

JVM. Организация памяти, сборщики мусора, VisualVM



Григорий
Вахмистров



Григорий Вахмистров

Backend Developer в Tennisi.bet

План занятия

1. [Зачем знать JVM](#)
2. [Основные понятия](#)
3. [Как работает JVM: ClassLoaders](#)
4. [Как работает JVM: Runtime Data Area](#)
5. [Как работает JVM: Execution Engine](#)
6. [Как работает JVM: Execution Engine: Garbage Collection](#)
7. [Visual VM](#)
8. [Итоги](#)
9. [Домашнее задание](#)



Зачем знать JVM?

Зачем знать JVM?

- JVM - черный ящик. Приоткроем завесу тайны.



Зачем знать JVM?

- JVM - черный ящик. Приоткроем завесу тайны.
- Знание, как все работает “под капотом” поможет эффективно использовать экосистему Java.



Зачем знать JVM?

- JVM - черный ящик. Приоткроем завесу тайны.
- Знание, как все работает “под капотом” поможет эффективно использовать экосистему Java.
- Будут очевидны и понятны многие концепции, сложные для понимания с нуля.





Основные понятия

JVM



Java Virtual Machine (JVM, Java VM) — виртуальная машина Java, основная часть исполняющей системы Java

JVM



Java Virtual Machine (JVM, Java VM) — виртуальная машина Java, основная часть исполняющей системы Java



Java Runtime Environment (JRE) — минимальное окружения для запуска java-приложений.
Содержит *JVM* и инструменты развёртывания

JVM



Java Virtual Machine (JVM, Java VM) — виртуальная машина Java, основная часть исполняющей системы Java



Java Runtime Environment (JRE) — минимальное окружения для запуска java-приложений. Содержит *JVM* и инструменты развёртывания



Java Development Kit (JDK) - окружение для разработки. *JRE* + инструменты разработчика

Java-программа

Java-программа - это набор текстовых файлов с расширением .java и написанных в корректном синтаксисе джава



Основной концепт Java

Write once, run anywhere (WORA) - написал однажды, запустил где угодно.

Основной концепт Java

Write once, run anywhere (WORA) - написал однажды, запустил где угодно.



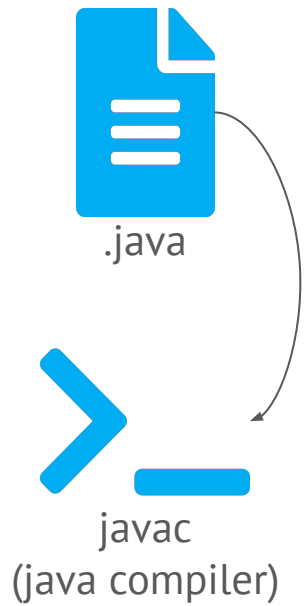
Как это становится возможным?

Как работает Java

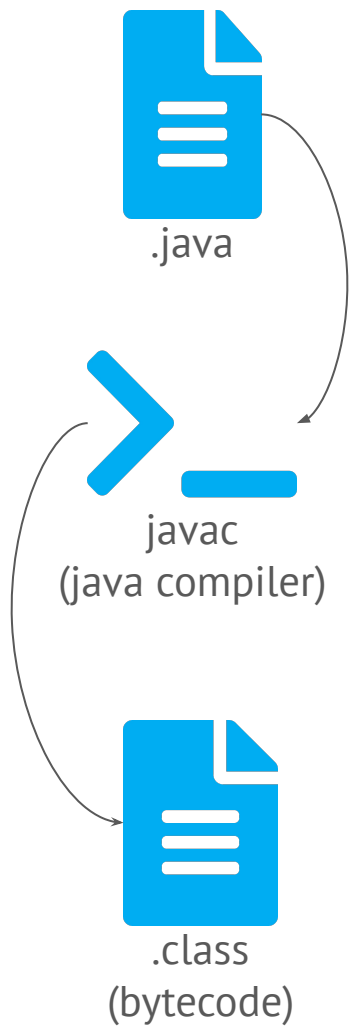


.java

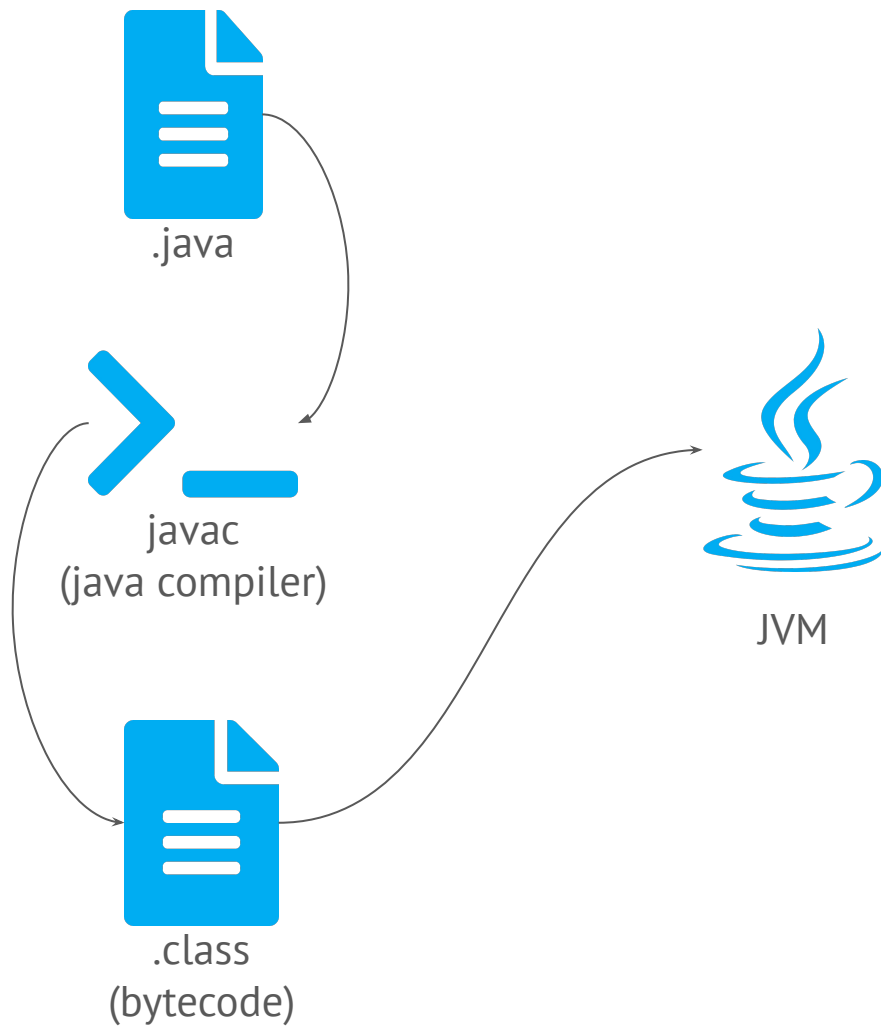
Как работает Java



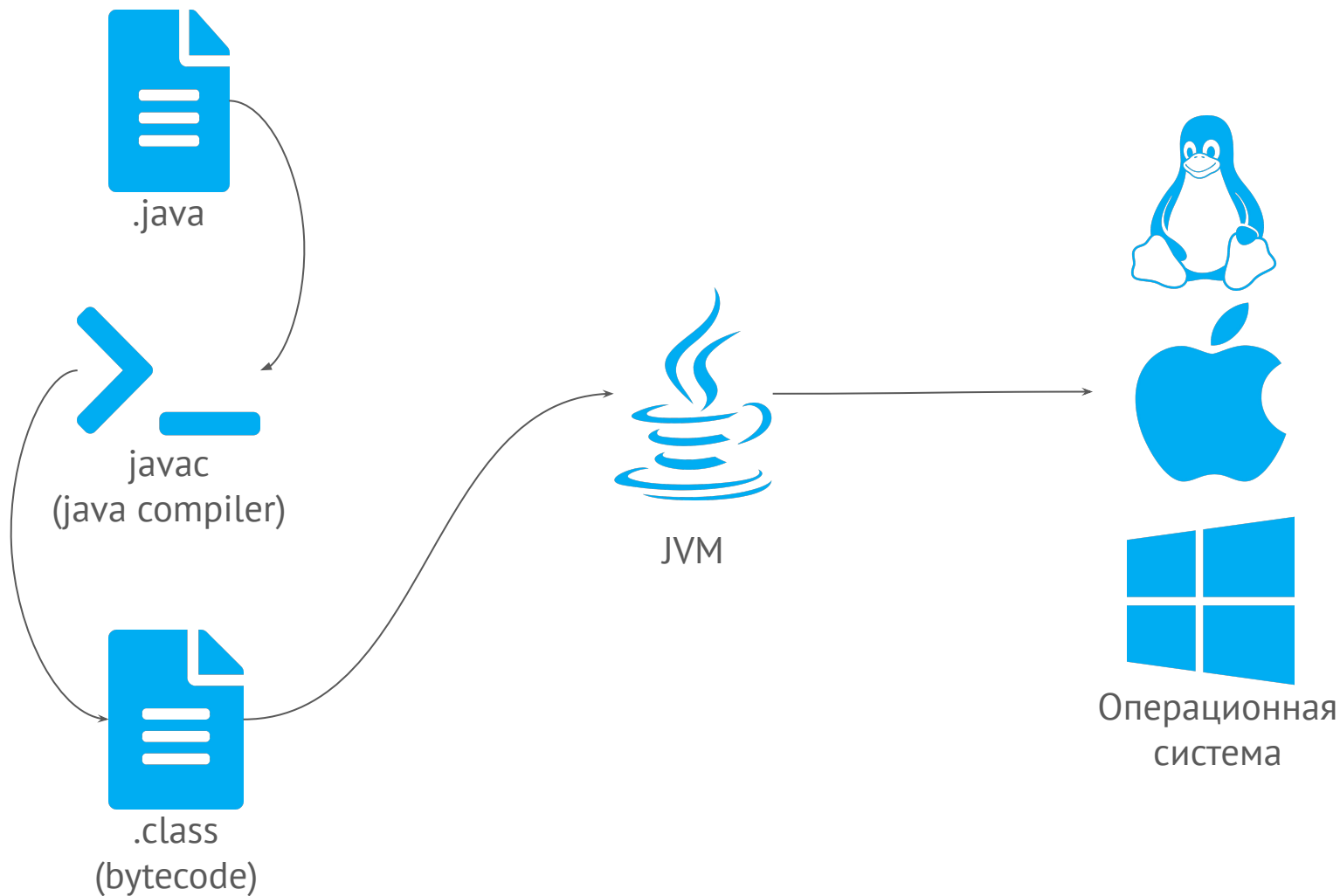
Как работает Java



Как работает Java



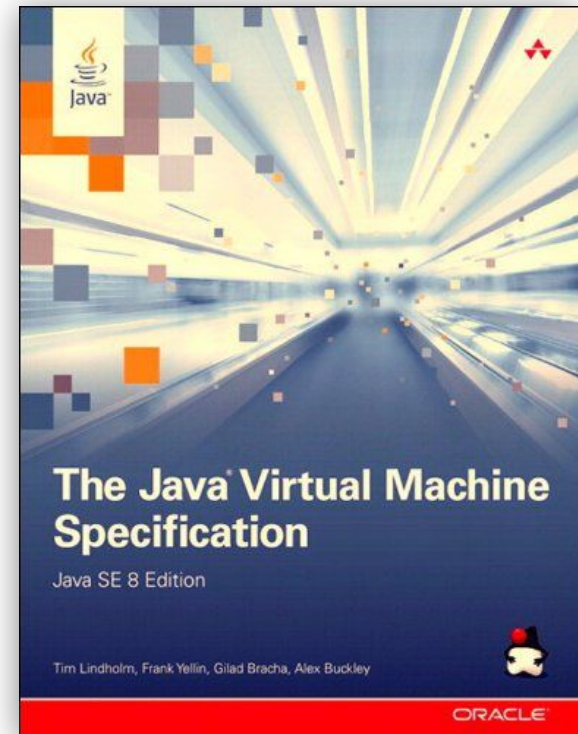
Как работает Java



JVM - не существует

Это просто **спецификация** :)

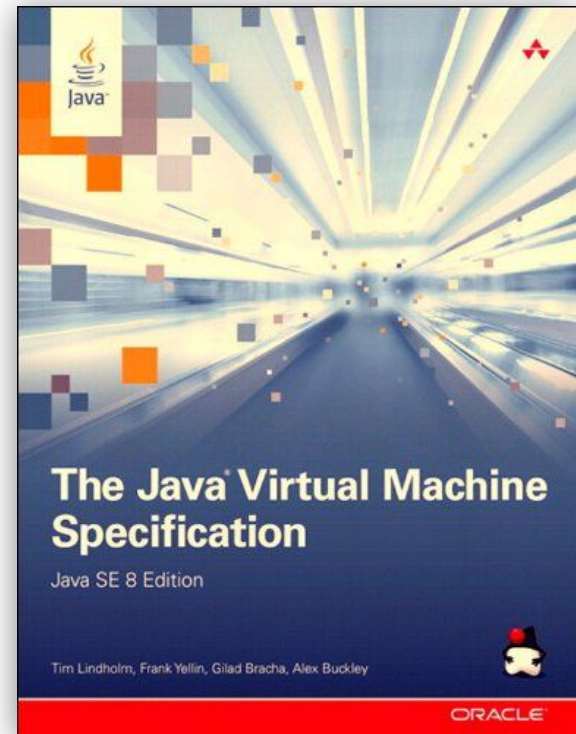
- У которой есть различные реализации.



JVM - не существует

Это просто **спецификация** :)

- У которой есть различные реализации.
- Под разные OS и разные потребности.





