Память в JVM

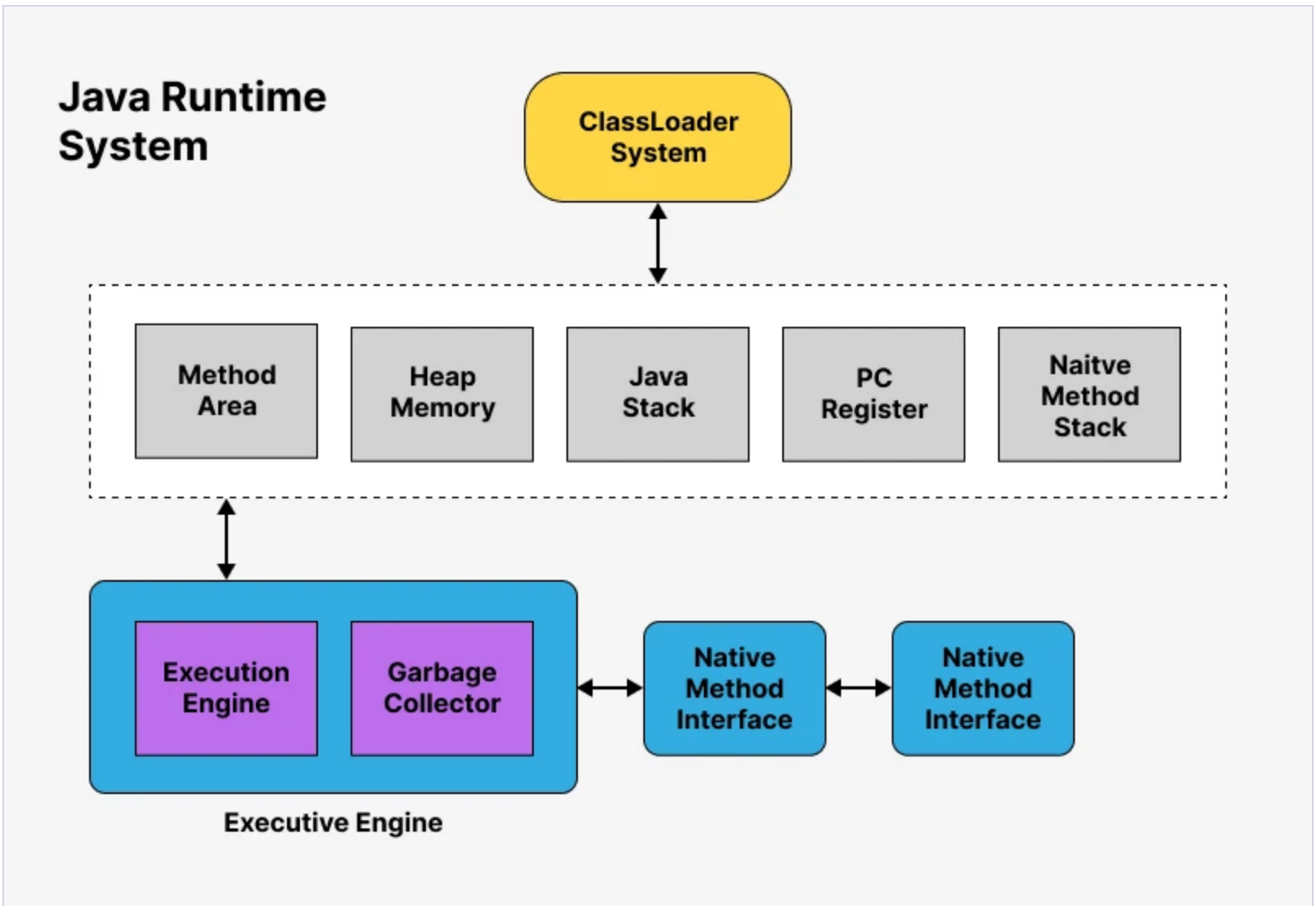
JSP & Servlets
18 уровень, 0 лекция

ОТКРЫТА

Знакомство с памятью в JVM

Как ты уже знаешь, JVM запускает Java-программы внутри себя. Как и любая виртуальная машина, она имеет собственную систему организации памяти.

