Java Memory Management

with Rustam Khakov

JMM (Java Memory Management)

- оптимизация работы
- избегание ошибок в коде

- где хранятся данные
- как происходит работа
- как происходит доступ к данным в многопоточной среде
- как происходит выделение и подчищение памяти

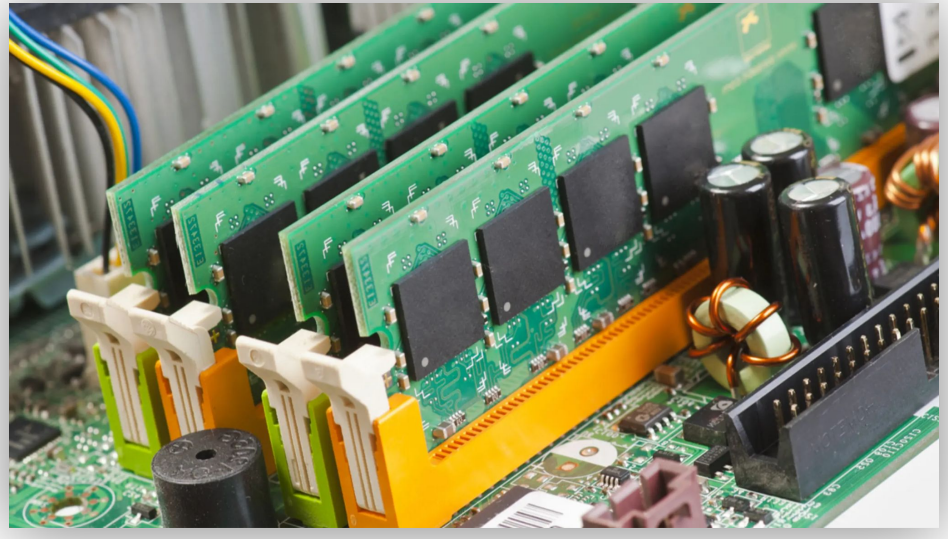
- Как же работает память в Java?
- Что такое Стек (The Stack) и Куча(The Heap)
- final/mutable/immutable
- Примеры несанкционированного доступа к данным
- Сборщик Мусора (Garbage Collector)

Java Memory

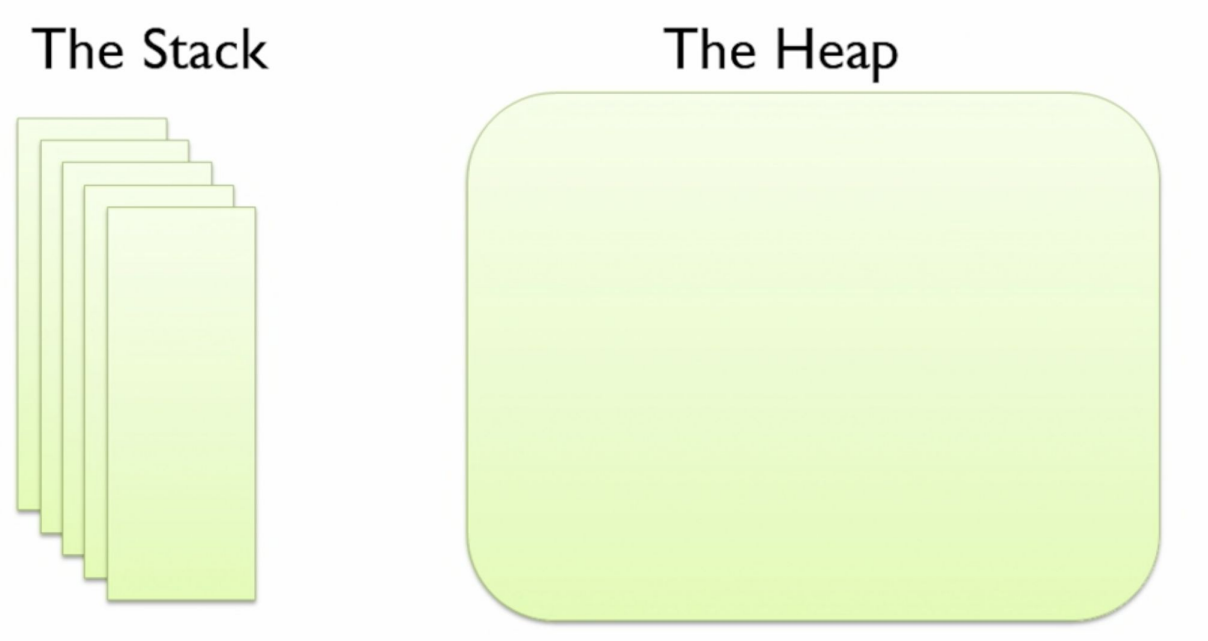
- Переменные
- Метадата
- Методы
- Код внутри метода