Как работает JVM

Как работает JVM

1. ClassLoaders

Подсистема загрузчиков классов

Есть код:

```
class Simple {  
    public static void main(String[] args)  
    {  
        Netology.sout();  
    }  
}
```

Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args)  
    {  
        Netology.sout();  
    }  
}
```

Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args){  
        Netology.sout();  
    }  
}
```

Подгрузи классы Simple, Netology

Class
Loading

Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

Подгрузи классы Simple, Netology

Bootstrap ClassLoader

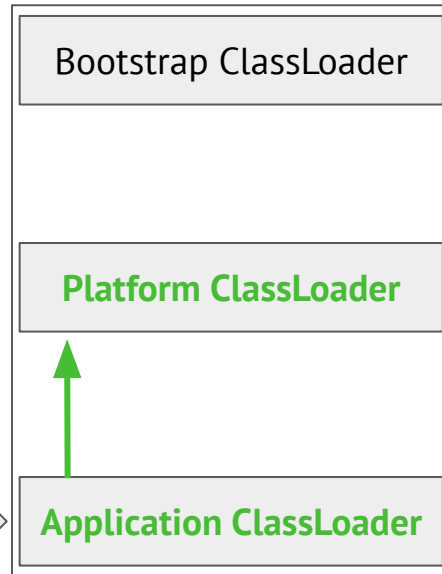
Platform ClassLoader

Application ClassLoader

Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

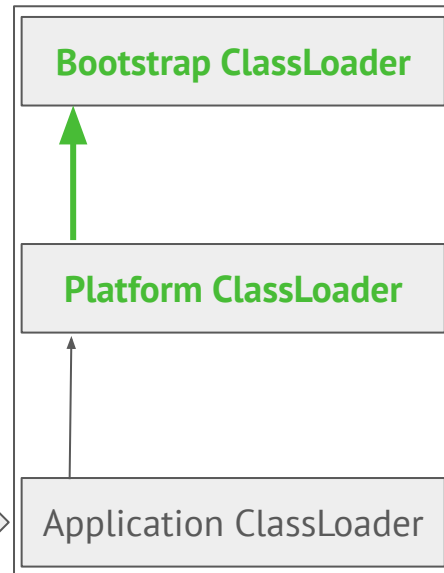
Подгрузи классы Simple, Netology



Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

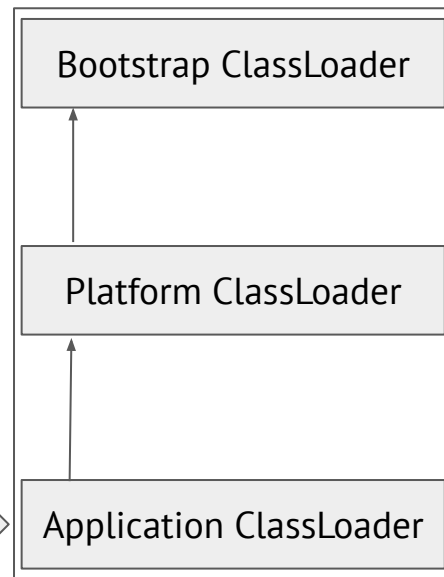
Подгрузи классы Simple, Netology



Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

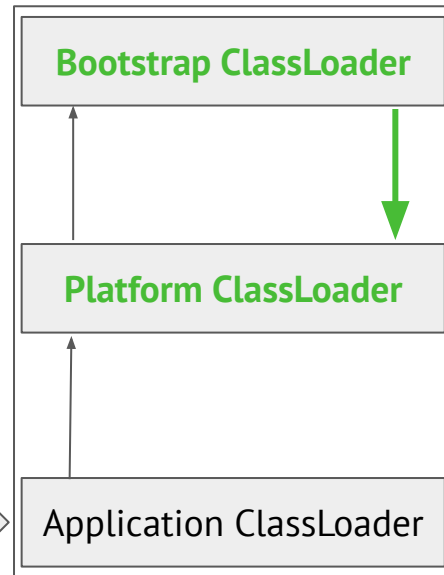
Подгрузи классы Simple, Netology



Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

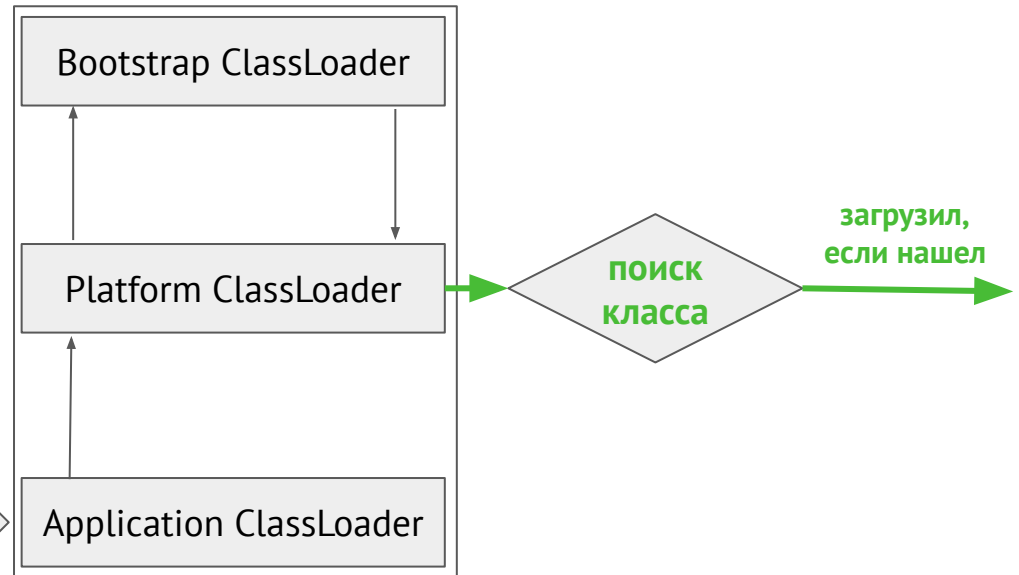
Подгрузи классы Simple, Netology



Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

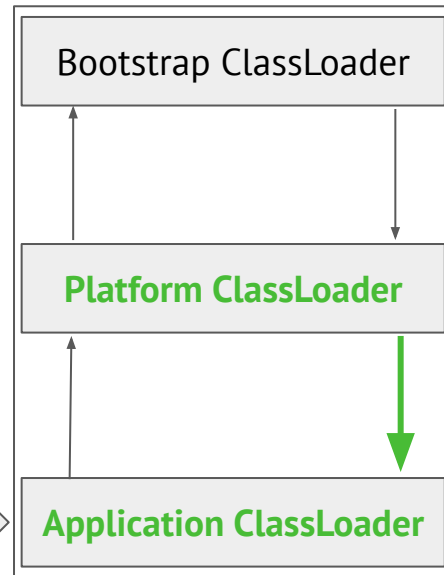
Подгрузи классы Simple, Netology



Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

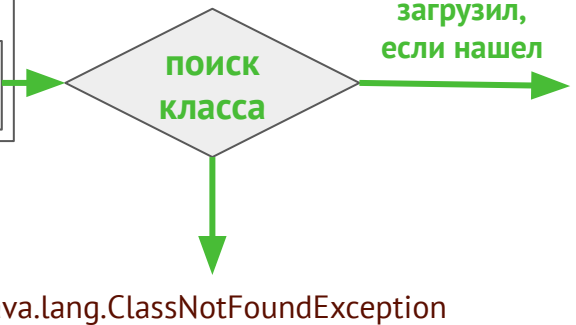
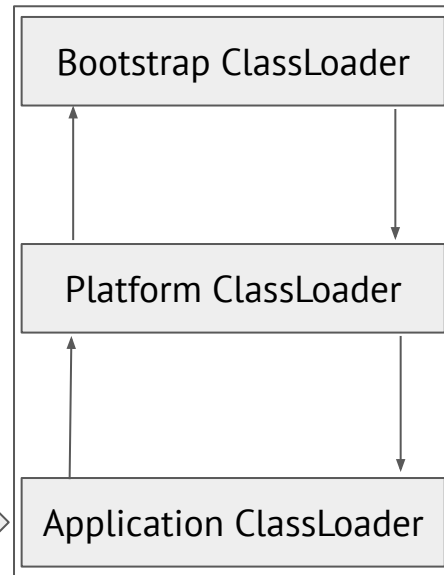
Подгрузи классы Simple, Netology



Подсистема загрузчиков классов

```
class Simple {  
    public static void main(String[] args) {  
        Netology.sout();  
    }  
}
```

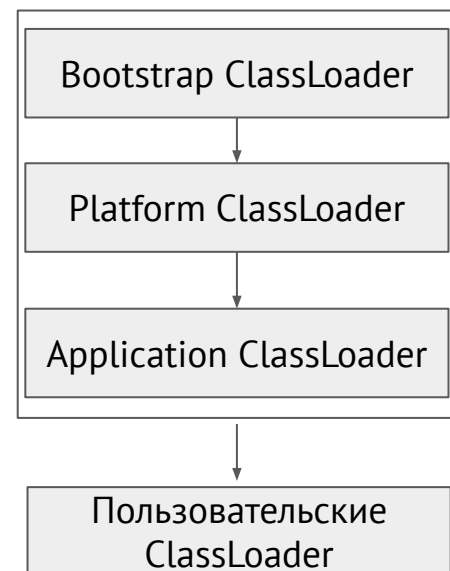
Подгрузи классы Simple, Netology



Подсистема загрузчиков классов

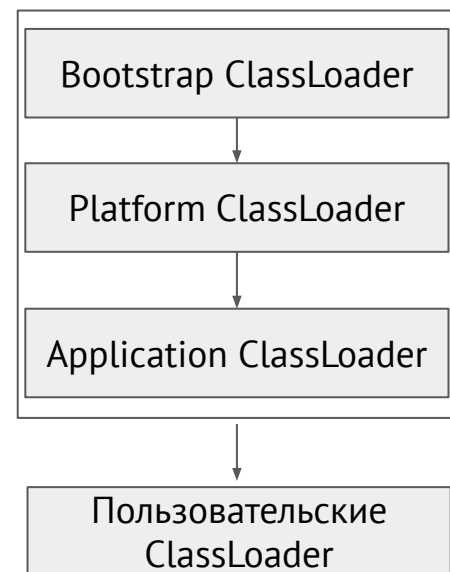
Можно написать свои загрузчики.

Например, для независимости приложений
(самый известный пример - загрузчики
Tomcat)



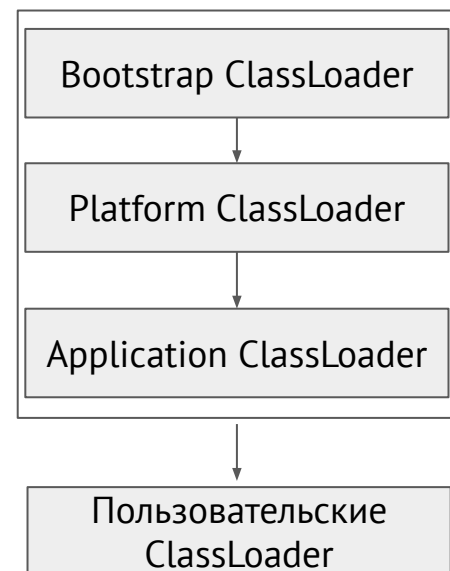
Подсистема загрузчиков классов

Повторим:



Подсистема загрузчиков классов

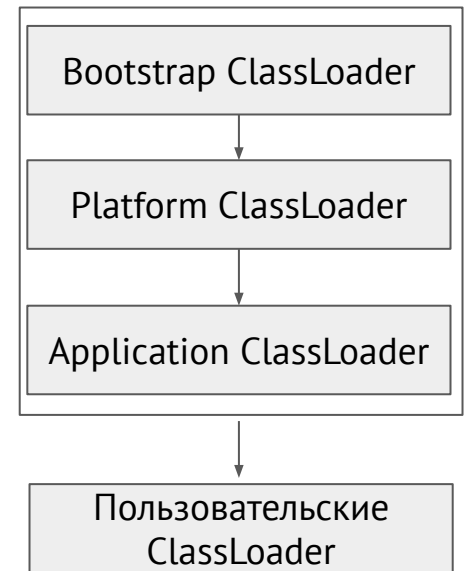
Когда в коде встречается новый класс - он отдается для загрузки в систему загрузчиков классов



Подсистема загрузчиков классов

Когда в коде встречается новый класс - он отдается для загрузки в систему загрузчиков классов

Соответственно, подгрузка классов “ленивая”.

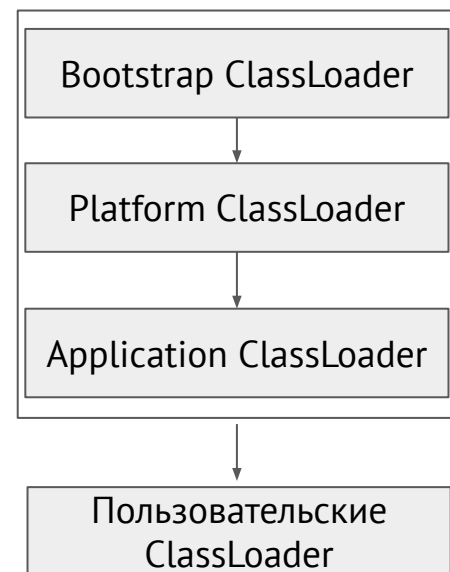


Подсистема загрузчиков классов

Когда в коде встречается новый класс - он отдается для загрузки в систему загрузчиков классов

Соответственно, подгрузка классов “ленивая”.

Три класслоадера, делегирующие загрузку классов - разных уровней.



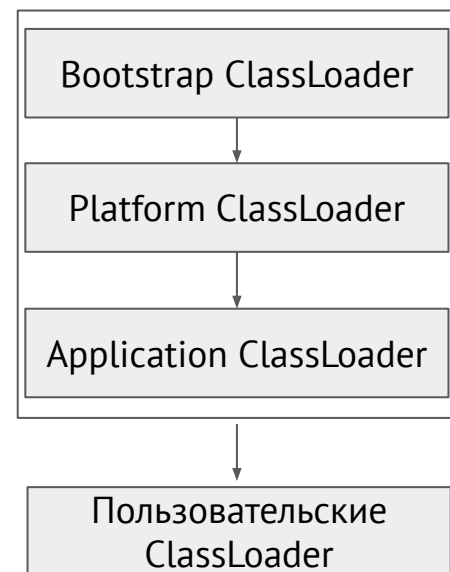
Подсистема загрузчиков классов

Когда в коде встречается новый класс - он отдается для загрузки в систему загрузчиков классов

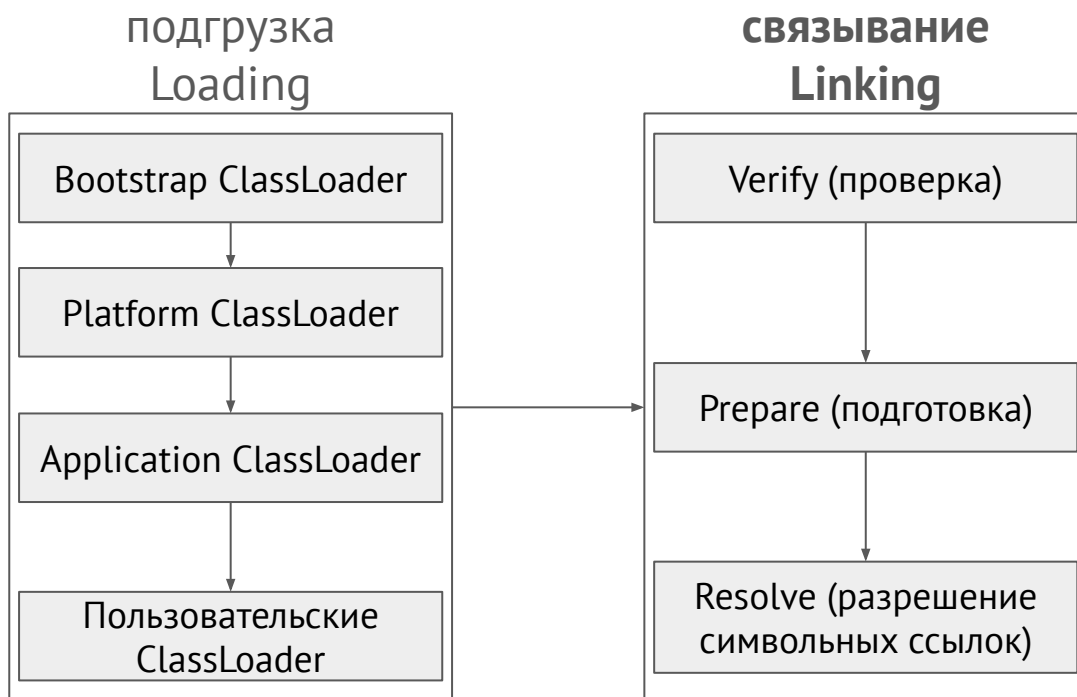
Соответственно, подгрузка классов “ленивая”.

