**Лабораторная работа № 6.**

**Удаление объектов и сборка мусора.**

**Линия поведения сборщика мусора (утилизатора памяти)**

Java-программисту не нужно следить за распределением памяти, так как сборщик мусора управляет памятью автоматически. Сборщик мусора запускается виртуальной машиной Java (JVM). Сборщик мусора — это низкоприоритетный процесс, который запускается периодически и освобождает память, использованную объектами, которые больше не нужны.

Разные JVM имеют отличные друг от друга алгоритмы сбора мусора. Существует несколько используемых алгоритмов, например: алгоритм подсчёта ссылок или алгоритмы разметки и очистки.

**Запуск сборщика мусора**

JVM обычно запускает сборщик мусора при низком уровне свободной памяти. Но работа сборщика мусора не гарантирует, что всегда будет оставаться достаточно свободной памяти.

Если памяти недостаточно даже после восстановления, JVM генерирует исключение OutOfMemoryError. Обратите внимание, что перед генерированием исключения JVM обязательно запускает сборщик мусора как минимум 1 раз.

Вы можете запросить запуск сборщика мусора, но вы не можете принудительно задавать это действие.

**Запрос запуска сборщика мусора**

Для запроса вы можете вызвать один из следующих методов:

System.gc()

Runtime.getRuntime().gc()

**Пригодность для запуска сборщика мусора**

Объект подлежит утилизации, когда он недоступен живому потоку.

Объект может подлежать утилизации в разных случаях:

* Если переменная ссылочного типа, которая ссылается на объект, установлена в положение "0", объект подлежит утилизации, в том случае, если на него нет других ссылок.
* Если переменная ссылочного типа, которая ссылается на объект, создана для ссылки на другой объект, объект подлежит утилизации, в том случае, если на него нет других ссылок.
* Объекты, созданные локально в методе, подлежат утилизации, когда метод завершает работу, если только они не экспортируются из этого метода (т.е, возвращаются или генерируются как исключение).
* Объекты, которые ссылаются друг на друга, могут подлежать утилизации, если ни один из них не доступен живому потоку.

Рассмотрим пример:

public class TestGC {

public static void main(String [] args) {

Object o1 = new Integer(3); // Line 1

Object o2 = new String("Tutorial");// Line 2

o1 = o2; // Line 3

o2 = null; // Line 4

// Rest of the code here

}

}

В этом примере объект Integer (целочисленный), на который первоначально ссылается указатель o1 может подвергаться утилизации после строки 3, так как o1 теперь ссылается на объект String (строковый). Несмотря на то, что o2 создан для ссылки к нулю, объект String (строковый) не подлежит утилизации, так как o1 ссылается на него.

**Финализация**

Java-технология позволяет использовать метод finalize() (финализировать), чтобы произвести необходимую очистку перед тем, как сборщик мусора извлекает объект из памяти. Этот метод вызывается для объекта сборщиком мусора, когда сборщик мусора вычисляет, что ссылок к объекту больше нет. Это описано в классе Object, а значит, это наследуется всеми классами. Подкласс отменяет метод finalize(), чтобы освободиться от системных ресурсов или для ещё одной очистки:

protected void finalize() throws Throwable

Если незарегистрированное исключение генерируется методом finalize(), то исключение игнорируется и финализация этого объекта прекращается. Метод finalize() будет активизирован только один раз за время существования объекта. Возможно использование метода finalize() любого объекта, чтобы защитить его от утилизации. Но в этом случае сборщик мусора уже не активирует finalize() для этого объекта. Метод finalize() всегда будет активизирован один раз перед тем, как объект будет удалён сборщиком мусора. Однако, возможно, что метод finalize() не будет активизирован для данного объекта за всё время его существования, так как он может не подлежать утилизации.

Разбор задачи: *Сколько объектов будут подлежать утилизации?*

public class TutorialGC {

public static void main(String [] args) {

Object a = new Integer(100); // Line1

Object b = new Long(100); // Line2

Object c = new String("100"); // Line3

a = null; // Line4

a = c; // Line5

c = b; // Line6

b = a; // Line7

// Rest of the code here

}

}

Из трёх объектов, созданных в строках 1, 2 и 3, только объект Integer подлежит утилизации в конце строки 7. Переменная ссылки, a, которая первоначально ссылалась на объект Integer, ссылается на объект String в строке 5. Таким образом, Integer oбъект подлежит утилизации после строки 5, так как нет переменных, которые ссылаются на него. Переменные b и c ссылаются на объекты String и Long объекты в строках 6 и 7, поэтому они не подлежат утилизации.

**Задание**

Дополните программу из лабораторной работы №5 сборщиком мусора, предусмотрев, чтобы хотя бы один объект подлежал утилизации.