7

Конструирование классов

ОБСУЖДАЕМЫЕ ВОПРОСЫ

1. Проблемная область приложения
2. Объектно-ориентированное программирование (ООП)
3. Описание класса и реализация класса
4. Модификаторы доступа и ключевые слова в декларации класса
5. Создание объектов (экземпляров) некоторого класса
6. Порядок создания объектов
7. Методы и их вызов
8. Конструктор
9. Переменная **CLASSPATH**

ПРОБЛЕМНАЯ ОБЛАСТЬ ПРИЛОЖЕНИЯ

При рассмотрении проблемной области, объекты и действия в которой должно моделировать приложение, мы обычно видим довольно сложную картину, когда невозможно описать состояние объекта с помощью единственного примитивного типа данных . Состояние любого объекта описывает достаточно большой набор параметров и свести все многообразие состояния объектов к переменным примитивного типа невозможно.

На помощь нам приходят классы, как структуры более высокого уровня. Класс является хотя и более сложным типом данных, чем примитивный тип, но это также тип данных. То есть класс является описанием объектов, которые на основе этого описания будут создаваться. Можно представить класс как проект типового дома, а объекты это конкретные дома, которые по этому проекту строятся. Несмотря на то, что проект типовой, сами дома обладают индивидуальными качествами, например адресом.

ООП

Класс можно представить как шаблон, на основе которого создаются конкретные объекты.

Объектно-ориентированное программирование (ООП) и такой язык, как C++, в свое время внесли качественные изменения в технику разработки приложений.
Википедия дает ООП следующее определение: «Объектно-ориентированное программирование (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов (либо, в менее известном варианте языков с прототипированием, — прототипов)».

Поскольку язык Java является объектно-ориентированным языком, мы в дальнейшем будем постоянно использовать следующие основные принципы ООП:

- Абстракция
- Инкапсуляция
- Наследование
- Полиморфизм

ОПИСАНИЕ КЛАССА

Представим себе в виде примитивной диаграммы будущий класс, который в качестве объектов будет описывать разнообразных кошек и котов:

Cat
+ name : String + weight : int + age : int
+ eat() + sleep() + printDescription()

Атрибуты определяют **состояние** будущих объектов.

Методы определяют **поведение** будущих объектов.

В верхнем прямоугольнике мы видим название класса – `Cat`.
Во втором прямоугольнике описаны атрибуты с названиями и типами данных. Символ **+**, стоящий слева показывает, что атрибуты являются общедоступными, то есть к ним можно обращаться из других классов.
В третьем прямоугольнике описаны методы, которые определяют поведение будущих объектов.

РЕАЛИЗАЦИЯ КЛАССА

При создании класса, описывающего группу объектов, мы всегда выбираем подмножество атрибутов и методов поведения действительных объектов. Например, в нашем классе `Cat` мы опустили породу кошки, окрас и много чего еще.

```
public class Cat {
  public String name;
  public int weight;
  public int age = 1;

  public void eat() {
    // кошка ест
   }
  public void sleep() {
    // кошка спит
   }
  public void printDescription(){
     System.out.println( "name = " + name
                         + " weight = "+ weight + " age = "+ age);
   }
}
```

Создавая класс, мы тем самым создаем абстрактное представление о действительных объектах.
Абстрагирование неизбежно.

МОДИФИКАТОРЫ ДОСТУПА И КЛЮЧЕВЫЕ СЛОВА В ДЕКЛАРАЦИИ КЛАССА

Класс может иметь модификатор доступа `public`,
а также модификатор доступа может быть опущен (пакетный доступ).

ЗАМЕЧАНИЕ: При описании вложенных (и только вложенных) классов можно еще использовать и модификаторы доступа `protected` и `private`.

Переменные, описанные в классе следующим образом:

```
public String name;
public int weight;
public int age;
```

являются атрибутами описывающими состояние объекта, называются переменными экземпляра и создаются в куче, а не в стеке. Базовый синтаксис декларации атрибута:

```
<модификатор_доступа>* <тип> <название> [ = <инициализирующее_значение>];
```

<тип> и **<название>** являются обязательными. То, что стоит в квадратных скобках (в данном случае инициализация) – необязательно.

СОЗДАНИЕ ОБЪЕКТОВ (ЭКЗЕМПЛЯРОВ) НЕКОТОРОГО КЛАССА

Конкретный кот может иметь для переменной **name** значение **Perl**, для переменной **weight** значение **12**, а для переменной **age** значение **3**. Другой кот может иметь имя **Fuzzy**, вес **10** кг, а возраст **2** года.

```
Perl
12
3
```

```
Fuzzy
10
2
```

Создание объектов осуществляется с помощью оператора **new**. Переменная, которая будет ссылаться на объект типа **Cat**, декларируется таким же образом, что и любая примитивная переменная.

```
Cat cat;
cat = new Cat();
```

Второй оператор выполняет ее инициализацию. Однако мы можем сразу описать переменную и проинициализировать ее.

```
Cat cat = new Cat();
```