JVM

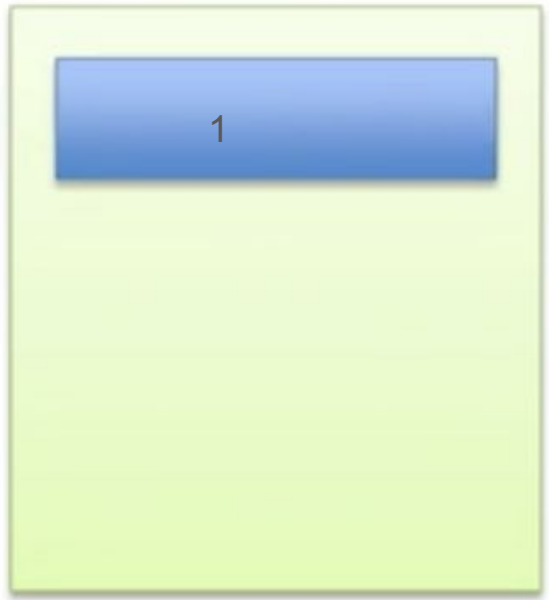
- Выделяет память под объект
- Удаляет неиспользованный объект из памяти
- Контролирует переполнение памяти
- Состоит из кучи и стека
-



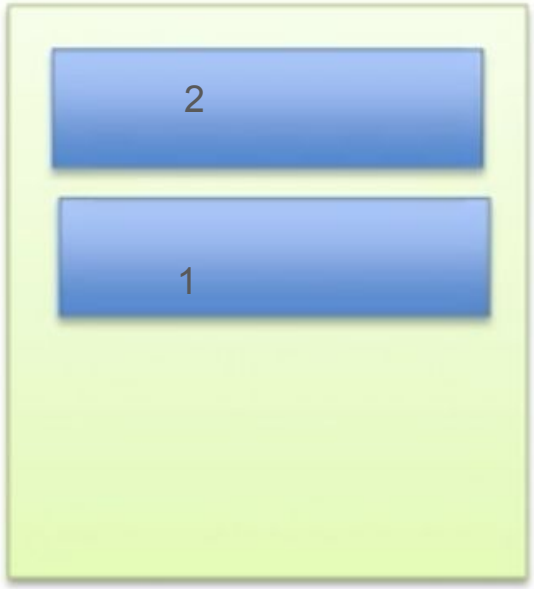
Стек(The Stack)



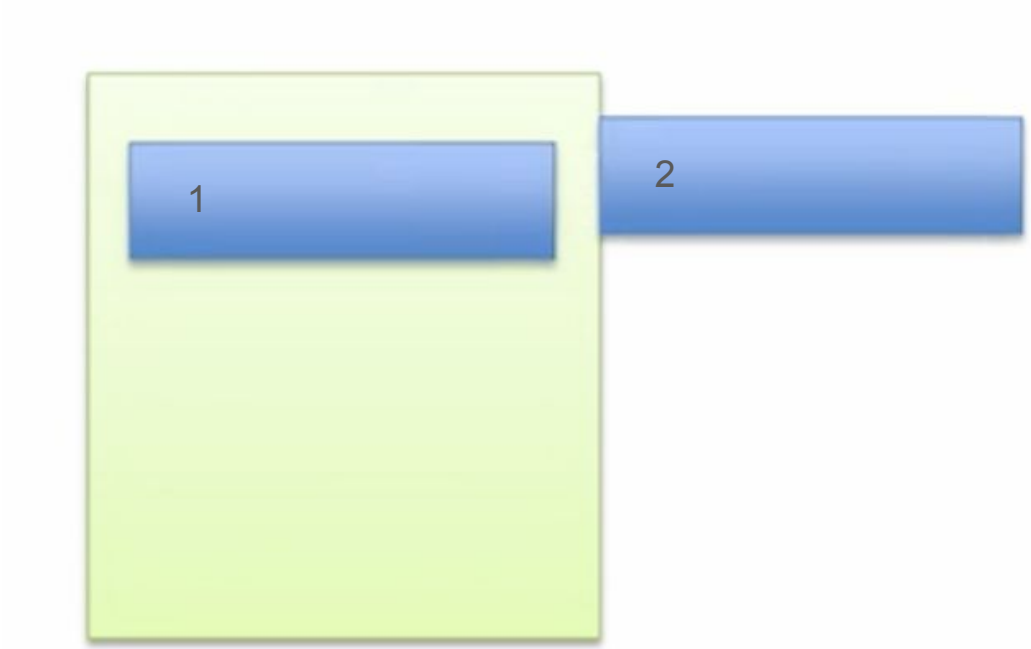
Стек(The Stack)



