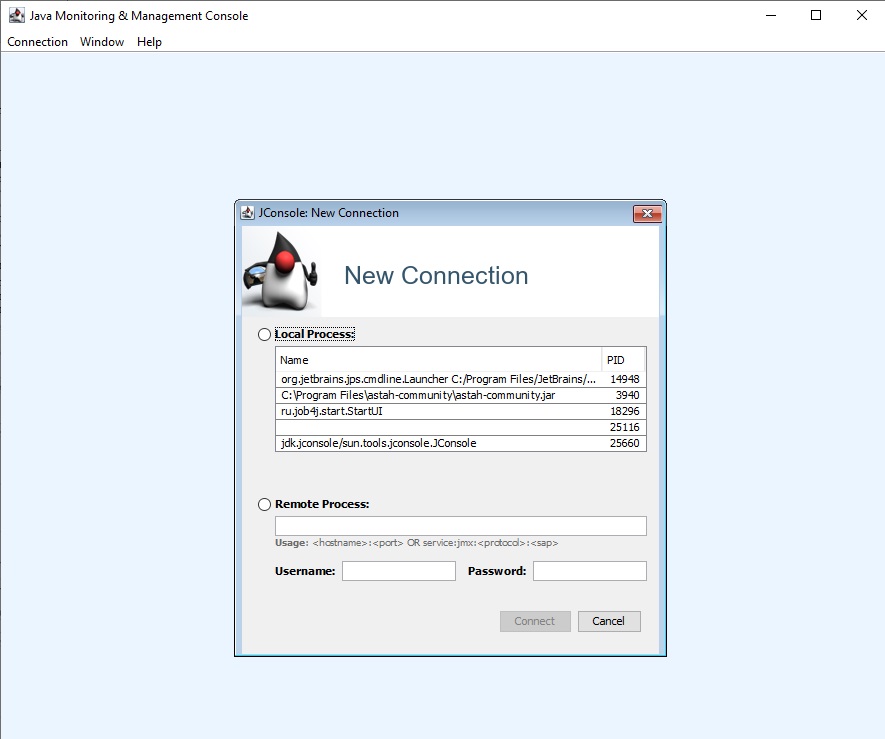
**Анализ работы Garbage Collection с использованием различных программ.**

## 1) JConsole

## JConsole - это инструмент мониторинга, который предоставляет информации о производительности и потреблении ресурсов приложений, работающего на платформе Java. Данная программа позволяет оценить объём используемой Heap Memory, позволяет увидеть количество живущих threads, количество загруженных классов и нагрузку CPU, а также даёт возможность выявления утечки памяти в режиме реального времени. Использование данного продукта позволяет самостоятельно отследить работу GC и анализировать эффективность его. Вызов программы осуществляется из командной строки командой “jconsole”

## Анализ работы JConsole представлен на основе работы программы из второго модуля, параметры virtual machine (VM) перед запуском приложения - -XX:+UseSerialGC -Xmx50m.

## Для того чтобы запустить JConsole необходимо запустить сначала приложение которое мы испытываем, после чего в командной строке вводим команду jconsole, далее экране на предлагается выбрать программу для мониторинга.



## Рисунок 1. – использование JConsole.

## Приложение для мониторинга может быть, как локальным, так и дистанционным.

## Для подключения к локальным процессам достаточно выбрать его в Local Process, а для подключения к дистанционным процессам необходимо будет вести localhost и логин с паролем.

## Приложение JConsole состоит из шести вкладок.

## Overview: отображает обзорную информацию о виртуальной машине Java и отслеживаемых значениях.

## Memory: отображает информацию об использовании памяти.

## Threads: отображает информацию об использовании потоков.

## Classes: отображает информацию о загрузке классов в приложении.

## VM Summary: отображает информацию о Java VM.

## MBeans: отображает информацию о MBeans.

## C:\Users\Asus\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.jpgВо вкладке Overview мы наблюдаем следующую картину:

## Рисунок 1. – использование JConsole.

## На первом графике Heap memory usage наблюдается резкое падение использование Heap. Тут можно сделать вывод, что на момент запуска модуля произошла очистка памяти за счёт вызова GC, что привело к высвобождению памяти.