Три класслоадера, делегирующие загрузку классов - разных уровней.

Можно написать собственный ClassLoader.



Подсистема загрузчиков классов



Подсистема загрузчиков классов

Здесь происходит подготовка классов к выполнению

СВЯЗЫВАНИЕ
Linking

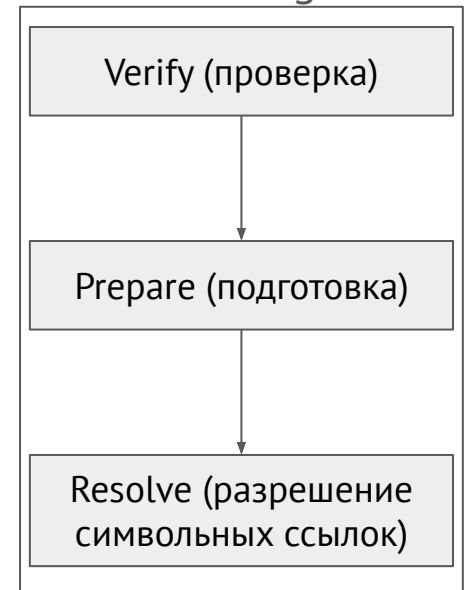


Подсистема загрузчиков классов

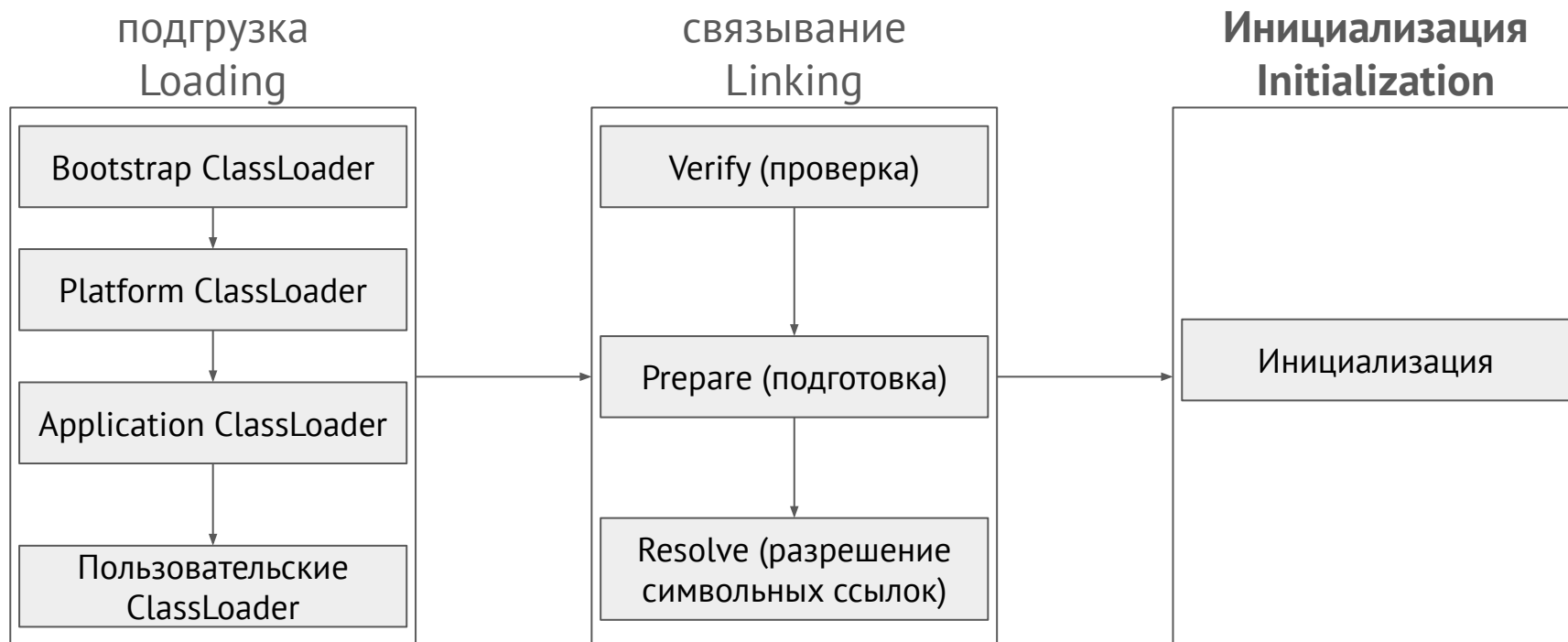
Здесь происходит подготовка классов к выполнению:

- проверка, что код валиден,
- подготовка примитивов в статических полях,
- связывание ссылок на другие классы.

СВЯЗЫВАНИЕ Linking



Подсистема загрузчиков классов



Подсистема загрузчиков классов

Инициализация класса:

Выполняются static инициализаторы и инициализаторы static полей.

```
class Netology {  
    static int a = initStaticField();  
  
    static {  
        System.out.println("static init");  
    }  
  
    static int initStaticField() {  
        return 42;  
    }  
}
```

Инициализация
Initialization

Инициализация

Как работает JVM

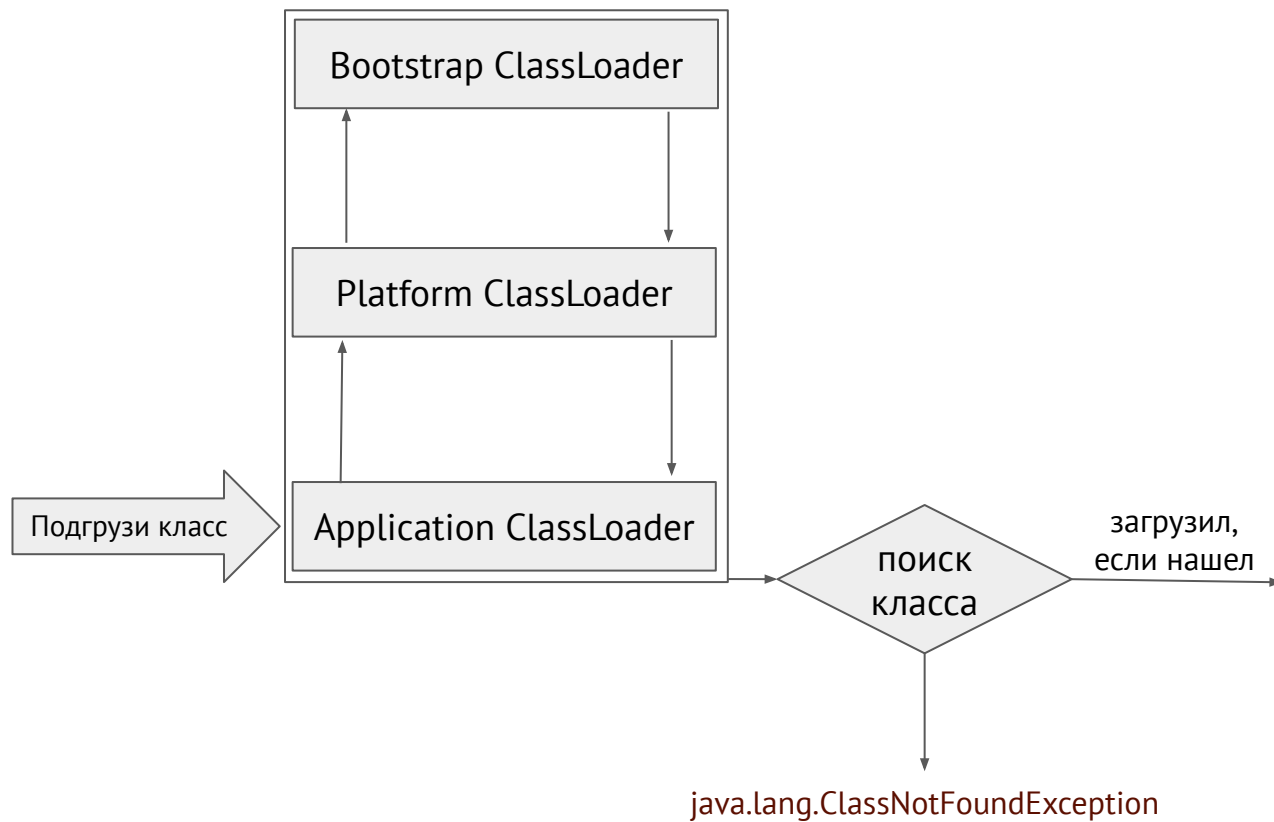
1. ClassLoaders
2. Runtime Data Area

~~Области тьмы~~ Области памяти

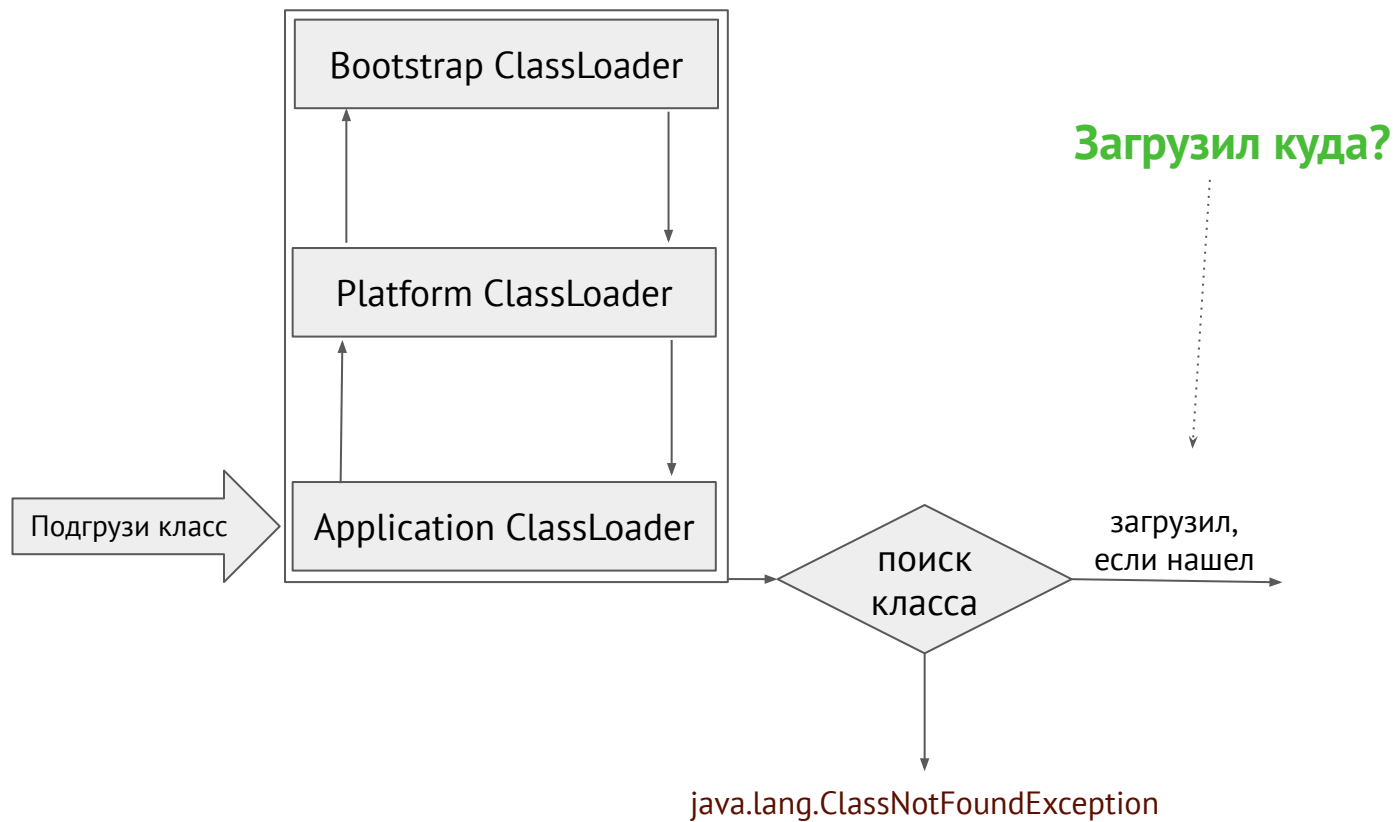


Область памяти

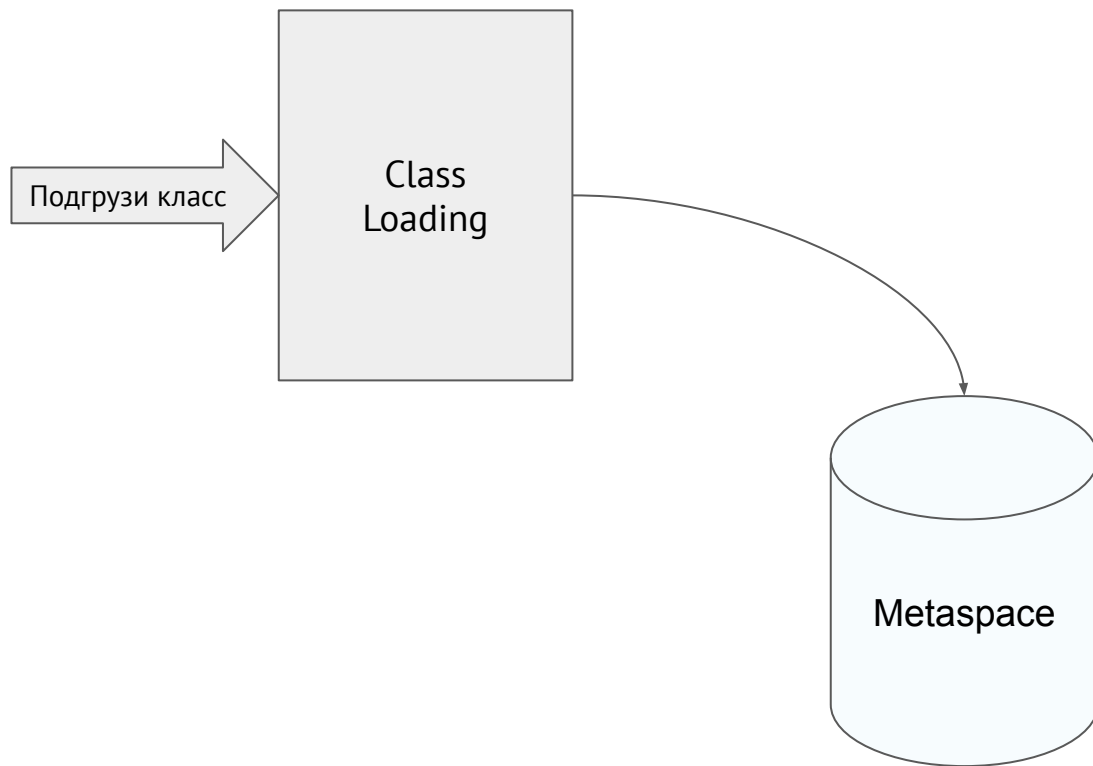
Помните?