Стек(The Stack)



Стек(The Stack)

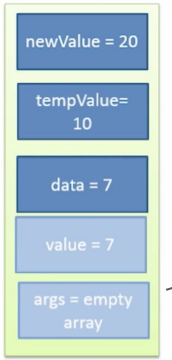


Стек(The Stack)

- FILO (First in last out) - первый зашел, последний вышел

- выполнение методов
- содержит примитивные типы и ссылки на объекты
- блоки методов
- после выполнения метода он удаляется из стека

Стек(The Stack)



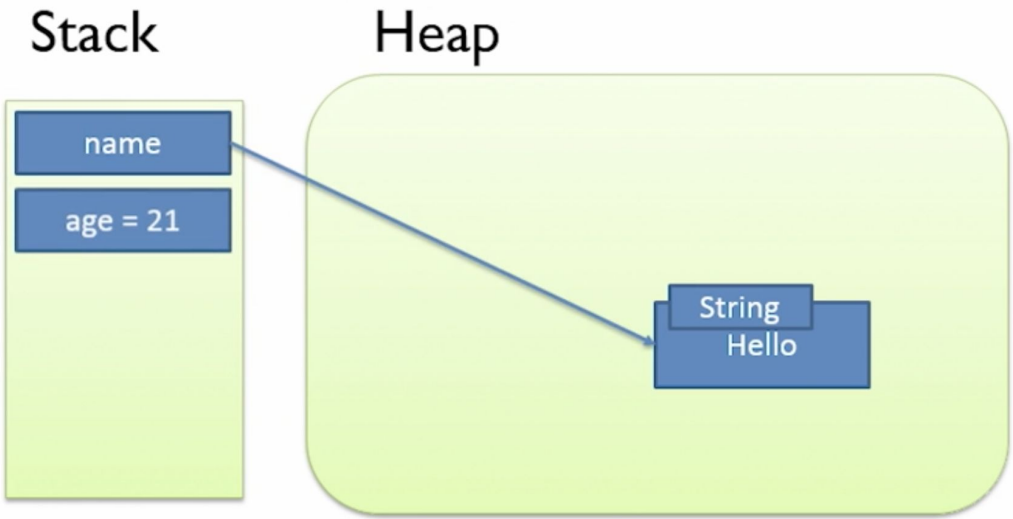

```java
public class Main {
    public static void main(String[] args) {
        int value = 7;
        value = calculateValue(value);
    }

    private static int calculateValue(int data) {
        int tempVal = data + 3;
        int newVal = tempVal * 2;
        return newVal;
    }
}
```

Куча (The Heap)

- Хранит все объекты приложения
- Доступен отовсюду в приложении по адресу памяти
- Объекты в куче содержат примитивные значения ссылки на другие объекты
- Куча и стек

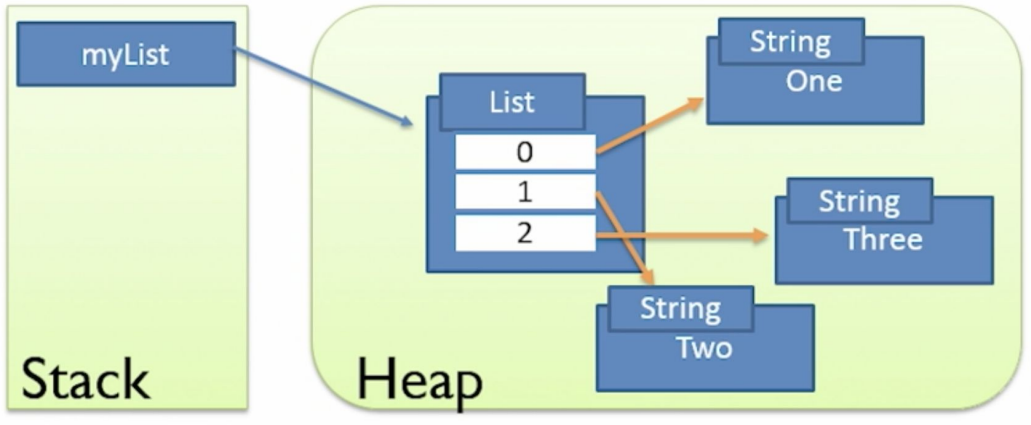
Куча (The Heap)

```
int age = 21;
String name = "Hello";
```