## График Threads показывает какое количество потоков работаю в данный момент.

## На графике Classes показывается количество загруженных классов.

## На графике CPU Usage показывается использование центрального процесса для этого модуля.

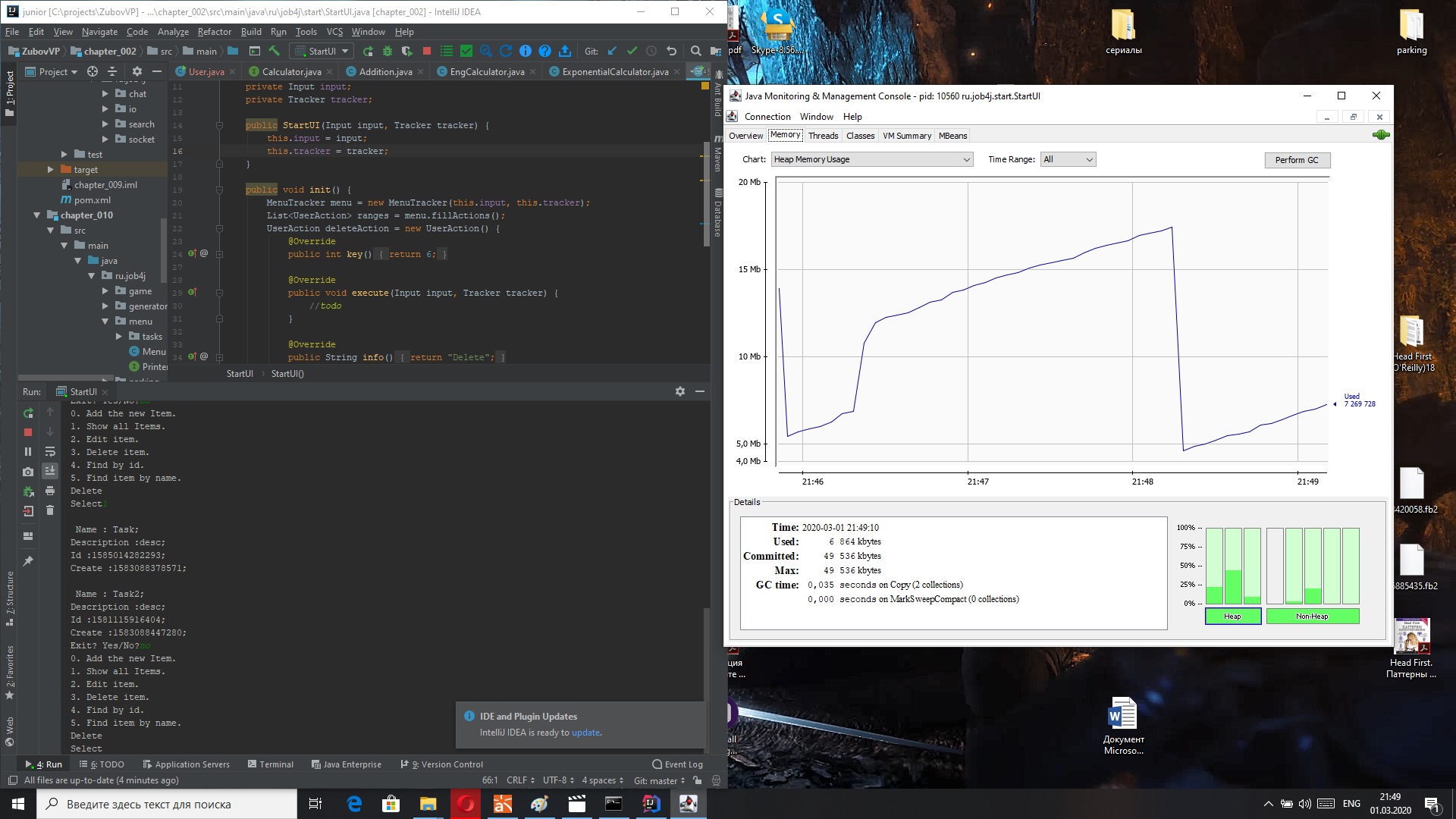
## В правом верхнем углу присутствует зелёный значок, который позволяет в любой момент отключаться от приложения и подключатся к нему повторно.

## C:\Users\Asus\AppData\Local\Microsoft\Windows\INetCache\Content.Word\5.jpgПосле того как наш модуль был загружен, мы начинаем добавлять заявки в программу, в результате мы наблюдаем рост используемого Heap на первом графике, увеличение количество загруженных классов в Heap и увеличения работы центрального процессора.

## Рисунок 2. – использование JConsole.

## На рисунке 2 мы наблюдает плавное увеличение использования Heap, а затем резкое падение, несмотря на то, что в параметры VM мы задали объем Heap равный 50 мб и наша программа могла продолжить заполнять Heap полностью, но VM решила запустить GC не дожидаясь полного заполнения Heap.