ПРИМЕР СОЗДАНИЯ ОБЪЕКТОВ

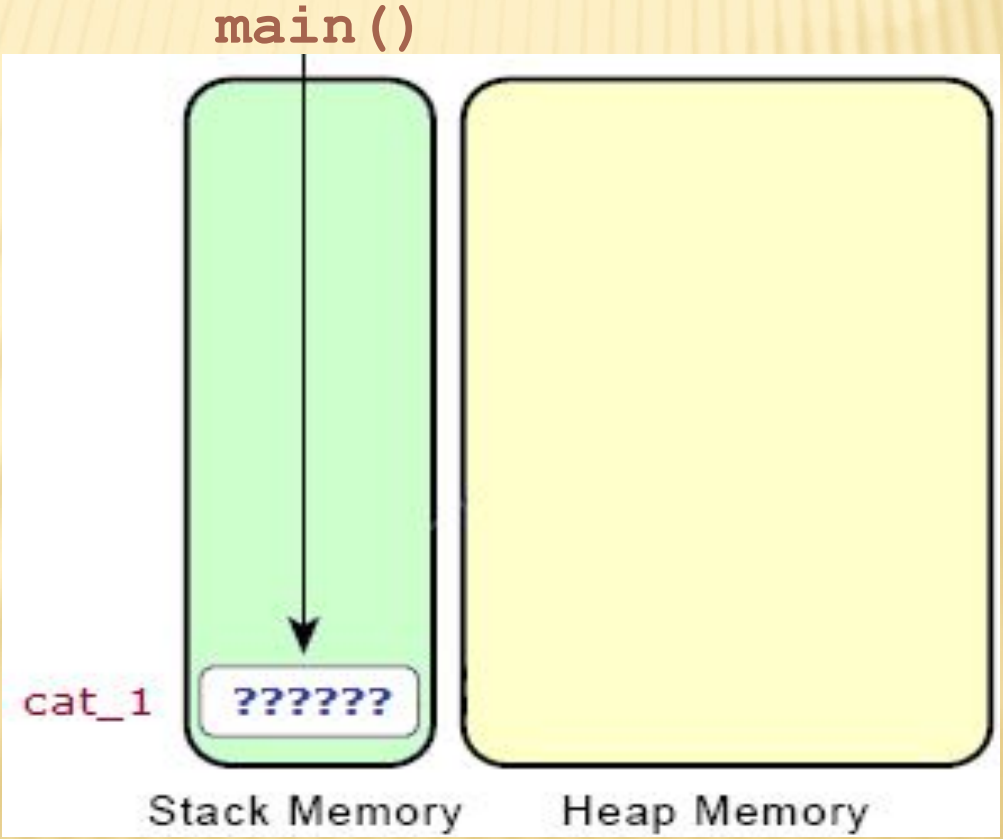
Создадим два объекта, описывающие двух котов в отдельном классе `TestCats`:

```
public class TestCats {

 public static void main(String[] args) {

   Cat cat_1 = new Cat();
    cat_1.name = "Perl";
    cat_1.weight = 12;
    cat_1.age = 3;
   Cat cat_2;
    cat_2 = new Cat();
    cat_2.name = "Fuzzy";
    cat_2.weight = 10;
    cat_2.age = 2;
  }
}
```

Переменная `cat_1` имеет тип `Cat`. Так как она создается в методе `main()`, она является локальной или стековой, то есть ячейка памяти под нее по-прежнему выделяется в стеке. Однако в отличие от примитивных переменных она будет содержать не конкретное примитивное значение, а ссылку на объект, созданный в куче. Объект создается с помощью оператора `new` и вызова конструктора класса (в данном случае `Cat()`).

ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ: ШАГ 1

1. Сначала в стеке создается ячейка под переменную `cat_1`. Это аналогично декларации переменной, без выполнения ее инициализации (`Cat cat_1;`):