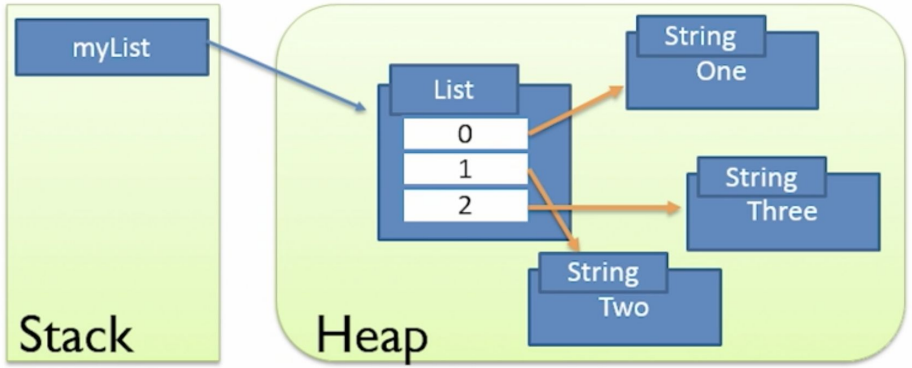
Куча (The Heap)

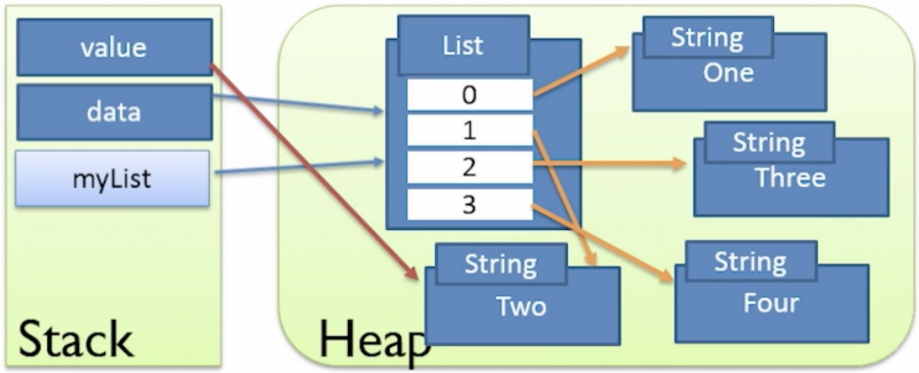
```java
public class Main2 {
    public static void main(String[] args) {
        List<String> myList = new ArrayList<>();
        myList.add("One");
        myList.add("Two");
        myList.add("Three");
        printList(myList);
    }

    private static void printList(List<String> data) {
        System.out.println(data);
    }
}
```



Куча (The Heap)

```java
public class Main2 {
    public static void main(String[] args) {
        List<String> myList = new ArrayList<>();
        myList.add("One");
        myList.add("Two");
        myList.add("Three");
        test(myList);
    }

    private static void test(List<String> data) {
        String value = data.get(1);
        data.add("Four");
        System.out.println(value);
    }
}
```