## Во вкладке Memory представлена более подробная информация по использованию Heap.

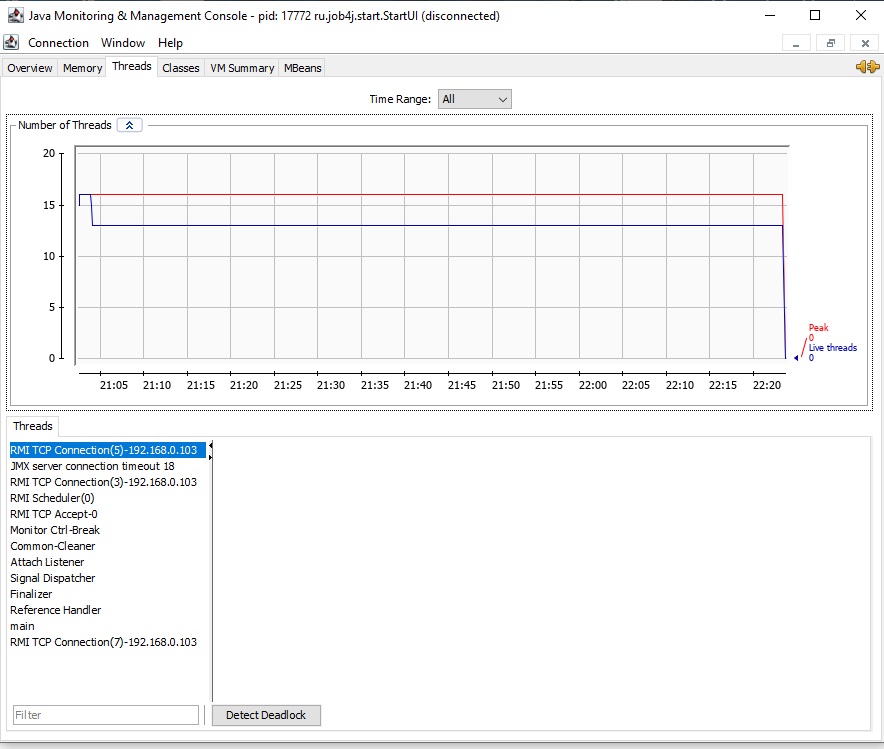


## Рисунок 3. – использование JConsole.

На вкладке Memory имеется кнопка «Perform GC», которая принудительно запускает GC. Диаграмма показывает использование памяти виртуальной машиной Java с течением времени для heap и non-neap, а также для определенных пулов памяти. Доступные пулы памяти зависят от того, какая версия Java VM используется. Для Java VM HotSpot пулы памяти следующие:

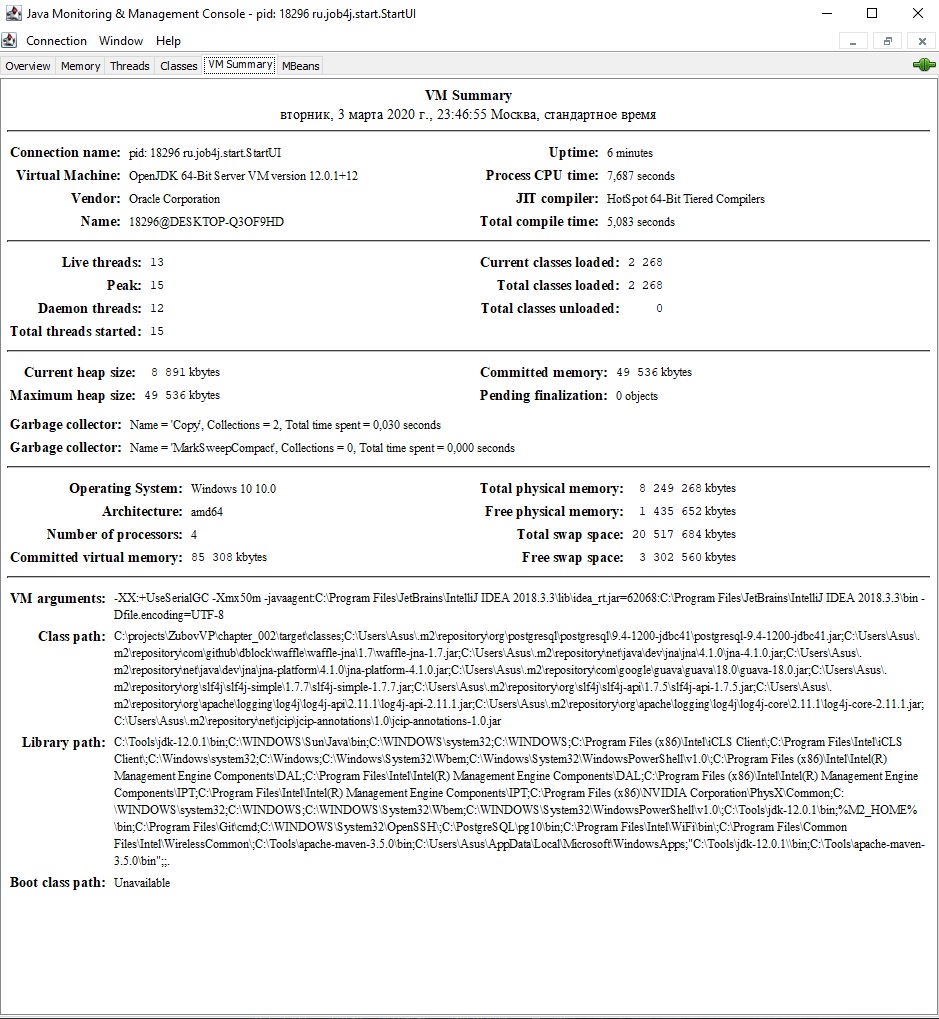
* *Eden Space (куча)* - пул, из которого память первоначально выделена для большинства объектов.
* Survior - пул, содержащий объекты, которые выжили после сбора мусора в пространстве eden.
* *Tenured Generation (heap)* - пул, содержащий объекты, которые существовали в пространстве survior в течение некоторого времени.
* Permanent Generation (non-heap) - пул, содержащий все отражающие данные самой виртуальной машины, такие как объекты классы и методы.
* Code Cache (non-heap) - Java VM HotSpot также включает кэш кода, содержащий память, которая используется для компиляции и хранения собственного кода.