Область памяти

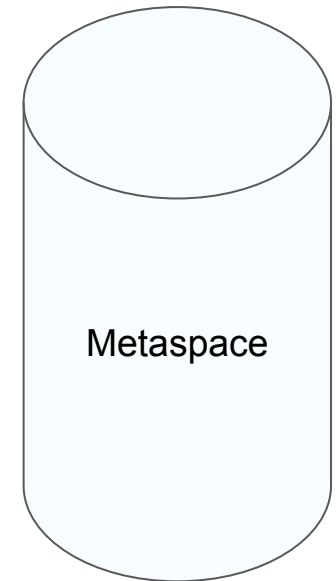


Область памяти: Metaspase



Область памяти: Metaspase

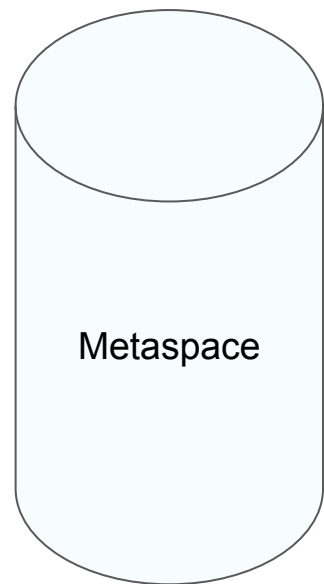
Metaspase (в прошлом PermGen) - область памяти, где хранится мета-информация



Область памяти: Metaspase

Metaspase (в прошлом PermGen) - область памяти, где хранится мета-информация:

- данные о классе (имя, методы, поля и др);
- константы.



Область памяти: Metaspace

Metaspace (в прошлом PermGen) - область памяти, где хранится мета-информация:

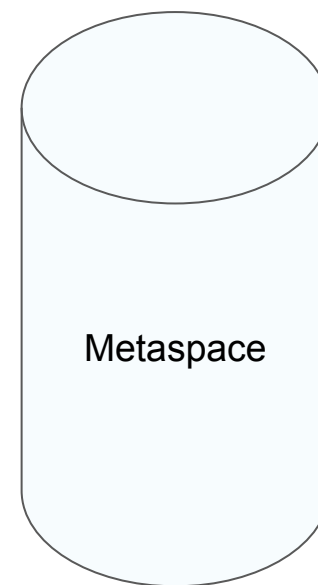
- данные о классе (имя, методы, поля и др);
- константы.

Его размер можно настроить параметром запуска

-XX:MetaspaceSize=N

-XX:MaxMetaspaceSize=N

(по дефолту неограничен)




Что нужно на практике

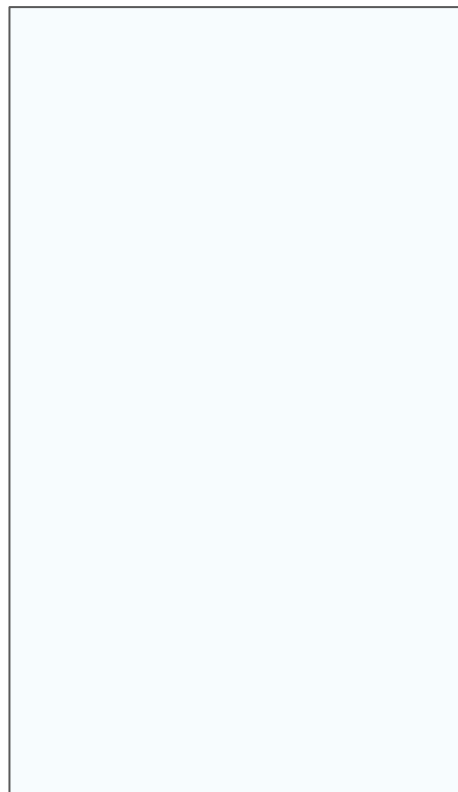
Области памяти

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Object obj = new Object();  
        Memory mem = new Memory();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```

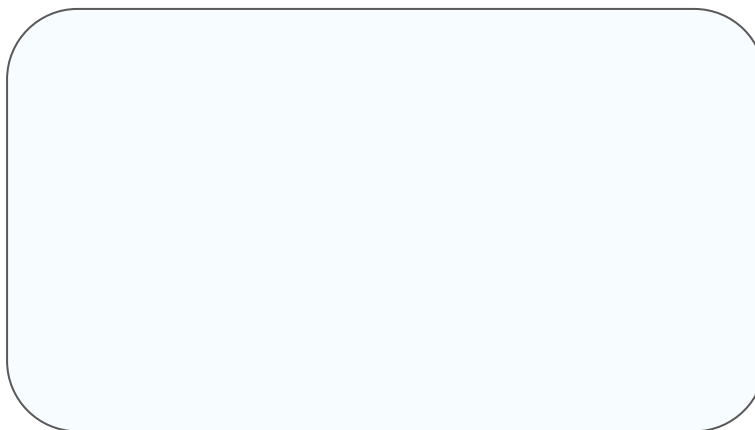
Области памяти



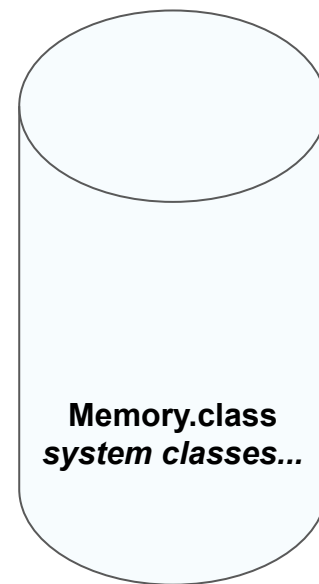
```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



Stack Memory



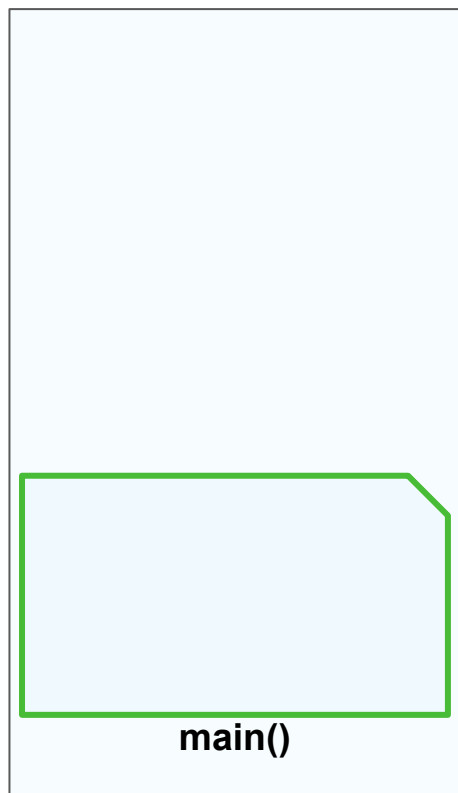
heap (куча)



Metaspace

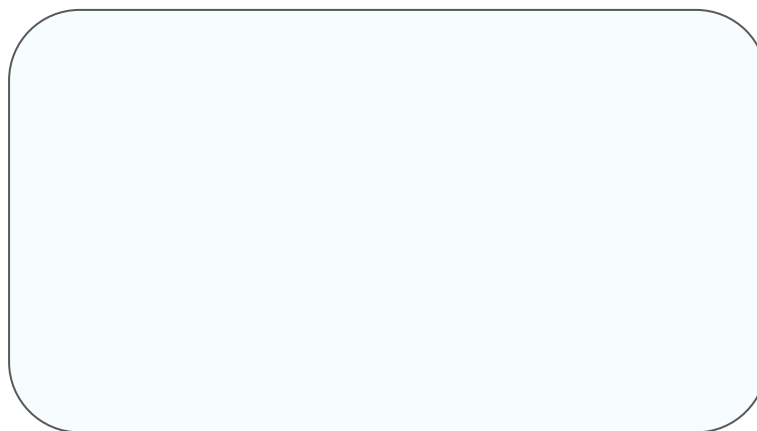
Области памяти

В момент вызова метода
создается фрейм(кадр) в стеке

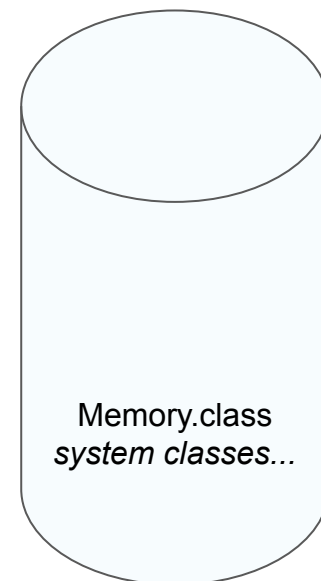


Stack Memory

```
public class Memory {  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



heap (куча)