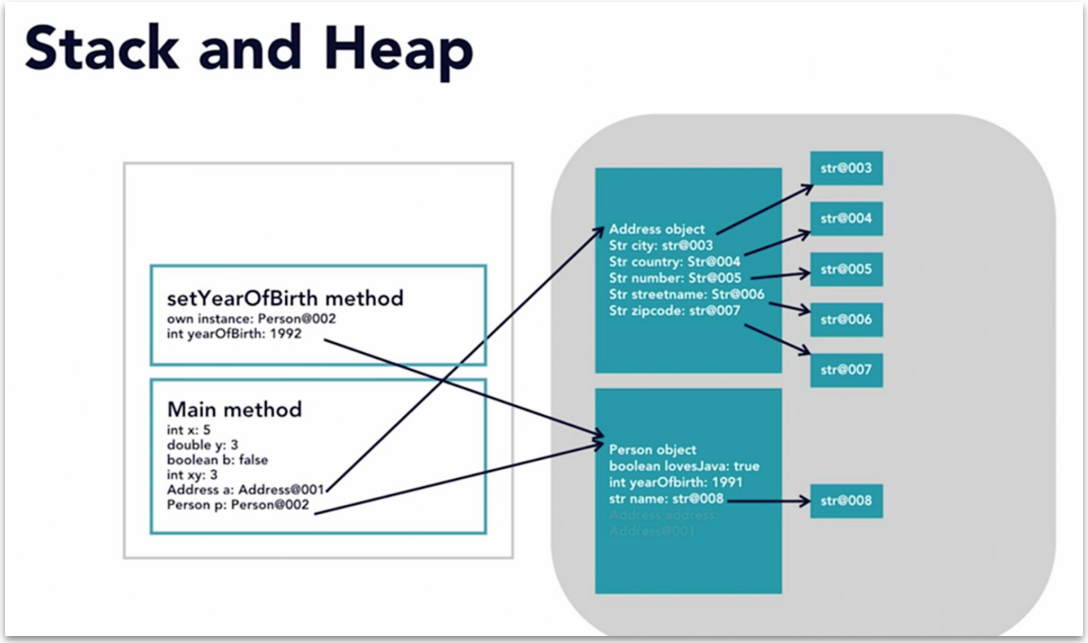


Куча и Стек

Стек	Куча
Только ссылки и примитивы	Объекты
Доступ к последнему	Доступ ко всем объектам
Маленькая память и удаляется сразу	Удаляется сборщиком мусора
Быстрый для доступа	Медленный для доступа
Существует пока работает метод	Существует пока работает приложение
StackOverFlowError	OutOfMemoryError

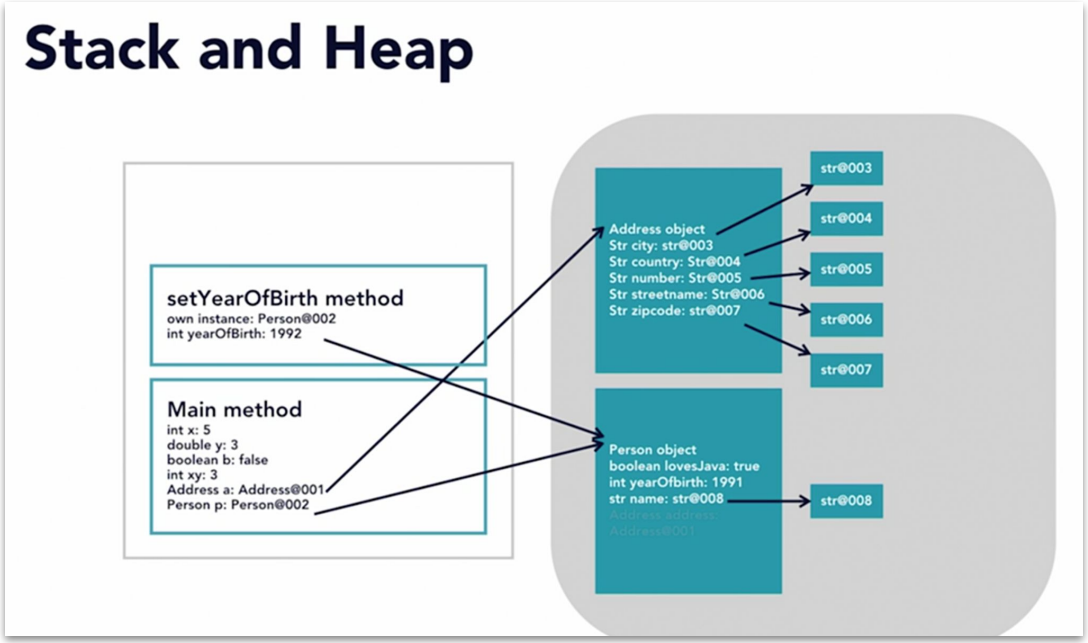
Методы

- Принимают объекты по ссылке
- Принимают примитивы по значению
- Если изменили объект, то он меняется везде (мы работаем с ссылкой на объект)
- == сравнивает объекты по ссылке



Методы

- Принимают объекты по ссылке
- Принимают примитивы по значению
- Если изменили объект, то он меняется везде (мы работаем с ссылкой на объект)
- == сравнивает объекты по ссылке

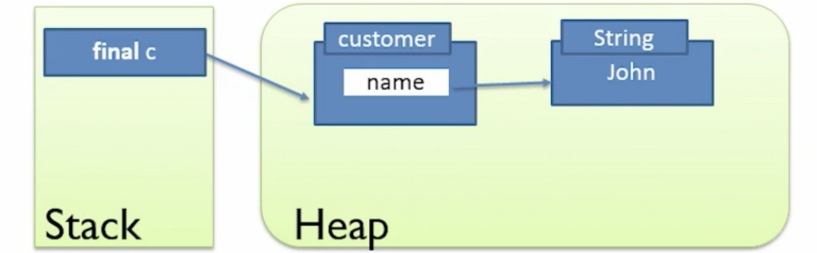


final переменные и immutable

- Имеет 1 состояние, при изменении создается новый объект
- Примитивные final работают так же как и обычные переменные
- Может быть объявлен только единожды, контролируется JVM

```
final Customer c = new Customer("John");
```

```
final Customer c;
c = new Customer("John");
c = new Customer("Susan");
```