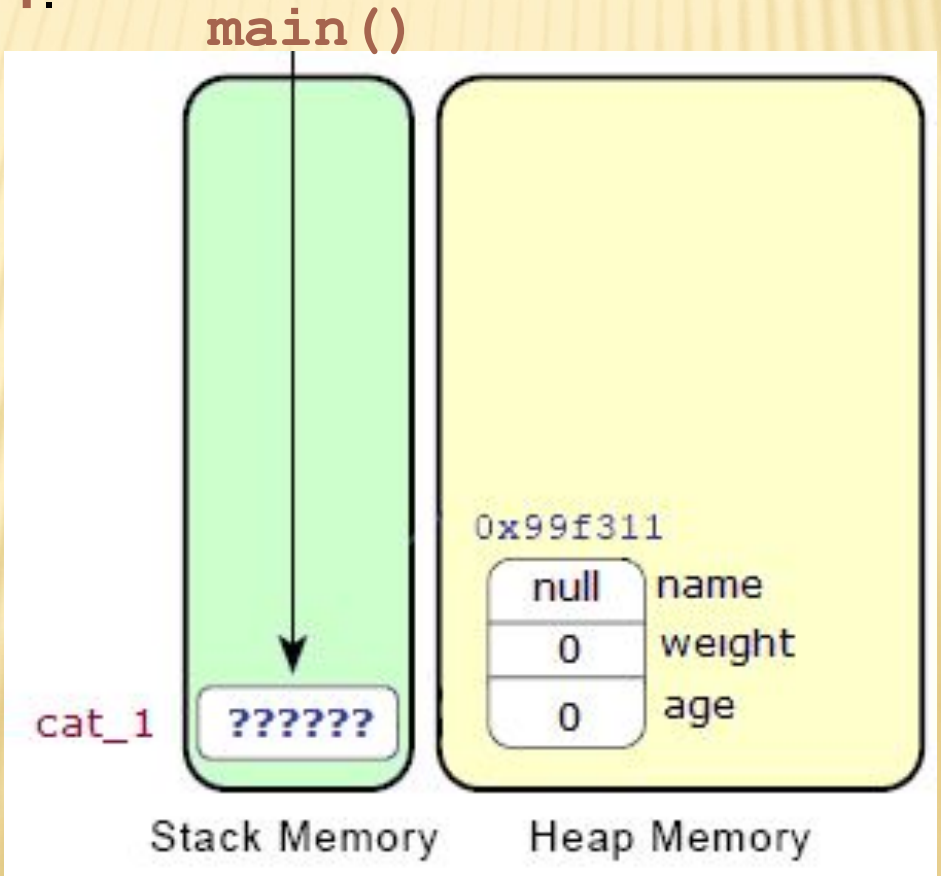
ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ : ШАГ 2

2. Затем в куче создается объект типа `Cat` с переменными экземпляра. Как мы помним, инициализацию локальных переменных проверяет компилятор. Проверить инициализирован ли явно атрибут класса весьма трудно, так как это может быть сделано, например, в конструкторе вызовом метода. Поэтому в атрибуты класса в обязательном порядке инициализируются системой неявно (по умолчанию), даже если имеется явная инициализация:

- Числовые переменные получают нулевые значения соответствующих типов.
- Символьные переменные инициализируется значением `'\u0000'`.
- Переменные типа `boolean` инициализируются значениями **false**.
- Ссылочные переменные получают значение **null**.

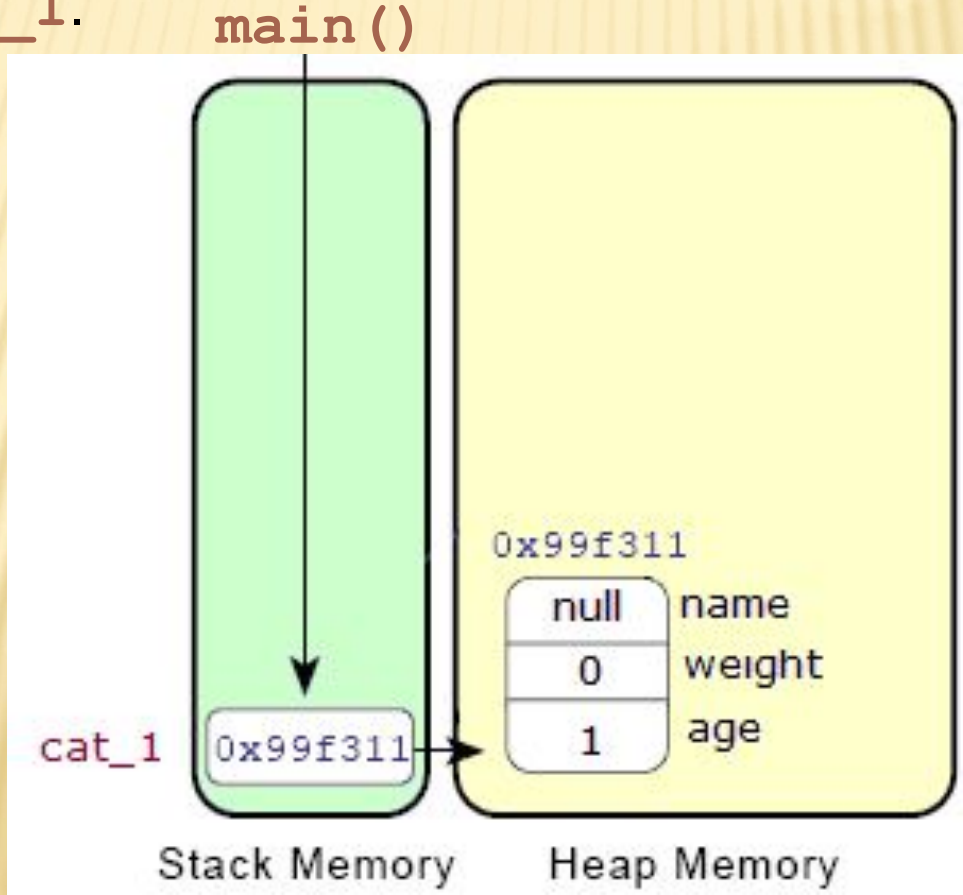
ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ : ШАГ 2 (ПРОДОЛЖЕНИЕ)

В нашем классе `Cat` атрибут `public int age = 1;` явно инициализируется значением **1**. Тем не менее, сначала он будет проинициализирован значением **0**, и только на следующем шаге он получит значение **1**.



ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ : ШАГ 3

3. Осуществляется явная инициализация переменных. В нашем случае в переменную `age` заносится значение `1`. Затем выполняется конструктор. И, наконец, ссылка на объект помещается в переменную `cat_1`.



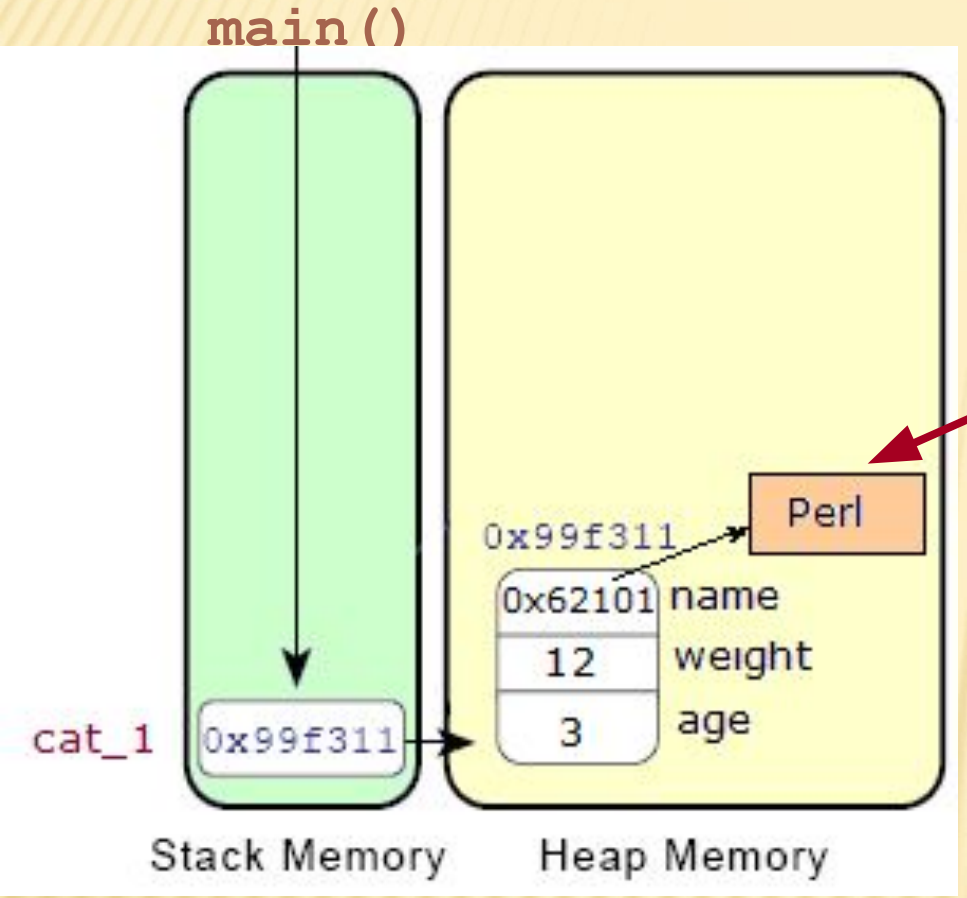
ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ : ШАГ 4

4. Поскольку переменные экземпляра в классе `Cat` имеют модификатор доступа `public`, доступ к ним возможен извне и напрямую, с использованием точечной нотации, что позволяет нам выполнить явную инициализацию переменных экземпляра:

```
cat_1.name = "Барсик";
cat_1.weight = 12;
cat_1.age = 3;
```

Так мы установили конкретные значения нашей кошке по имени «Жемчужина».

ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ : ШАГ 4 (ПРОДОЛЖЕНИЕ)




```
cat_1.name = "Perl";
cat_1.weight = 12;
cat_1.age = 3;
```

Обратите внимание, что переменная **name**, содержащая имя кота, на самом деле содержит адрес ссылки на другой объект в куче. И все потому, что **String** это класс.