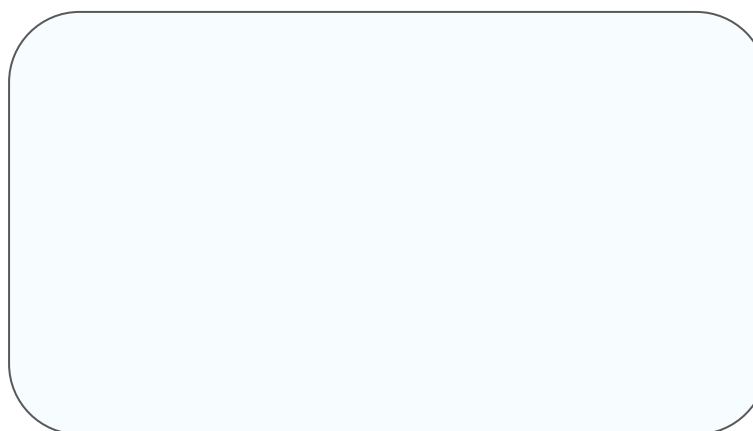
Metaspaces

Области памяти

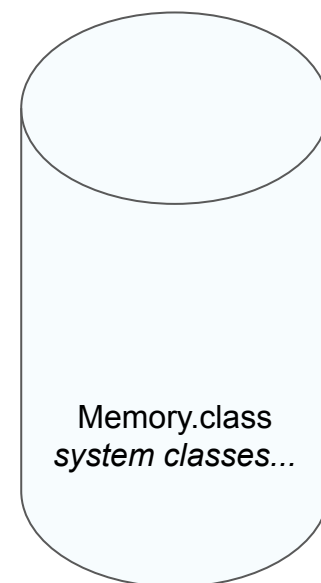


Stack Memory

```
public class Memory {  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```

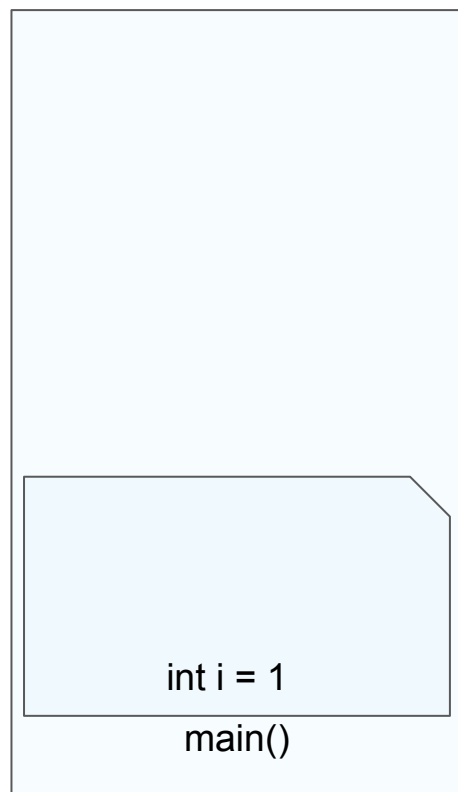


heap (куча)




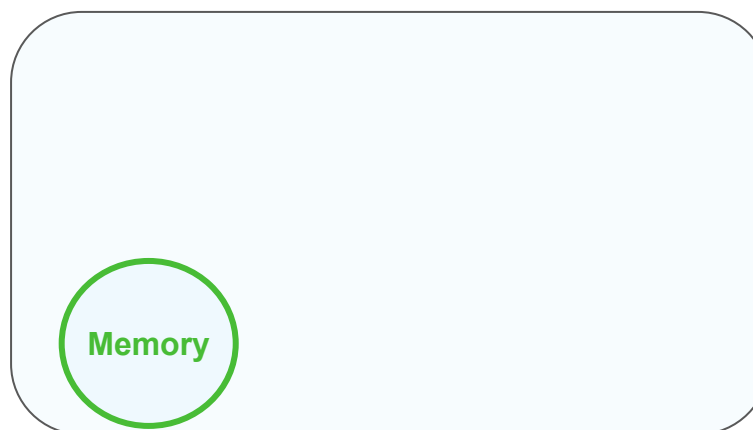
Metaspace

Области памяти

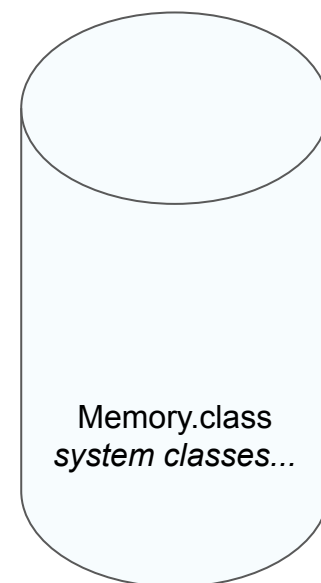


Stack Memory

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();   
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



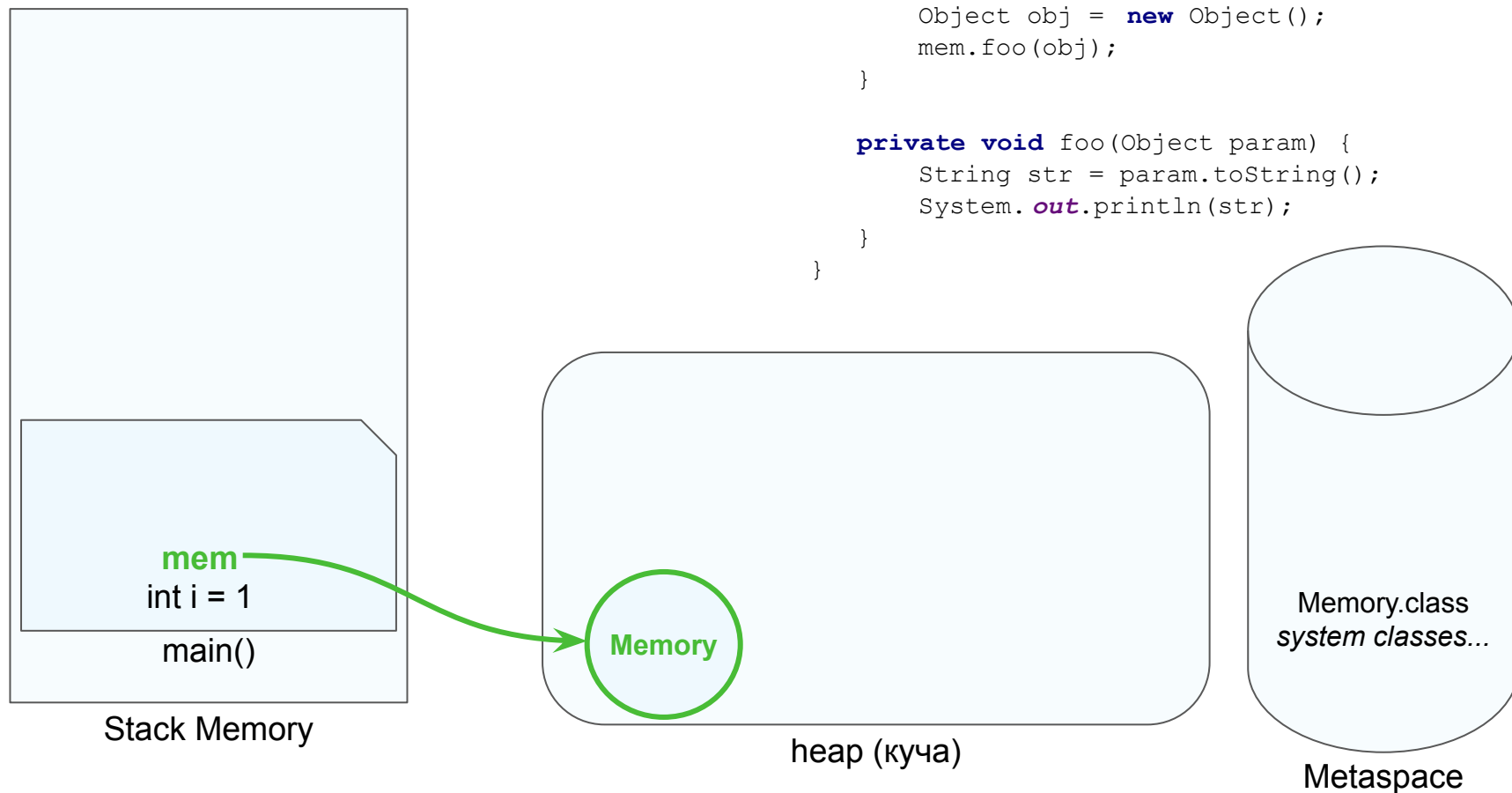
heap (куча)




Metaspaces

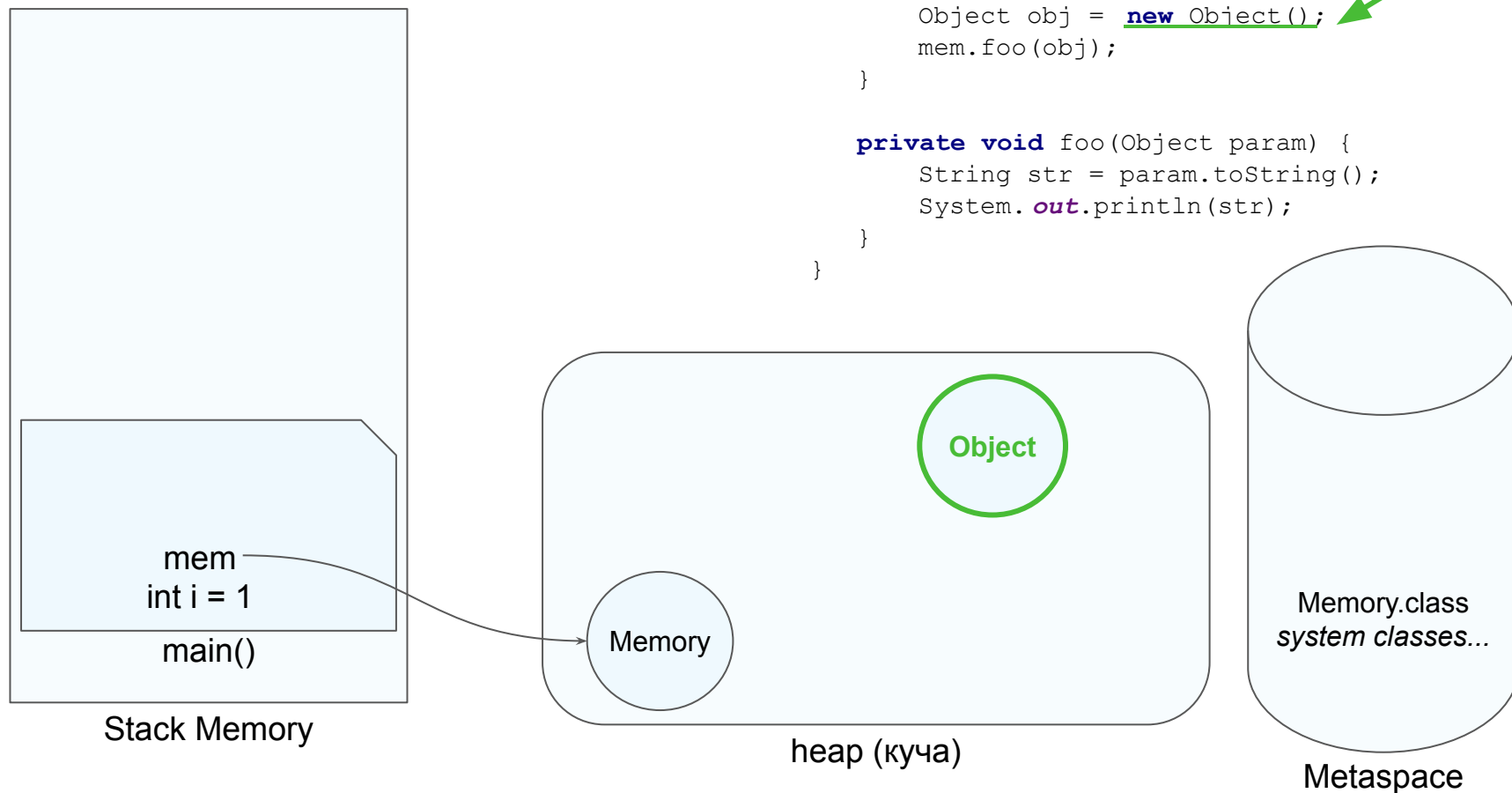
Области памяти

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



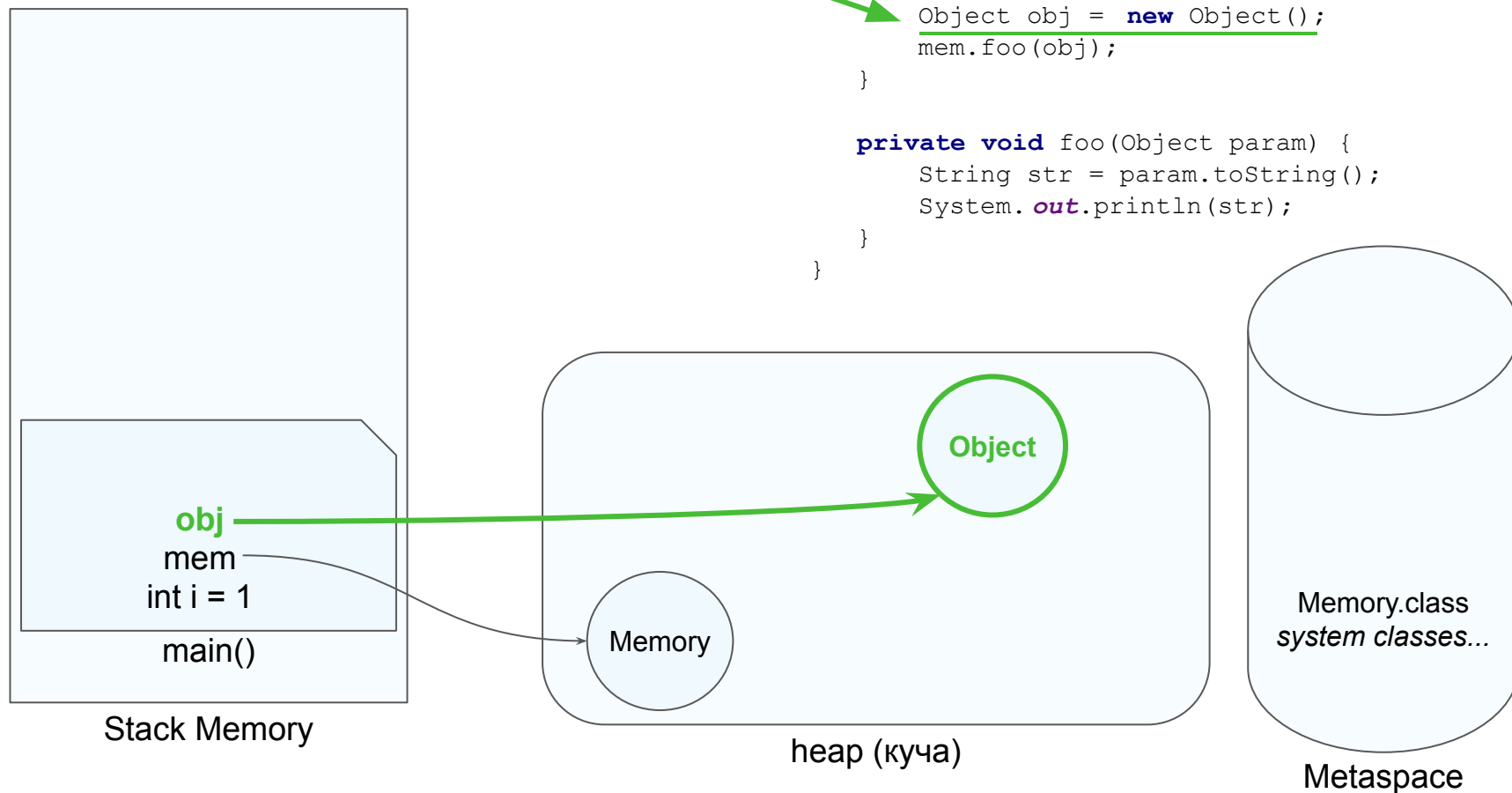
Области памяти

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();   
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



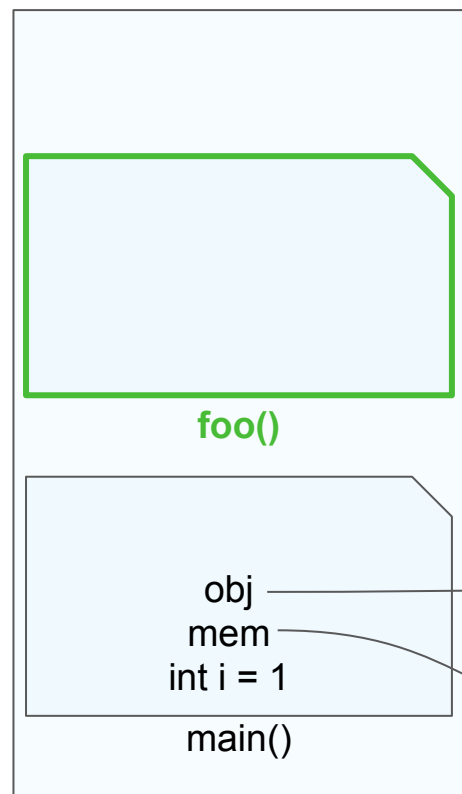
Области памяти

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



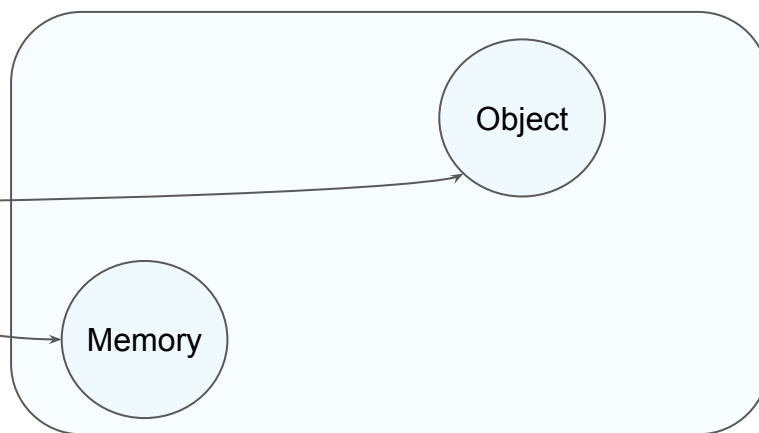
Области памяти

В момент вызова метода
создается фрейм в стеке

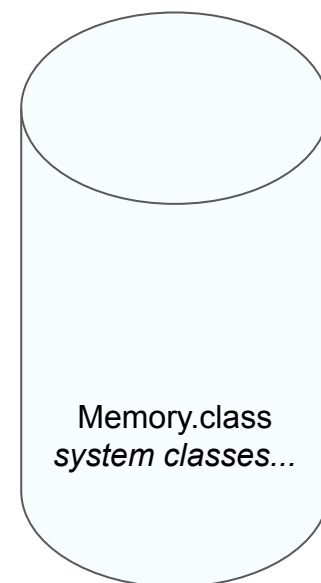


Stack Memory

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



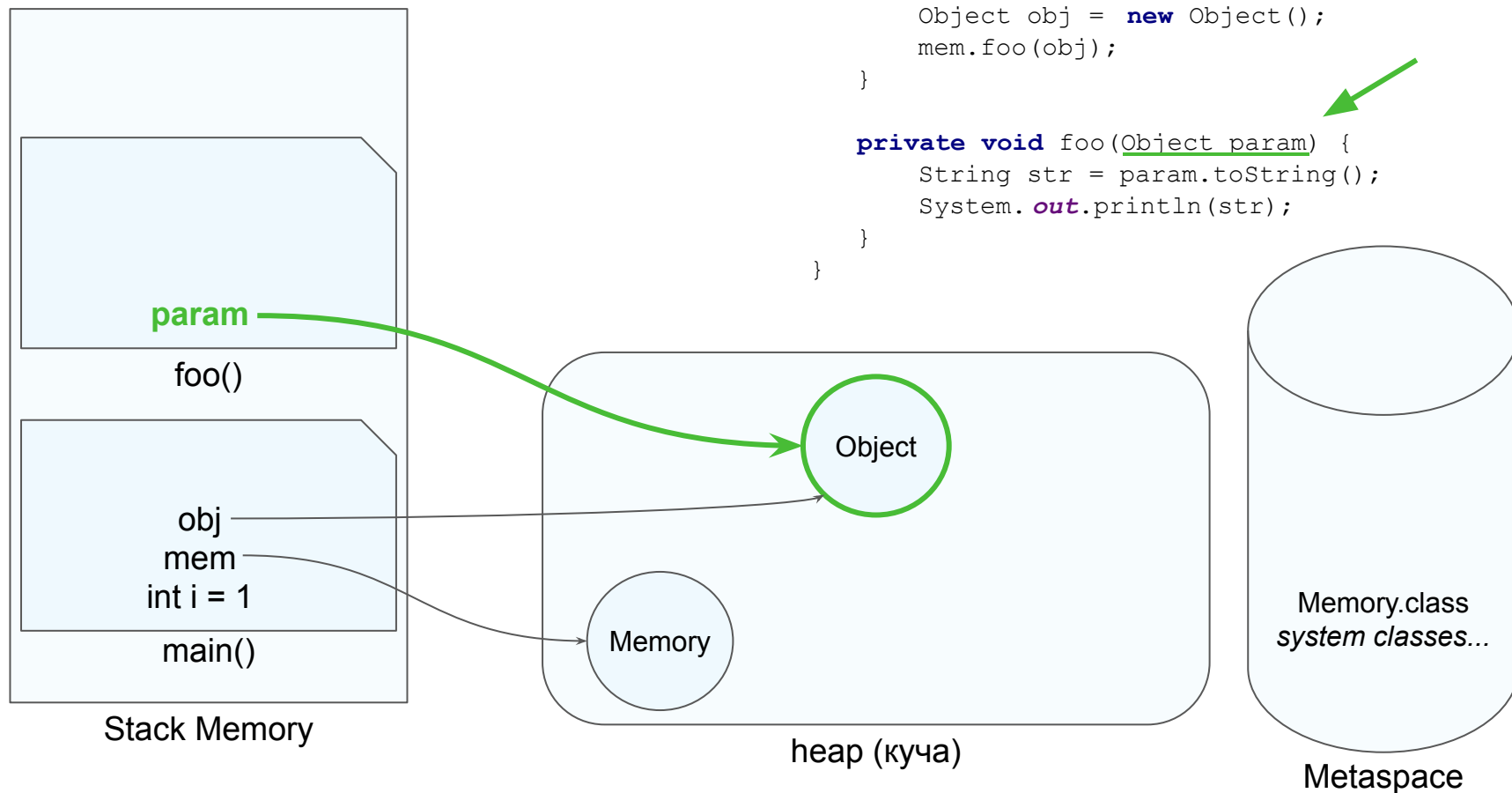
heap (куча)



Metaspace

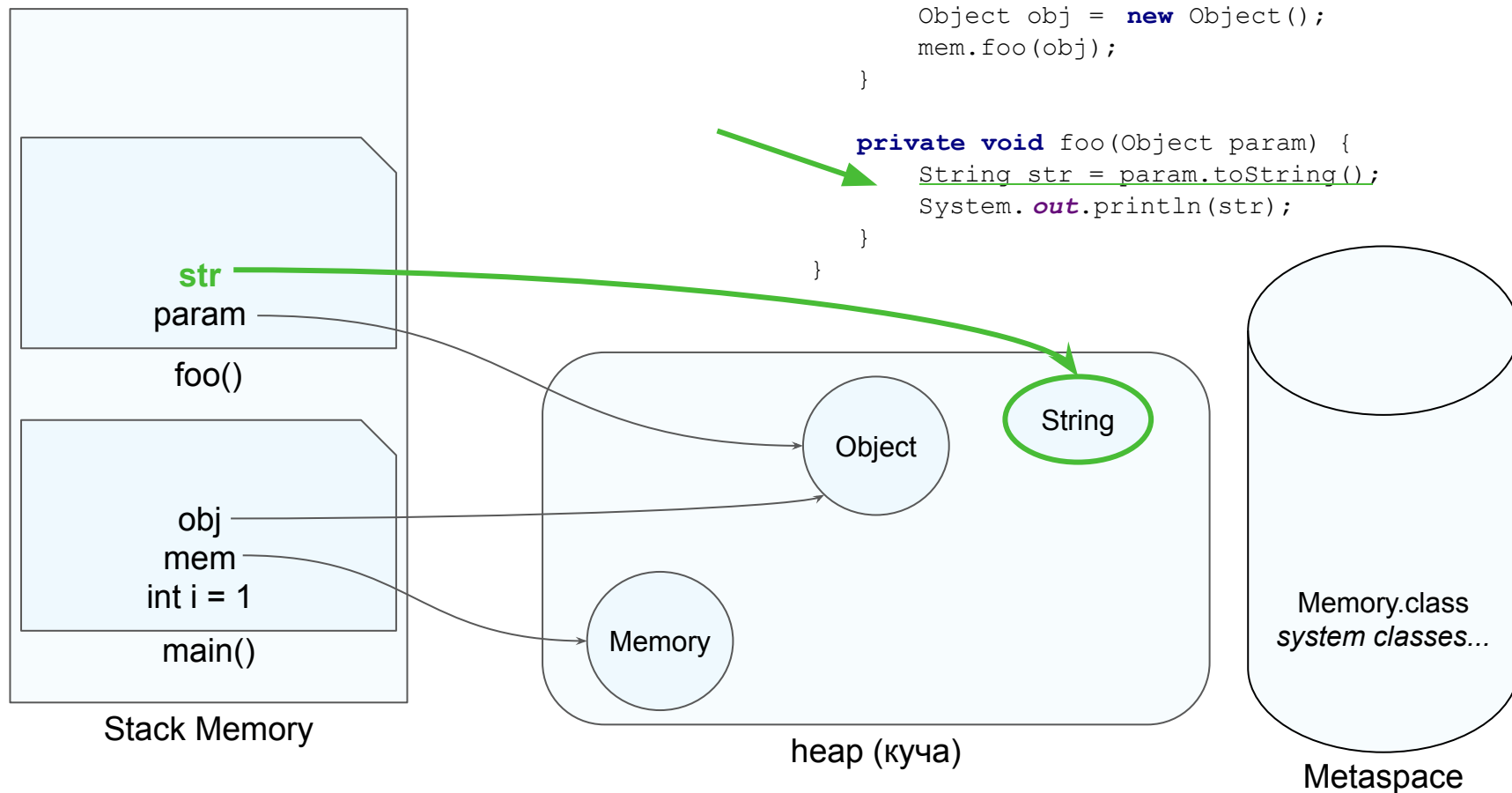
Области памяти

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



Области памяти