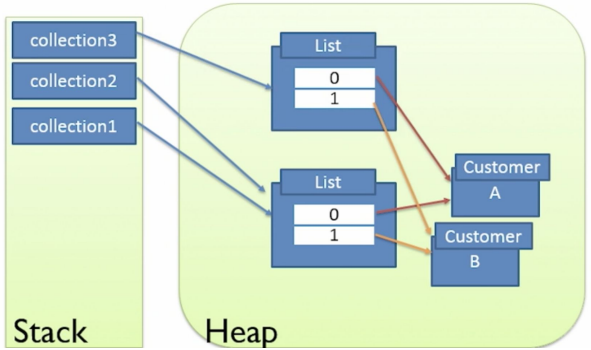


Immutable

- предотвратить изменение состояния извне

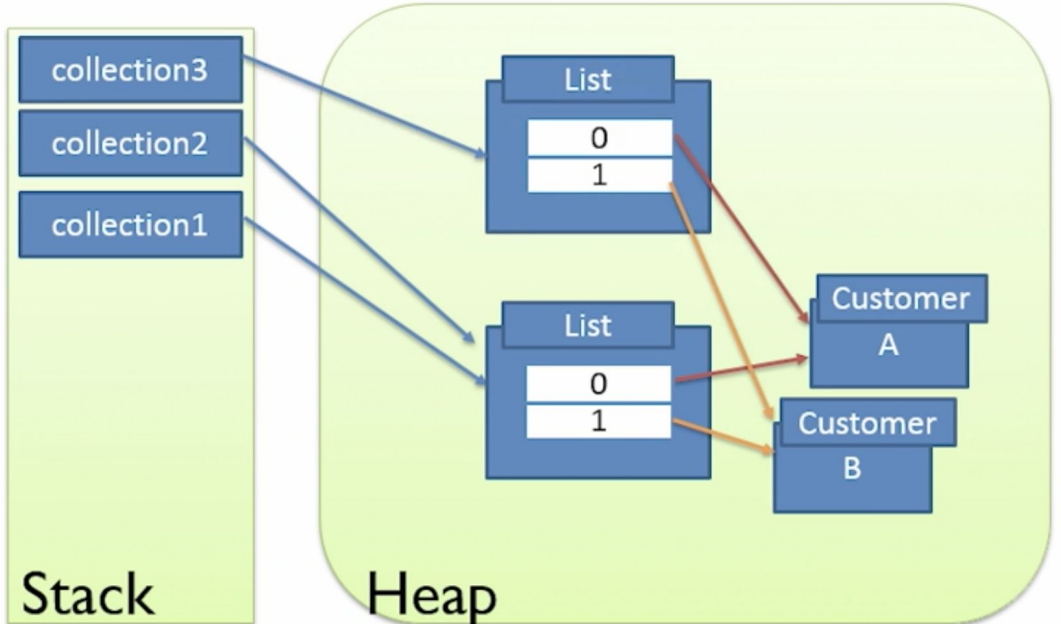
```
public class NotImmutableExample {
    private final int counter;
    private final String name;
    private final List<String> data;

    public NotImmutableExample(int counter, String name, List<String> data) {
        this.counter = counter;
        this.name = name;
        this.data = data;
    }

    public int getCounter() { return counter; }

    public String getName() { return name; }

    public List<String> getData() { return data; }
}
```

```
collection2 = collection1;
collection3 = new ArrayList<Customer>(collection1);
```