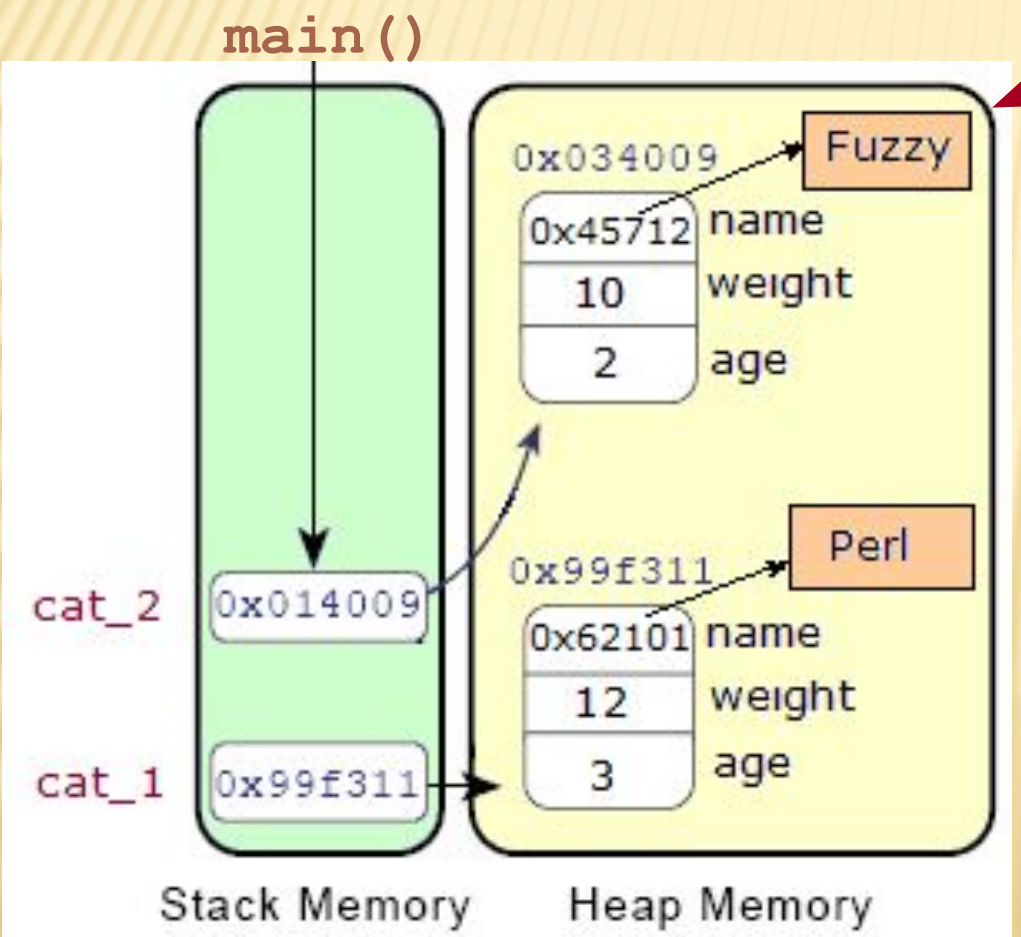
И на самом деле переменная **name** содержит ссылку на объект класса **String**, тогда как примитивные переменные **weight** и **age** содержат конкретные значения.

ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ : ШАГ 5

5. Теперь мы аналогично создаем новый объект, описывающий кота по имени «Пушистик»:

```
Cat cat_2;
cat_2 = new Cat();
cat_2.name = "Fuzzy";
cat_2.weight = 10;
cat_2.age = 2;
```