## Раздел Threads позволяет оценивать количество живущих потоков.



## Рисунок 4. – использование JConsole.

## Во вкладке VM Summary показывается общая информация.



## Рисунок 5. – использование JConsole.

Информация, представленная на этой вкладке, включает следующее.

* **Summary**
  + *Время работы*: общее время с момента запуска виртуальной машины Java.
  + *Process CPU Time*: общее количество процессорного времени, которое виртуальная машина Java использовала с момента своего запуска.
  + *Общее время компиляции*: Общее время, потраченное на компиляцию. Java VM определяет, когда происходит компиляция.
* **Threads**
  + *Живые потоки*: текущее количество потоков живых демонов плюс потоки, не являющиеся демонами.
  + *Пик*: наибольшее количество активных потоков с момента запуска Java VM.
  + *Потоки демоны*: Текущее количество живых потоков демонов (фоновые потоки).
  + *Всего запущенных потоков*: Общее количество потоков, запущенных с момента запуска Java VM, включая потоки демоны, которые не завершили свою работу.
* **Classes**
  + *Загруженные текущие классы*: количество классов, загруженных в память в данный момент времени.
  + *Всего загруженных классов*: Общее количество классов, загруженных в память с момента запуска Java VM, включая те, которые впоследствии были выгружены.
  + *Всего выгруженных классов*: Количество классов, выгруженных из памяти с момента запуска Java VM.
* **Memory**
  + *Текущий размер кучи*: количество килобайт, занятых в данный момент кучей.
  + *Выделенная память*: общий объем памяти, выделенный для использования кучей.
  + *Максимальный размер кучи*: максимальное количество килобайт, которая может содержать куча.
  + *Объекты, ожидающие завершения*: количество объектов, ожидающих завершения.
  + *Сборщик мусора*: информация о сборке мусора, включая имена сборщиков мусора, количество выполненных сборок и общее время, затраченное на сборку мусора.
* **Operating System**
  + *Общая физическая память*: объем оперативной памяти (RAM), который имеется в операционной системе.
  + *Свободная физическая память*: объем свободной оперативной памяти, доступный операционной системе.
  + *Выделенная виртуальная память*: объем виртуальной памяти, гарантированно доступный для запущенного процесса.
* **Other Information**
  + *Аргументы виртуальной машины*: входные аргументы, передаваемые приложением в виртуальную машину Java, за исключением аргументов основного метода.
  + *Путь к* классу: Путь к классу, который используется системным загрузчиком классов для поиска файлов классов.
  + *Путь к библиотеке*: список путей для поиска при загрузке библиотек.
  + *Путь к классу загрузки*: Путь к классу загрузки используется загрузчиком классов начальной загрузки для поиска файлов классов.

## Вывод: использование JConsole позволяет анализировать работу запущенного приложения в режиме реального времени, тем самым можно обнаружить ситуации, в результате которых приложение работает с максимальной нагрузкой с целью оптимизации работы самого приложения.