```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```

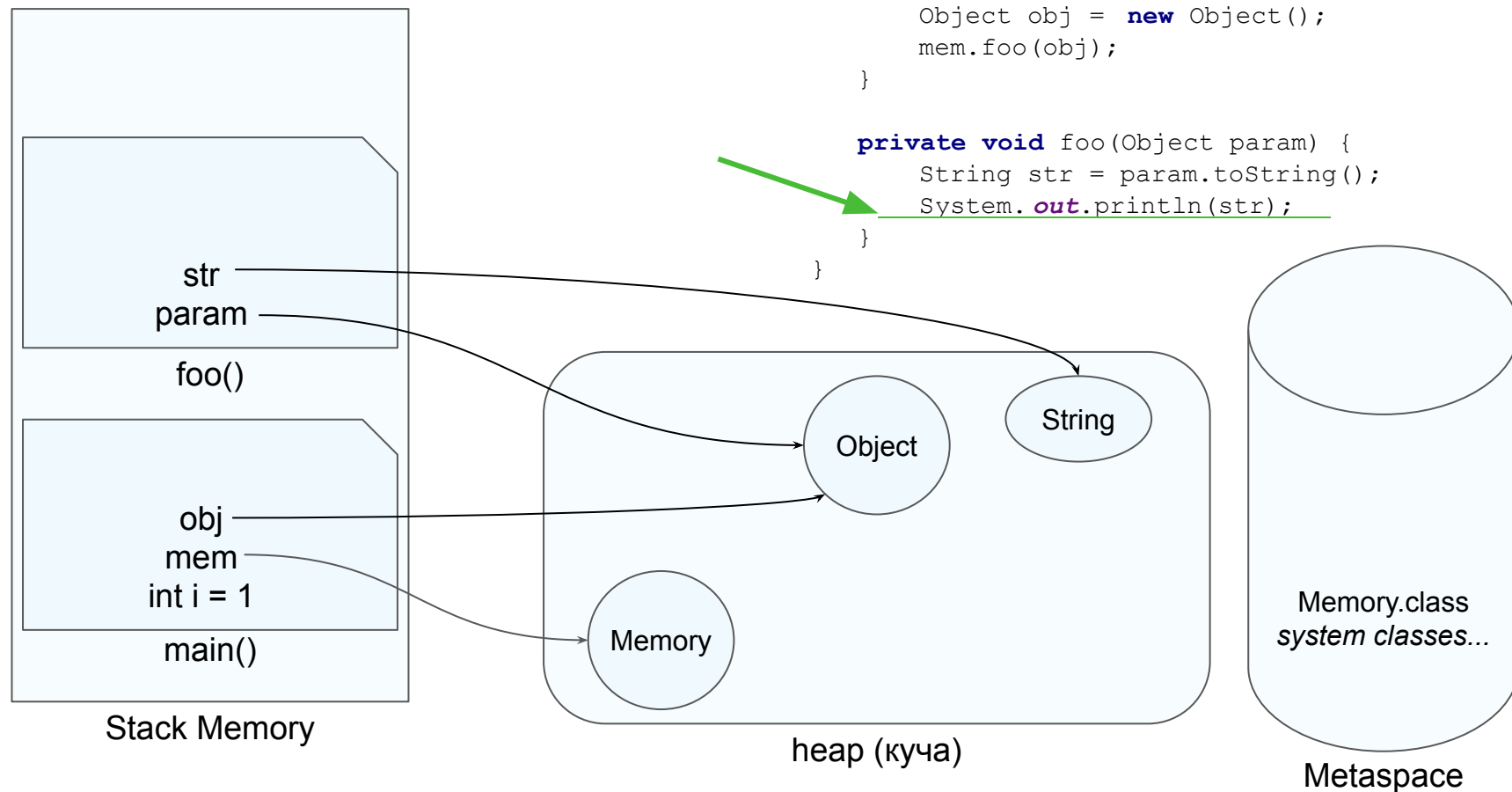


Области памяти

Что произойдёт в этой строчке?

Жду предположений в чате.

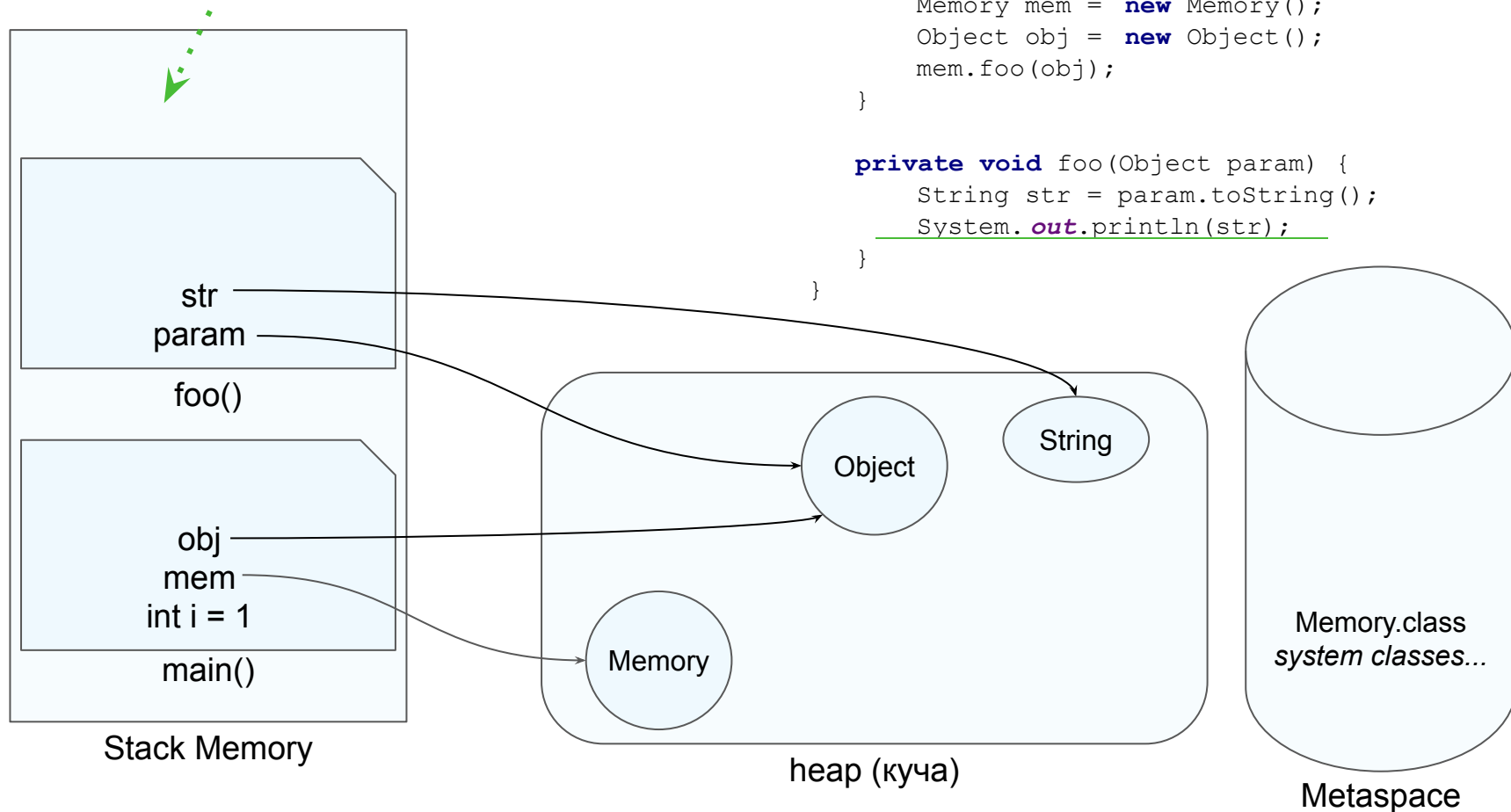
```
public class Memory {  
  
    public static void main(String[] args) {  
        int i = 1;  
        Memory mem = new Memory();  
        Object obj = new Object();  
        mem.foo(obj);  
    }  
  
    private void foo(Object param) {  
        String str = param.toString();  
        System.out.println(str);  
    }  
}
```



Области памяти

Что произойдёт в этой строчке?