Результаты отличаются незначительно

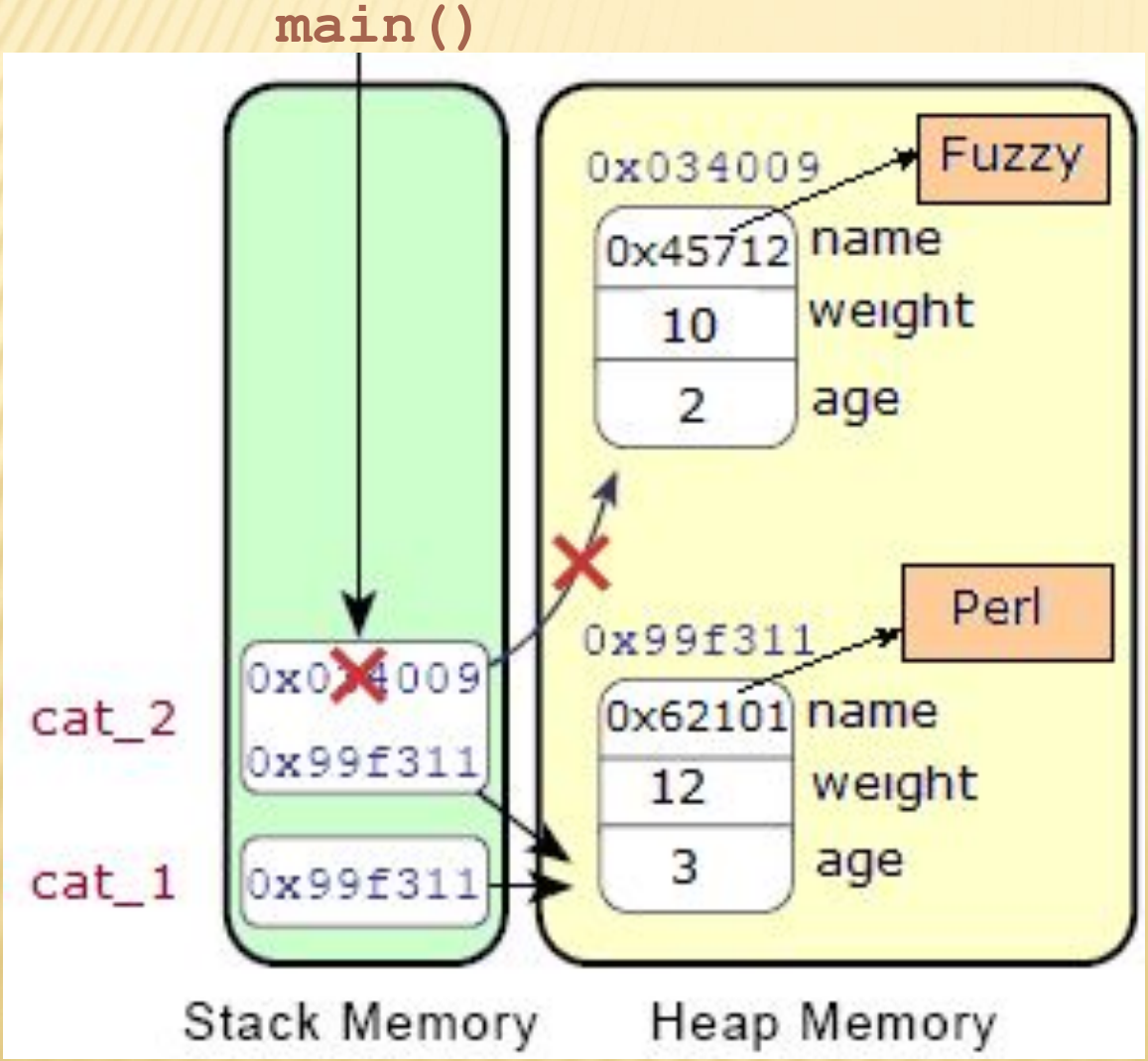


ЗАМЕЧАНИЕ: При этом мы разнесли в два оператора описание переменной и создание объекта, то есть не сразу проинициализировали переменную `cat_2`:

```
Cat cat_2;
cat_2 = new Cat();
```

ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ : ШАГ 6

6. Если мы добавим в конец метода `main()` оператор: `cat_2 = cat_1;`



мы тем самым переменной `cat_2` присвоим ссылку на «Жемчужину» и теперь будем иметь на этот объект две ссылки. А объект «Пушистик» больше ссылки не имеет, в связи с чем, он стал недоступен.

Попросту говоря, этот объект стал мусором, и сборщик мусора имеет полное право его удалить.

ПОРЯДОК СОЗДАНИЯ ОБЪЕКТОВ (ОБОБЩЕНИЕ)

Итак, обобщив механизм инициализации переменной с помощью оператора new, получим следующий набор шагов:

1. В куче выделяется пространство для объекта, и в частности, для его атрибутов.
2. Атрибуты инициализируются значениями по умолчанию.
3. Выполняется явная инициализация атрибутов.
4. Выполняется конструктор.
5. Ссылка на новый объект, созданный оператором `new`, помещается в соответствующую ссылочную переменную.

ВЫЗОВ МЕТОДОВ

Обращение к методам объекта из объекта другого класса также выполняется с помощью точечной нотации:

```
cat_1.printDescription();
```

Обратите внимание, что таким образом мы даем знать методу с состоянием какого объекта он должен работать. То есть вызов `cat_1.printDescription()` выводит информацию о «Жемчужине», а вызов `cat_2.printDescription()` выводит информацию о «Пушистике».

Базовый синтаксис метода:

```
<модификатор_доступа>* <возвращаемый_тип> <название>( <параметр>* ) {
    <оператор>*
  }
```

Возвращаемый тип, название метода, круглые скобки для параметров, а также фигурные скобки для тела метода обязательны.
Список параметров и тело могут быть пустыми.

ВЫЗОВ МЕТОДОВ (ПРОДОЛЖЕНИЕ)

В некоторых языках программирования функции, возвращающие значение, и процедуры – не возвращающие, являются разными программными компонентами. У нас имеются только – методы:

- Если метод возвращает значение примитивного типа или ссылку на объект (по сути, является функцией), то он должен иметь, хотя бы один, оператор `return`, возвращающий значение описанного типа.
- Если метод не возвращает значения (по сути, является процедурой), он должен возвращать значение типа `void` (пустое значение) и наличие оператора `return` не требуется.

Оператор `return` предназначен для возврата управления из выполняющегося метода в вызывающий метод.