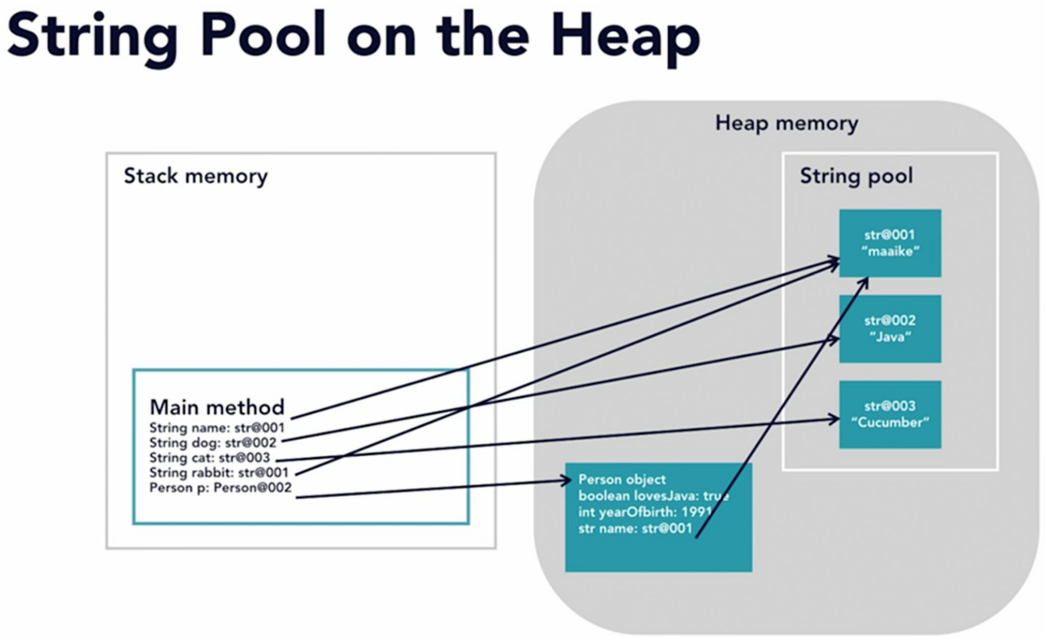


Immutable

```
collection2 = collection1;
collection3 = new ArrayList<Customer>(collection1);
```

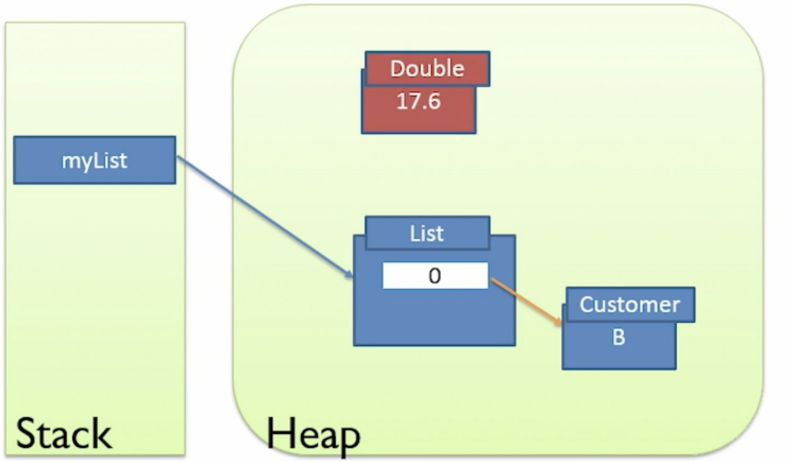


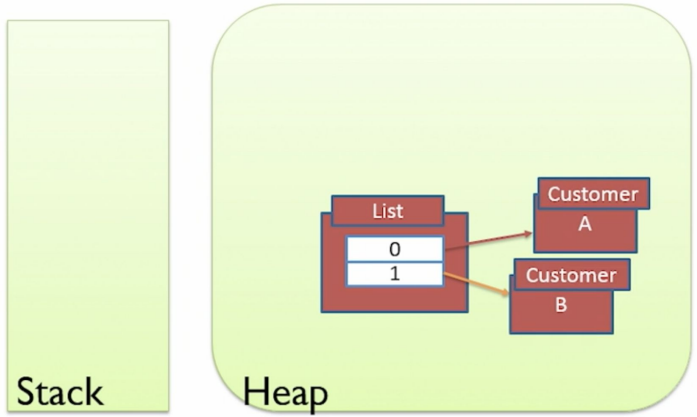
String Pool

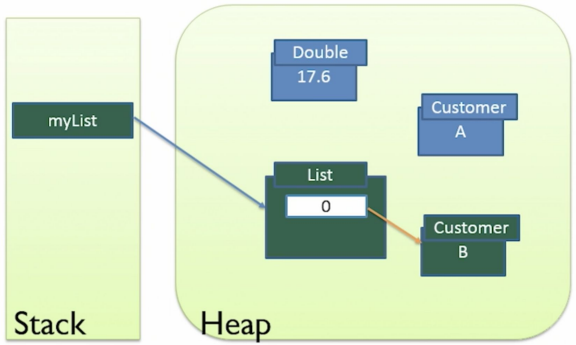


- Экономия места в памяти
- Оптимизация работы со строкой
- метод intern() скидывает в пул, если создали через конструктор
- “Test” -> new String(“Test”).intern()

