классов пытается загрузить классы с помощью Class.forName(), ClassLoader.loadClass() или ClassLoader.findsystemclass(), но определение класса с указанным именем не найдено.
* **NoClassDefFoundError**. Происходит, когда компилятор успешно скомпилировал класс, но загрузчик классов не может найти файл класса во время выполнения.
* **OutOfMemoryError**. Происходит, когда JVM не может выделить объект из-за нехватки памяти, и сборщик мусора не может предоставить больше памяти.
* **StackOverflowError**. Происходит, если в JVM не хватает места при создании новых кадров стека во время обработки потока.

### **Стек (Stack)**

Стековая память отвечает за хранение ссылок на объекты кучи и за хранение типов значений

### **Куча (Heap)**

Эта часть памяти хранит в памяти фактические объекты, на которые ссылаются переменные из стека.

### **Типы ссылок**

#### **1. Сильная ссылка**

Это самые популярные ссылочные типы - Объект в куче не удаляется сборщиком мусора, пока на него указывает сильная ссылка или если он явно доступен через цепочку сильных ссылок.

#### **2. Слабая ссылка**

Попросту говоря, слабая ссылка на объект из кучи, скорее всего, не сохранится после следующего процесса сборки мусора

#### **3. Мягкая ссылка**

Эти типы ссылок используются для более чувствительных к памяти сценариев, поскольку они будут собираться сборщиком мусора только тогда, когда вашему приложению не хватает памяти. Следовательно, пока нет критической необходимости в освобождении некоторого места, сборщик мусора не будет касаться легко доступных объектов.

#### **4. Фантомная ссылка**

Используется для планирования посмертных действий по очистке, поскольку мы точно знаем, что объекты больше не живы.

### **Ссылки на String**

Ссылки на тип String в Java обрабатываются немного по- другому. Строки неизменяемы, что означает, что каждый раз, когда вы делаете что-то со строкой, в куче фактически создается другой объект. Для строк Java управляет пулом строк в памяти. Это означает, что Java сохраняет и повторно использует строки, когда это возможно.

### **Процесс сборки мусора**

Как обсуждалось ранее, в зависимости от типа ссылки, которую переменная из стека содержит на объект из кучи, в определенный момент времени этот объект становится подходящим для сборщика мусора