ВЫЗОВ МЕТОДОВ (ПРОДОЛЖЕНИЕ)

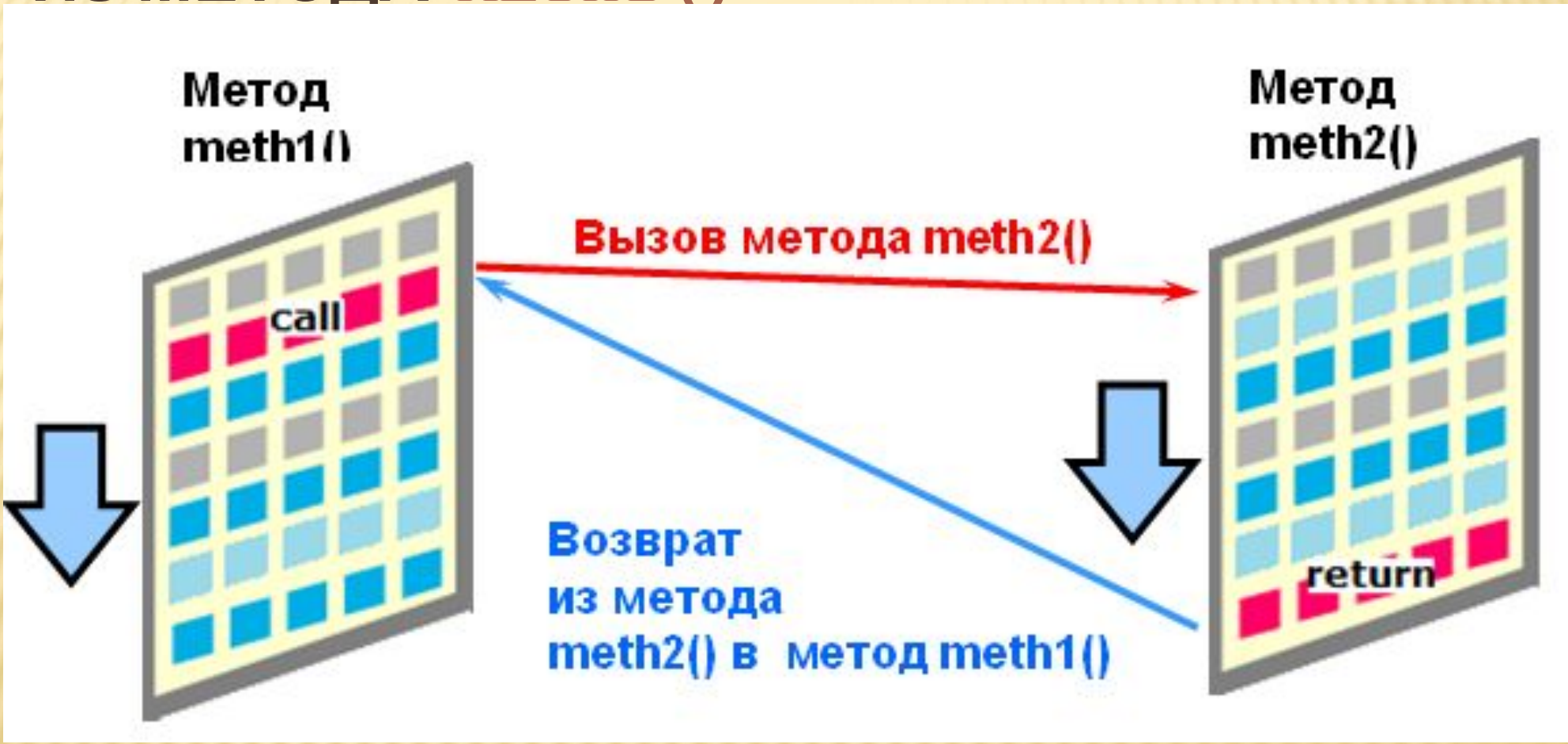
Если метод не возвращает значений, то есть его возвращаемое значение имеет тип `void`, то необязательный в этом случае оператор `return` не содержит возвращаемого значения. Если метод возвращает значение отличное от `void`, то `return` обязательно должен возвращать значение данного типа.
Возвращаемым значением может быть литерал, значение переменной или выражения:

```
return (a + 12) / getAge();
```

В этом случае сначала будет выполнено выражение, а затем результат его выполнения будет передан в вызывающий метод.

Хотя это считается не слишком хорошим стилем программирования, вы можете в одном методе задействовать несколько операторов `return`, с использованием операторов управления, естественно.

ВЫЗОВ МЕТОДА `METH2()` ИЗ МЕТОДА `METH1()`



Вызов метода может осуществляться из другого метода данного класса, тогда метод вызывается для текущего объекта и префикс в виде ссылочной переменной не нужен:

```
printDescription();
```

МЕТОДЫ С ПАРАМЕТРАМИ

Если в методе описаны формальные параметры, то при вызове надо указывать фактические параметры, значение, которых будет скопировано в формальные параметры метода. Фактическим параметром может быть не только литерал, но и инициализированная переменная (возможно ссылочная).

```
int k = 12;
double m;
. . .
m = mult( 23.7, k );
```

Сам метод в таком случае может быть следующим:

```
public double mult( double p_multiplier1, double p_multiplier2 ) {
   return p_multiplier1 * p_multiplier2;
}
```

ЗАМЕЧАНИЕ: Обратите внимание на то, что второй фактический параметр представлен переменной целого типа, а формальный параметр имеет тип `double`. То есть в таком случае будет выполняться неявное приведение типа .