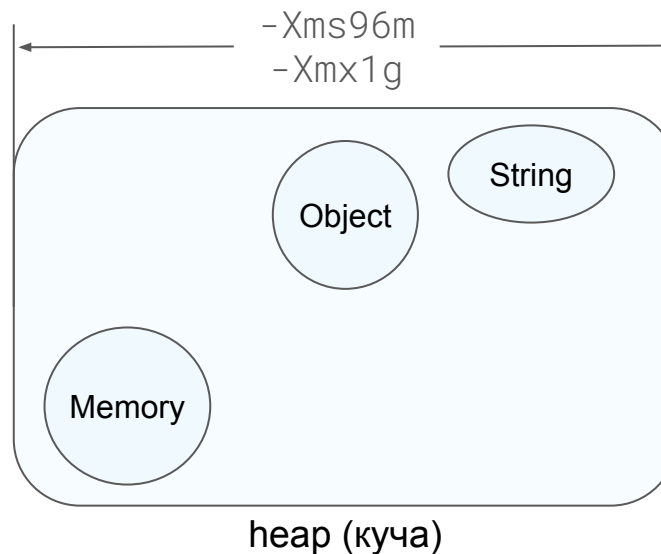
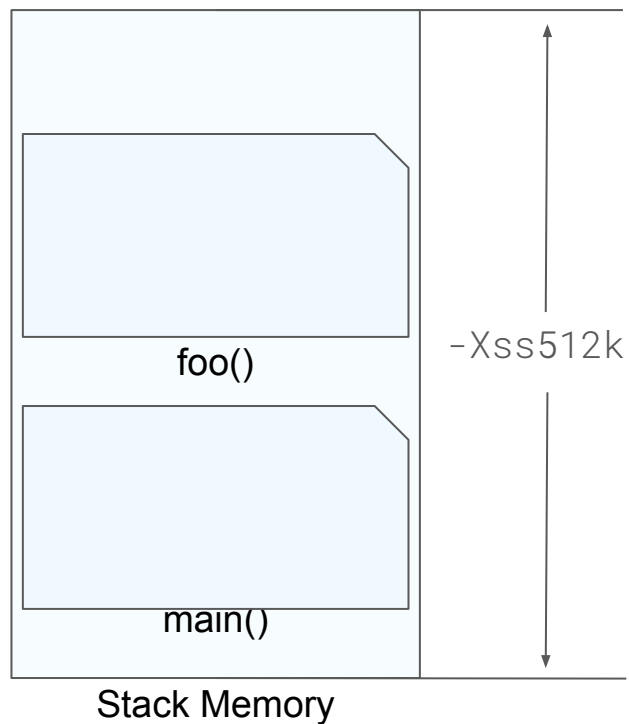
Создастся новый фрейм в стеке, куда передадим ссылку на *str*



Области памяти: стэк и куча

Размер можно отрегулировать:

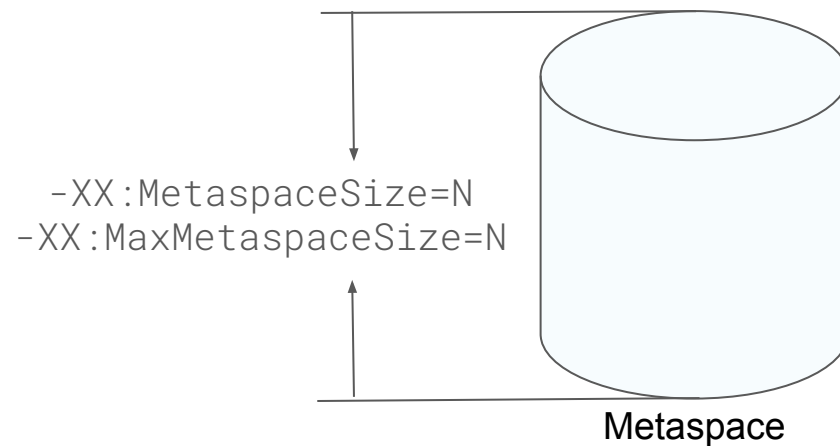
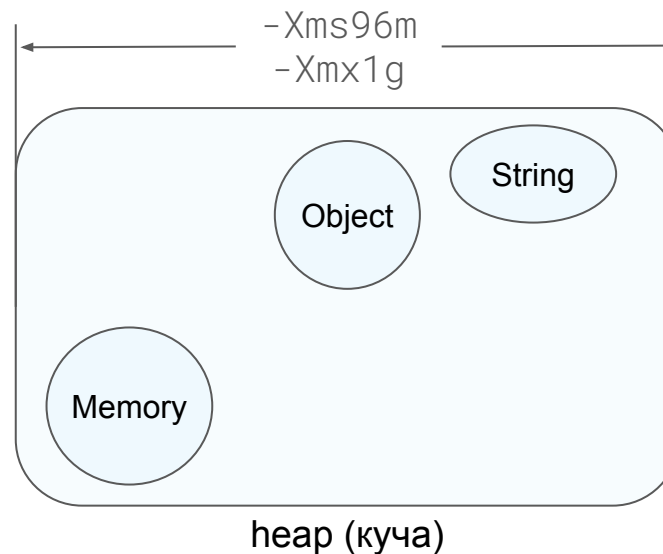
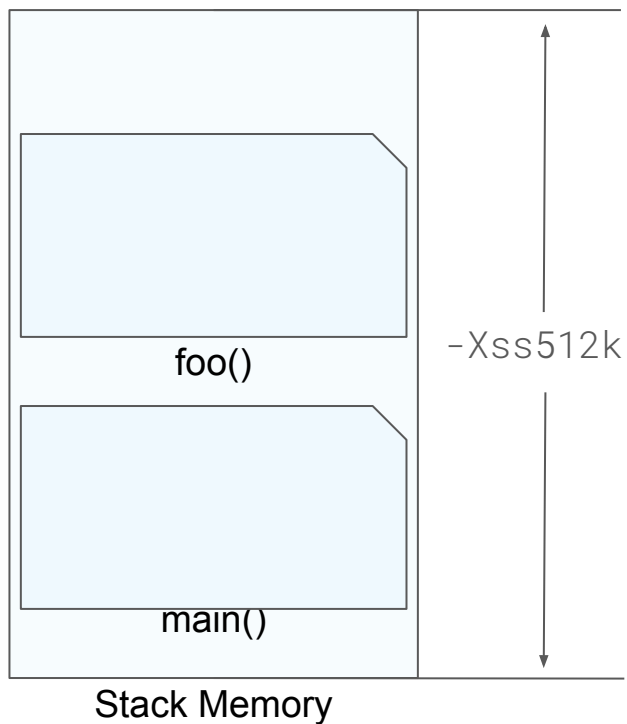
- Xss512k - размер стэка
- Xms96m - изначальный размер кучи
- Xmx1g - максимальный размер кучи



Области памяти: стэк и куча

Размер можно отрегулировать:

- Xss512k - размер стэка
- Xms96m - изначальный размер кучи
- Xmx1g - максимальный размер кучи



Как работает JVM

1. ClassLoaders
2. Runtime Data Area
- 3. Execution Engine**

Движок выполнения

Что происходит во время выполнения?

(Когда JVM получает .class файл)

1. Код выполняется строка за строкой
2. Методы компилируются в машинный код прямо во время выполнения



Движок выполнения

Что происходит во время выполнения?

(Когда JVM получает .class файл)

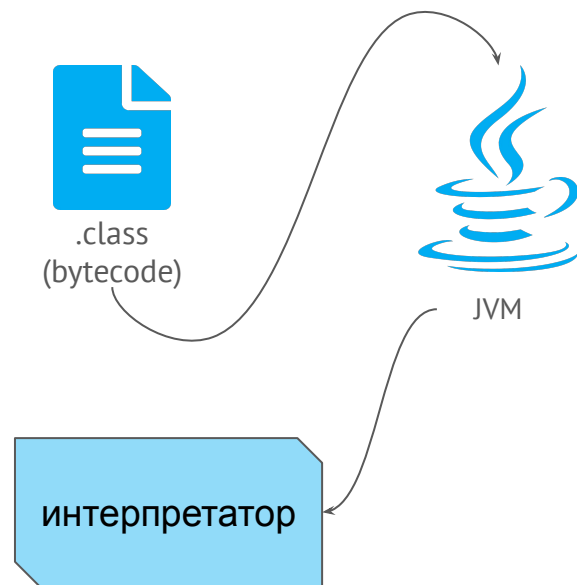
1. Код выполняется строка за строкой
2. Методы компилируются в машинный код прямо во время выполнения



Движок выполнения

Интерпретатор

- байт-код в .class файле интерпретирует строка за строкой. Затем выполняет



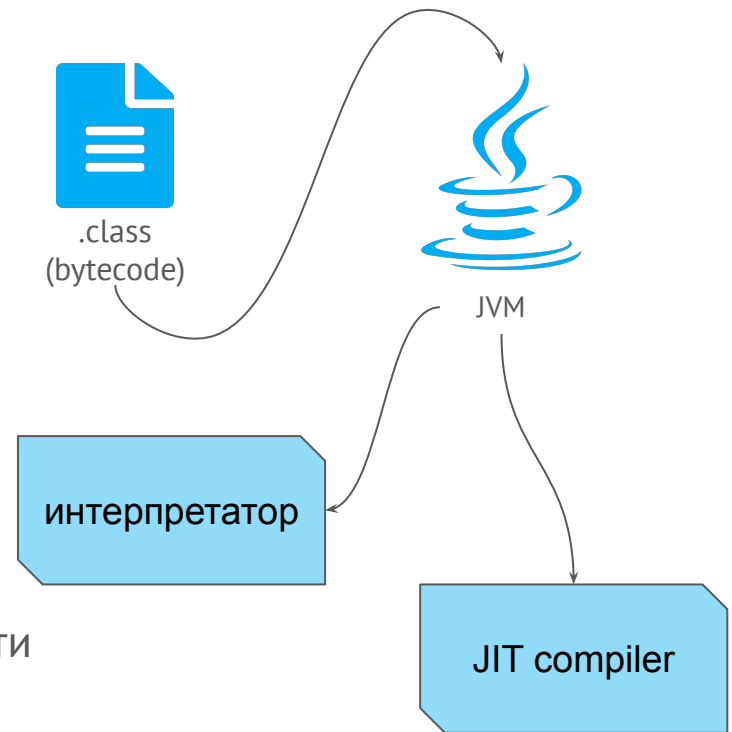
Движок выполнения

Интерпретатор

- байт-код в .class файле интерпретирует строка за строкой. Затем выполняет

Just In Time (JIT) компилятор

- Используется для повышения эффективности интерпретатора
 - Выполнение машинного кода быстрее, чем интерпретация строка за строкой
 - Сама компиляция в машинный код дольше, чем интерпретация



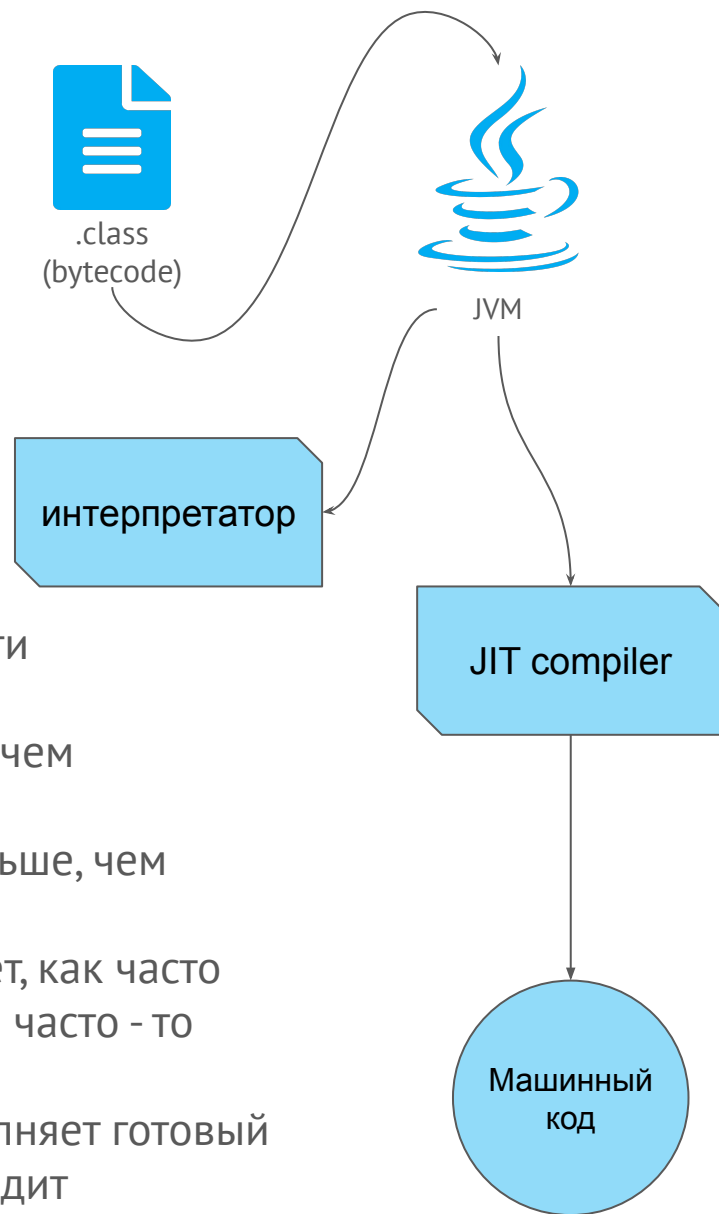
Движок выполнения

Интерпретатор

- байт-код в .class файле интерпретирует строка за строкой. Затем выполняет

Just In Time (JIT) компилятор

- Используется для повышения эффективности интерпретатора
 - Выполнение машинного кода быстрее, чем интерпретация строка за строкой
 - Сама компиляция в машинный код дольше, чем интерпретация
- Чтобы ускорить интерпретатор, он проверяет, как часто вызывается какой-то метод, и если слишком часто - то компилирует его код и сохраняет в кэше. При последующих вызовах метода он выполняет готовый машинный код, взятый из кэша, что происходит значительно быстрее



Как работает JVM

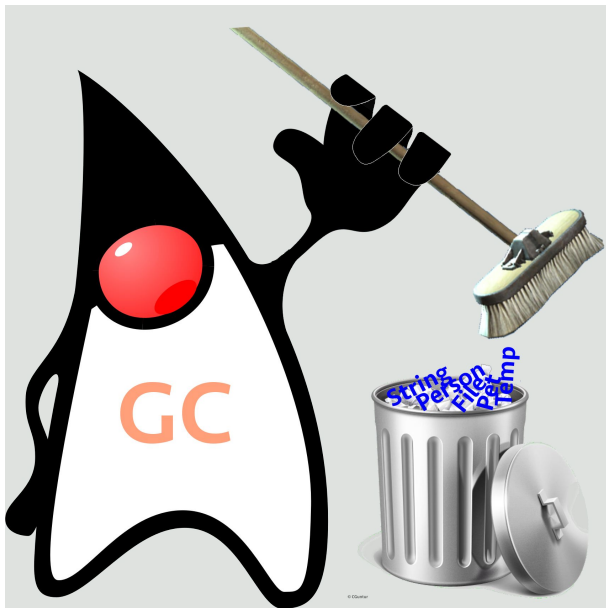
1. ClassLoaders
2. Runtime Data Area
3. Execution Engine
 - **Garbage Collection**

Движок выполнения: Сборка мусора



Сборка мусора

Периодически **собирает объекты из памяти (хипа)**, которые больше не используются



Сборка мусора: как?