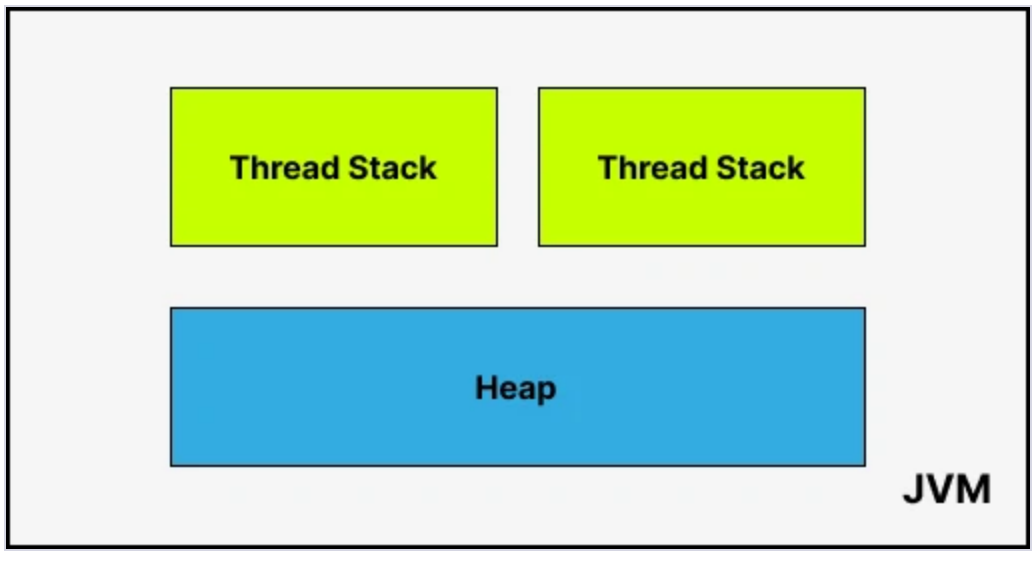
Схема организации внутренней памяти указывает на принцип работы вашего Java-приложения. Таким образом можно определить узкие места в работе приложений и алгоритмов. Давай разберемся, как она устроена.



Важно! Модель Java в первоначальном виде была недостаточно хороша, поэтому она была пересмотрена в Java 1.5. Эта версия используется по сей день (Java 14+).

Стек потока

Модель памяти в Java, используемая внутри JVM, делит память на стеки потоков (thread stacks) и кучу (heap). Посмотрим на Java-модель памяти, логично разделенную на блоки:



Все потоки, работающие в JVM, имеют **свой стек**. Стек в свою очередь держит информацию о том, какие методы вызвал поток. Я буду называть это “стеком вызовов”. Стек вызовов возобновляется, как только поток выполняет свой код.