ПЕРЕДАЧА ПАРАМЕТРОВ

- В языке Java при вызове методов передача значений фактических параметров в формальные параметры осуществляется копированием – т.е. по значению. Т.о. изменение значения формального параметра не влияет на значение фактического параметра.
- НО: При передаче *ссылок на объекты* осуществляется копирование *ссылки*. *После такого копирования, и фактический параметр, и формальный параметр, ссылаются на один и тот же объект.*
- В Java *НЕТ возможности задать умолчания* для последних параметров в списке формальных, как это возможно в C++ и Delphi.

```
// неверно – задать умолчания для параметров невозможно
public void doSomeJob( String name, int v = 2, int t = 0) {
    …
}
```

КОНСТРУКТОР

Наверное, у вас уже возник вопрос по поводу конструктора.
Что это такое и откуда он взялся? Мы ведь конструктор не писали.
Конструктор это набор инструкций по инициализации объектов, очень похожий на метод, который называется в точности так же, как класс.
Однако, строго говоря, конструктор методом не является и НЕ МОЖЕТ иметь возвращаемого значения.
Некоторые правила, связанные с конструкторами.

- *Каждый класс обязан иметь конструктор.*
- *Если в классе никакого конструктора явно не написано, то система автоматически создает конструктор без параметров, который называется конструктором по умолчанию.*
- *Если в классе явно описан какой-либо конструктор с параметрами, то конструктор по умолчанию системой не создается .*

КОНСТРУКТОР (ПРОДОЛЖЕНИЕ)

Конструкторы могут иметь такие модификаторы доступа, как **public**, **protected**, **private**, или модификатор может отсутствовать.

```
package zoostore.model;
public class Cat {
  public String name;
  public int weight;
  public int age = 1;

  public Cat() {
      weight = 10;
   }
  public void eat() {
    // . . .
   }
  public void sleep() {
    // . . .
   }
  public void printDescription(){
     System.out.println( "name = " + name
      + " weight = "+ weight + " age = "+ age);
   }
}
```

Однако при описании конструктора можно использовать только модификаторы доступа, а такие ключевые слова, как **abstract**, **final**, **native**, **static** или **synchronized** - нельзя. Конструктор, объявленный как **private**, можно вызвать только из другого метода этого же класса, но не извне.

Обратите внимание на первый оператор **package**

ЗАМЕЧАНИЕ: Конструктор не имеет возвращаемого значения. Если вы по ошибке укажете для конструктора возвращаемое значение, то ваш конструктор будет рассматриваться не как конструктор, а как обычный метод.

КОМПИЛЯЦИЯ И ВЫПОЛНЕНИЕ ПРИЛОЖЕНИЯ TESTCATS

Теперь компиляция и выполнение программы потребуют некоторых дополнительных усилий. Набор выполняемых для этого команд требует дополнительного внимания:

```
C:
cd \
cd  sources\demo\d_07
SET CLASSPATH=./classes

javac -d ./classes ./src/zoostore/model/Cat.java

javac -d ./classes ./src/zoostore/test/TestCats.java
java zoostore.test.TestCats
```

С помощью команды `SET CLASSPATH=./classes` мы устанавливаем значение переменной окружающей среды операционной системы, `CLASSPATH`, которая указывает, где находятся классы приложения.

ПЕРЕМЕННАЯ CLASSPATH

Если требуется добавить ссылки на дополнительные библиотеки (JAR файлы) это делается с использованием разделителя (; для Windows), например, так:

```
SET CLASSPATH=./classes;./lib/junit-4.5.jar
```

ЗАМЕЧАНИЕ: Вспомним, что символ точка означает текущую директорию .

При выполнении компиляции задается аргумент **-d**, значение которого **./classes** указывает на директорию, где должны размещаться файлы классов, полученные в результате компиляции. Обратите внимание на то, что символ разделителя директорий (/) может быть указан, как для платформы Windows, так и для платформы Unix. При компиляции множества классов из одного пакета можно использовать символ *:

```
javac -d ./classes ./src/zoostore/model/*.java
```

ЗАМЕЧАНИЕ: При выполнении название стартового класса должно быть полностью квалифицировано – с указанием цепочки иерархии пакетов.

ИТОГИ

В теме рассмотрены:

- Проблемная область приложения
- Объектно-ориентированное программирование (ООП)
- Описание класса и реализация класса
- Модификаторы доступа и ключевые слова в декларации класса
- Создание объектов (экземпляров) некоторого класса
- Порядок создания объектов
- Методы и их вызов
- Конструктор
- Переменная **CLASSPATH**