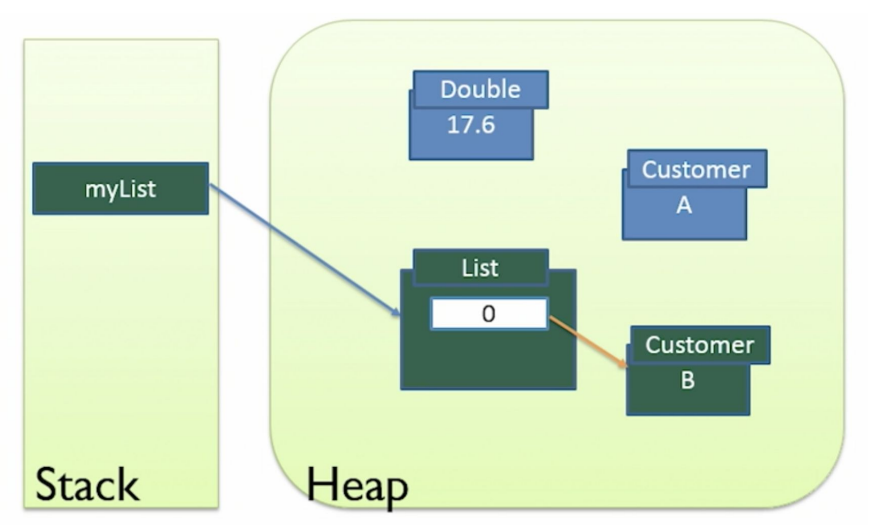
Сборщик Мусора

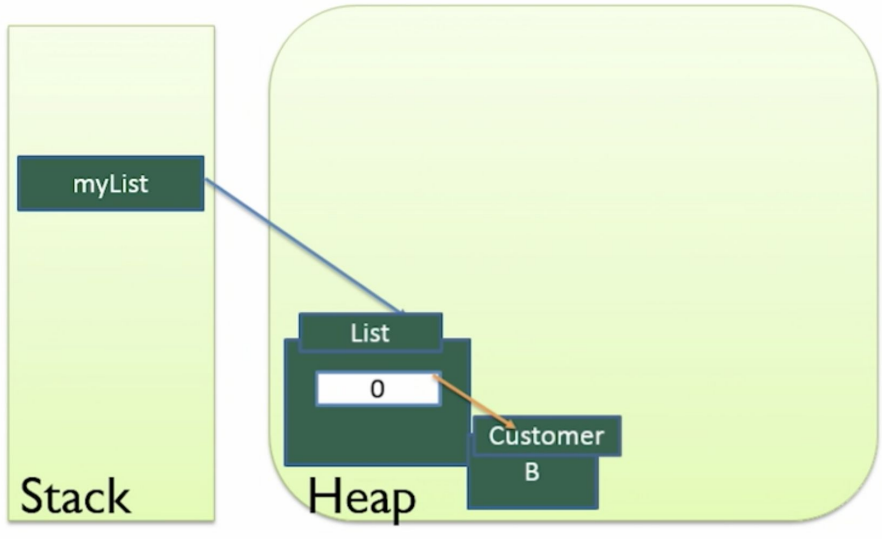






Mark & Sweep


myList
Double
17.6
Customer
A
List
0
Customer
B
Stack
Heap


myList
List
0
Customer
B
Stack
Heap

PermGen & MetaSpace

Garbage Collection

- удаляет из памяти объекты, на которые нет ссылки со стека (выполняется на фоне) -
- System.gc() - это лишь предложение для GC для чистки, не дает гарантию чистки в момент запуска
- finalize() метод вызывается когда происходит чистка - но нет гарантии его вызова
- Тюнинг:
    - увеличить и уменьшить память -Xmx10m
- Утечки памяти - есть ссылки на объекты которые не используются так как на их никто не будет использовать в будущем
    - использование много статики
    - VisualVM (или другие профилировщики) к примеру для мониторинга того что используется

Контрольные вопросы

1. где хранится объект?
2. когда удаляется объект?
3. когда удаляется метод из стека?
4. где хранятся примитивные переменные метода? примитивные переменные объекта
5. что работает быстрее куча или стек?
6. за что отвечает JMM?
7. Что такое Garbage Collector
8. Как работает Garbage Collector
9. Что такое стринг пул?
10. Как вызвать сброс в стринг пул?

Самостоятельное изучение

1. Copy https://habr.com/ru/post/246993/
2. Immutable https://habr.com/ru/company/otus/blog/552630/
3. Модель памяти ч1 https://habr.com/ru/post/510454/
4. Модель памяти ч2 https://habr.com/ru/post/510618/
5. Как устроена память https://www.javatpoint.com/memory-management-in-java
6. Heap size https://docs.oracle.com/cloud-machine/latest/soacs_gs/CSBCS/GUID-FC8FB9F1-99D4-434E-8825-3D368850A37A.htm
7. GC https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html