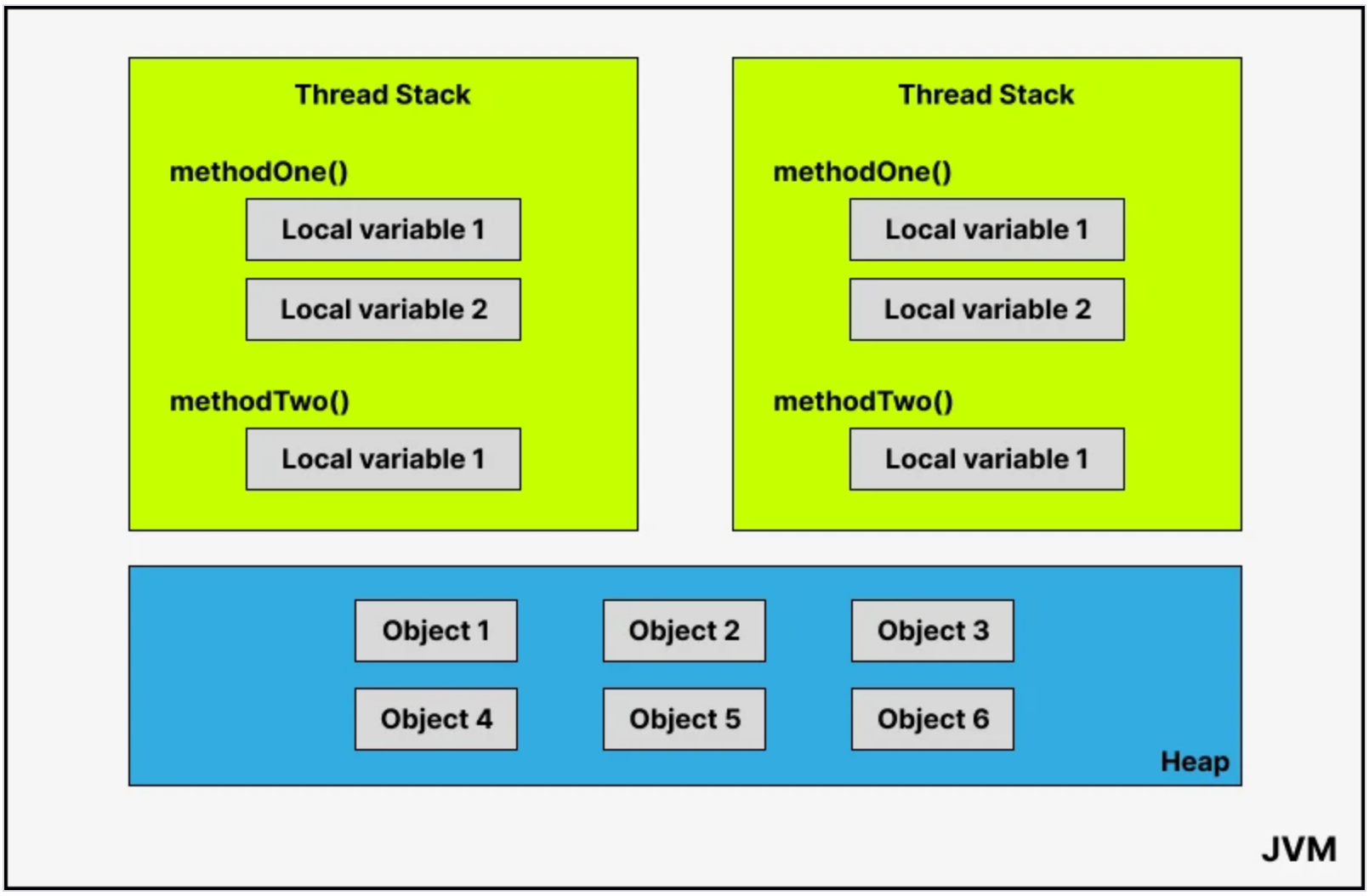
Стек потока содержит в себе **все локальные переменные**, требующиеся для выполнения методов из стека потока. Поток может получить доступ только к своему стеку. Локальные переменные не видны остальным потокам, только потоку, создавшему их. В ситуации, когда два потока выполняют один и тот же код, они оба создают свои локальные переменные. Таким образом, каждый поток имеет свою версию каждой локальной переменной.

Все локальные переменные примитивных типов (boolean, byte, short, char, int, long, float, double) полностью хранятся в стеке потоков и не видны другим потокам. Один поток может передать копию примитивной переменной другому потоку, но не может совместно использовать примитивную локальную переменную.

Куча (heap)

Куча содержит все объекты, созданные в вашем приложении, независимо от того, какой поток создал объект. К этому относятся и обертки примитивных типов (например, Byte, Integer, Long и так далее). Неважно, был ли объект создан и присвоен локальной переменной или создан как переменная-член другого объекта, он хранится в куче.

Ниже диаграмма, которая иллюстрирует стек вызовов и локальные переменные (они хранятся в стеках), а также объекты (они хранятся в куче):



В случае, когда локальная переменная примитивного типа, она хранится в стеке потока.

Локальная переменная также может быть ссылкой на объект. В этом случае ссылка (локальная переменная) хранится в стеке потоков, но сам объект хранится в куче.