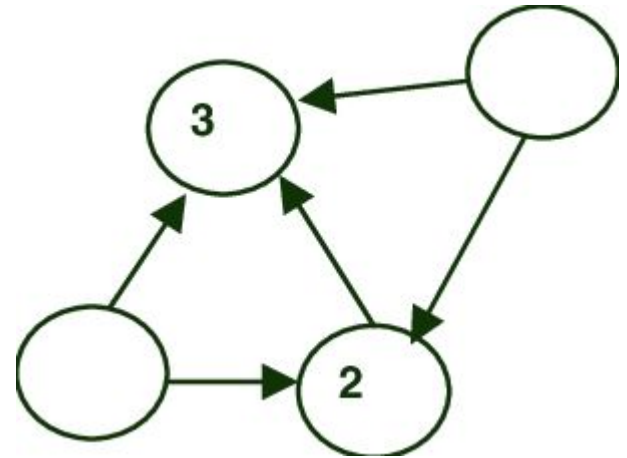
Как определить объекты,
которые больше не
используются?

Сборка мусора: как?

Как определить объекты, которые больше не используются?

Два основных метода:

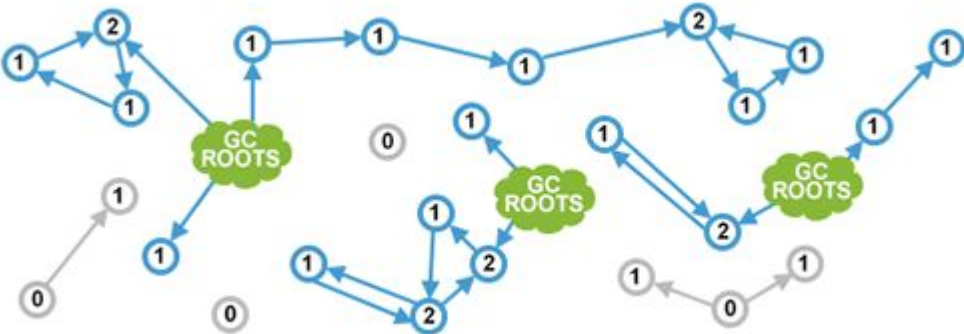
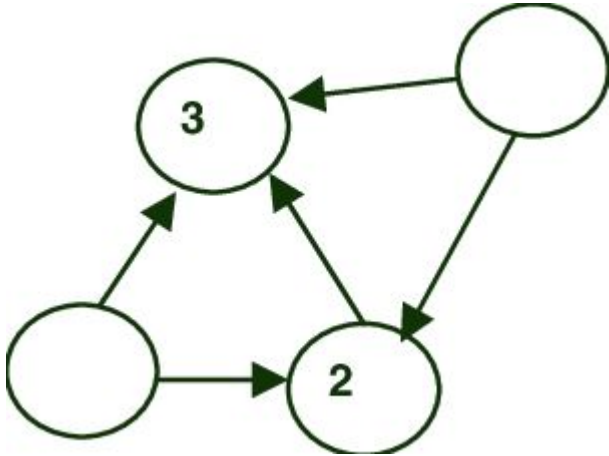
- подсчет ссылок (reference counting)



© 2006 The Authors

Два основных метода:

- подсчет ссылок (reference counting)
- **обход графа достижимых объектов (mark-and-sweep, copying collection)**



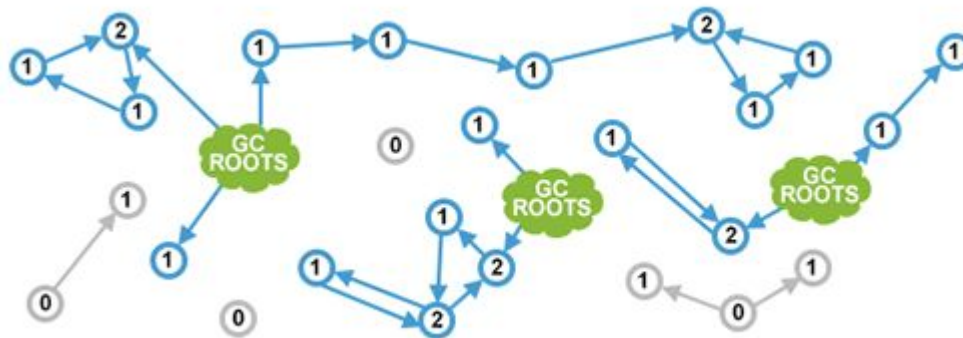
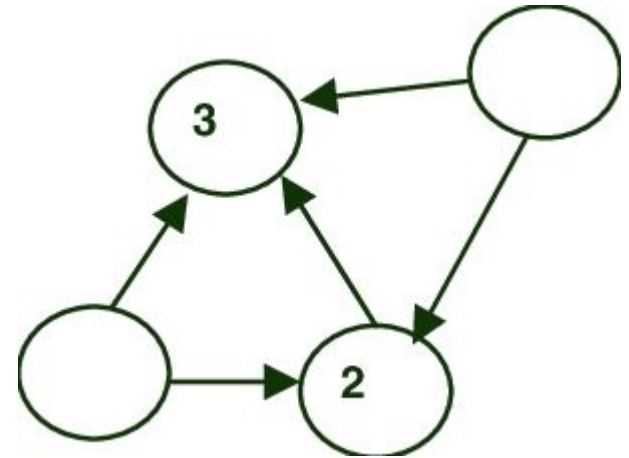
Сборка мусора: как?

Как определить объекты, которые больше не используются?

Два основных метода:

- подсчет ссылок (reference counting)
- обход графа достижимых объектов (mark-and-sweep, copying collection)

Первый подход испытывает трудности с циклическими ссылками - **В основном используется второй.**

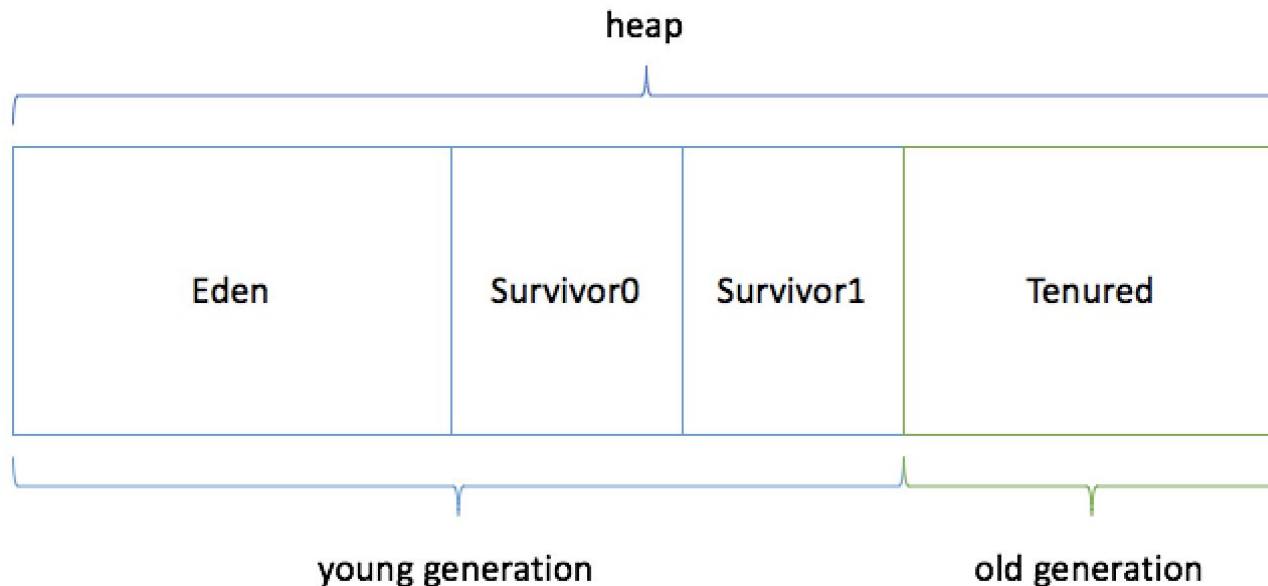


Сборка мусора: как?

Недостижимые объекты удаляются.

А достижимые обычно *группируются по времени жизни* (эти группы называют *поколения*) - чем дольше объект живёт, тем реже проверяют, нужно ли его удалить

Сформулируйте в чат, почему?



Сборка мусора: когда?

Обычно **для сборки мусора происходит приостановка программы** - *Stop The World* пауза - это полная остановка потоков программы для безопасной сборки мусора

(Частый вопрос на собеседах! 😊)

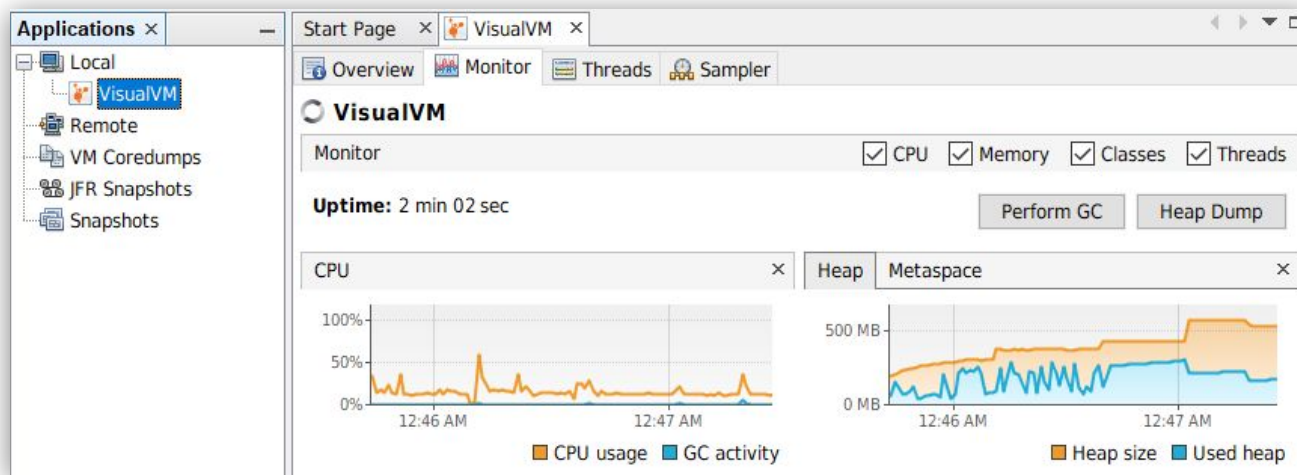


VisualVM

VisualVM

Проследить работу GC и вообще использование памяти можно с помощью утилиты **VisualVM**.

Она поможет при выполнении оптимизаций.





Итоги

Итоги

- Части JVM
 - *Что помните? (напишите в чат)*



Итоги

- **Части JVM**
 - Подсистема загрузки классов
 - Области памяти
 - Движок выполнения

Итоги

- Части JVM
 - Подсистема загрузки классов
 - **Области памяти**
 - *Какие области памяти помните?*
 - Движок выполнения



Итоги

- **Части JVM**
 - Подсистема загрузки классов
 - **Области памяти**
 - **Stack** -Xss512k
 - **Heap** -Xmx512m
 - **Metaspace** -XX:MaxMetaspaceSize=1g
 - **Движок выполнения**

Итоги

- Части JVM

- Подсистема загрузки классов
- Области памяти
 - Stack -Xss512k
 - Heap -Xmx512m
 - Metaspace -XX:MaxMetaspaceSize=1g
- **Движок выполнения**
 - *Перечислите в чате компоненты движка выполнения?*



Итоги

- **Части JVM**

- Подсистема загрузки классов
- Области памяти
 - **Stack** -Xss512k
 - **Heap** -Xmx512m
 - **Metaspace** -XX:MaxMetaspaceSize=1g
- **Движок выполнения**
 - Интерпретатор
 - JIT компилятор
 - Сборщик мусора

Итоги

- Части JVM

- Подсистема загрузки классов
- Области памяти
 - Stack -Xss512k
 - Heap -Xmx512m
 - Metaspace -XX:MaxMetaspaceSize=1g
- Движок выполнения
 - Интерпретатор
 - JIT компилятор
 - Сборщик мусора

- *Как посмотреть в runtime работу сборщика мусора, размер занимаемой памяти?*



Итоги

- **Части JVM**

- Подсистема загрузки классов
- Области памяти
 - **Stack** -Xss512k
 - **Heap** -Xmx512m
 - **Metaspace** -XX:MaxMetaspaceSize=1g
- Движок выполнения
 - Интерпретатор
 - JIT компилятор
 - Сборщик мусора

- **VisualVM**

- Отслеживает работу сборщика мусора, размер занимаемой памяти в runtime



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

Спасибо!

Good

Job

**Задавайте вопросы и
пишите отзыв о лекции!**

Григорий Вахмистров