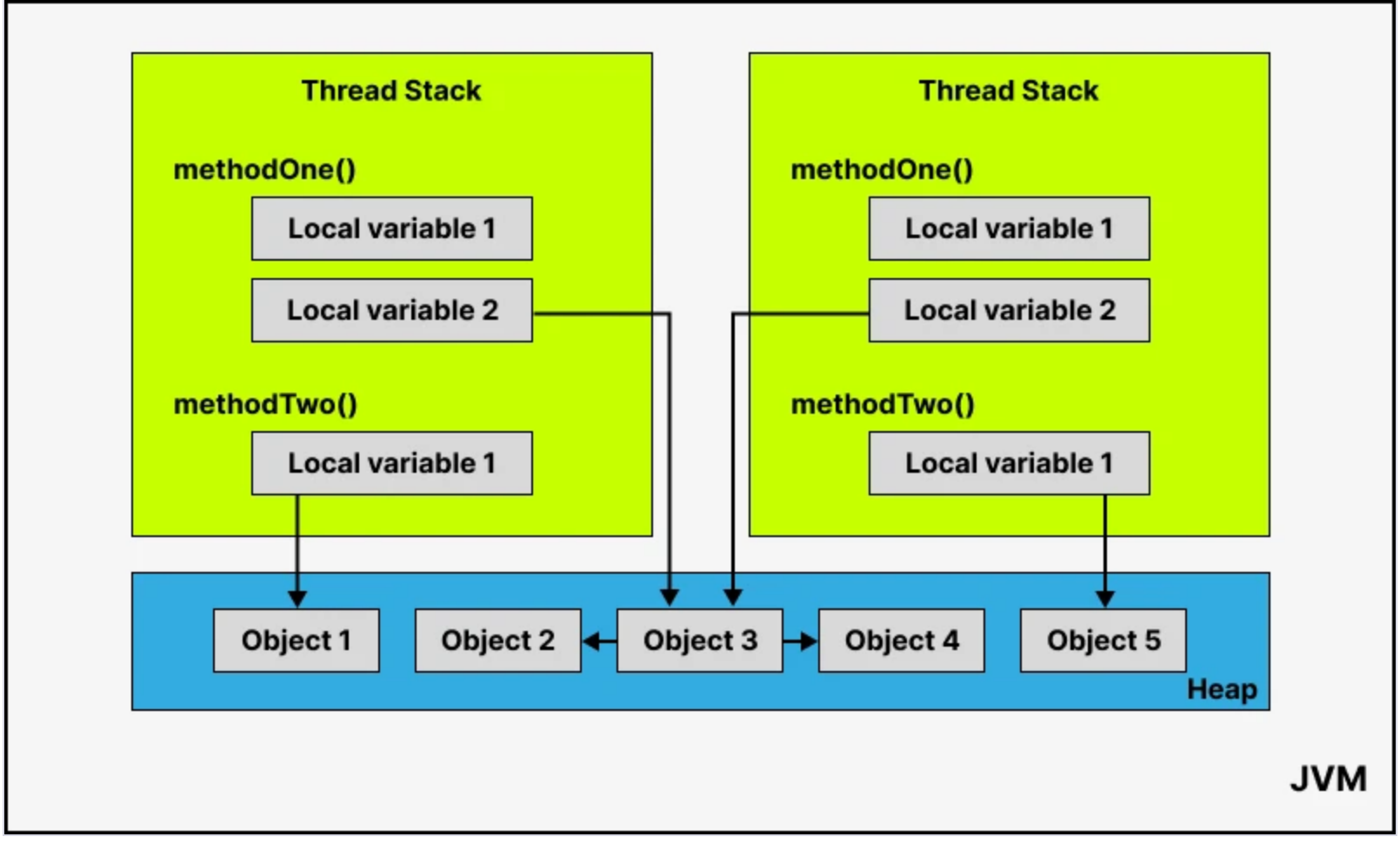
Объект содержит методы, эти методы содержат локальные переменные. Эти локальные переменные также хранятся в стеке потоков, даже если объект, которому принадлежит метод, хранится в куче.

Переменные-члены объекта хранятся в куче вместе с самим объектом. Это верно как в случае, когда переменная-член имеет примитивный тип, так и в том случае, если она является ссылкой на объект.

Статические переменные класса также хранятся в куче вместе с определением класса.

Взаимодействие с объектами

К объектам в куче могут обращаться все потоки, которые имеют ссылку на объект. Если поток имеет доступ к объекту, то он может получить доступ к переменным этого объекта. Если два потока вызывают метод для одного и того же объекта одновременно, они оба будут иметь доступ к переменным-членам объекта, но каждый поток будет иметь свою собственную копию локальных переменных.



Два потока имеют набор локальных переменных. `Local Variable 2` указывает на общий объект в куче (`Object 3`). Каждый из потоков имеет свою копию локальной переменной со своей ссылкой. Их ссылки являются локальными переменными и поэтому хранятся в стеках потоков. Тем не менее, две разные ссылки указывают на один и тот же объект в куче.

Обрати внимание, что общий `Object 3` имеет ссылки на `Object 2` и `Object 4` как переменные-члены (показано стрелками). Через эти ссылки два потока могут получить доступ к `Object 2` и `Object` 4.

На диаграмме также показана локальная переменная (`Local variable 1` из `methodTwo`). Каждая ее копия содержит разные ссылки, которые указывают на два разных объекта (`Object 1` и `Object 5`), а не на один и тот же. Теоретически оба потока могут обращаться как к `Object 1` , так и к `Object 5` , если они имеют ссылки на оба этих объекта. Но на диаграмме выше каждый поток имеет ссылку только на один из двух объектов.

Пример взаимодействия с объектами

Давай посмотрим, как мы можем продемонстрировать работу в коде:

```
public class MySomeRunnable implements Runnable() {

    public void run() {
        one();
    }

    public void one() {
        int localOne = 1;

        Shared localTwo = Shared.instance;

        //… делаем что-то с локальными переменными

        two();
    }

    public void two() {
        Integer localOne = 2;

        //… делаем что-то с локальными переменными
    }
}
```

```java
public class Shared {

    // храним инстанс на наш объект в переменной

    public static final Shared instance = new Shared();

    // переменные-члены, указывающие на два объекта в куче

    public Integer object2 = new Integer(22);
    public Integer object4 = new Integer(44);
}
```

Метод `run()` вызывает метод `one()`, а `one()` в свою очередь вызывает `two()`.

Метод `one()` объявляет примитивную локальную переменную (`localOne`) типа int и локальную переменную (`localTwo`), которая является ссылкой на объект.

Каждый поток, выполняющий метод `one()`, создаст свою собственную копию `localOne` и `localTwo` в своем стеке. Переменные `localOne` будут полностью отделены друг от друга, находясь в стеке каждого потока. Один поток не может видеть, какие изменения вносит другой поток в свою копию `localOne` .

Каждый поток, выполняющий метод `one()`, также создает свою собственную копию `localTwo` . Однако две разные копии `localTwo` в конечном итоге указывают на один и тот же объект в куче. Дело в том, что `localTwo` указывает на объект, на который ссылается статическая переменная `instance` . Существует только одна копия статической переменной, и эта копия хранится в куче.

Таким образом, обе копии `localTwo` в конечном итоге указывают на один и тот же экземпляр `Shared`. Экземпляр `Shared` также хранится в куче. Он соответствует `Object 3` на диаграмме выше.

Обрати внимание, что класс `Shared` также содержит две переменные-члены. Сами переменные-члены хранятся в куче вместе с объектом. Две переменные-члены указывают на два других объекта `Integer` . Эти целочисленные объекты соответствуют `Object 2` и `Object 4` на диаграмме.

Также обрати внимание, что метод `two()` создает локальную переменную с именем `localOne` . Эта локальная переменная является ссылкой на объект типа Integer. Метод устанавливает ссылку `localOne` для указания на новый экземпляр Integer. Ссылка будет храниться в своей копии `localOne` для каждого потока. Два экземпляра Integer будут сохранены в куче и, поскольку метод создает новый объект Integer при каждом выполнении, два потока, выполняющие этот метод, будут создавать отдельные экземпляры Integer. Они соответствуют `Object 1` и `Object 5` на диаграмме выше.

Обрати также внимание на две переменные-члены в классе `Shared` типа Integer, который является примитивным типом. Поскольку эти переменные являются переменными-членами, они все еще хранятся в куче вместе с объектом. В стеке потоков хранятся только локальные переменные.

‹ Предыдущая лекция

Следующая лекция ›

Oleg Уровень 41 9 ноября 2022, 12:23

Последний абзац что-то типа повторения одного из предыдущих: "Обрати также внимание на две переменные-члены в классе Shared типа Integer, который является примитивным типом. Поскольку эти переменные являются переменными-членами, они все еще хранятся в куче вместе с объектом. В стеке потоков хранятся только локальные переменные"
При этом в нём есть ошибки: Integer не примитив, а объект.

Ответить +3