# Лекция 5. Инициализация объектов

**Класс** – это ссылочный тип данных в Java, поэтому для того чтобы создать новый объект используется оператор ***new***. При этом используется специальная конструкция вида:

|  |
| --- |
| Имя\_класса имя\_переменной = new Имя\_класса([список параметров]); |
| Формула 1. Инициализация переменной типа некоторого класса |

Инициализация полей в классе происходит именно при *создании* экземпляра некоторого класса, и может быть реализована несколькими способами:

* инициализация в месте объявления;
* инициализация в нестатическом блоке;
* инициализация в конструкторе.

# Инициализация полей класса в месте объявления

Как известно, поля класса фактически являются набором переменных некоторого типа, совокупное состояние которых называется состоянием объекта.

При создании экземпляров класса необходимо, чтобы все его поля были проинициализированы. Одним из способов инициализации является инициализация в месте объявления:

|  |
| --- |
| public class Dog {    String name = “Garry”; // имя  int age = 9; // возраст  String breed = “terrier”;// порода  } |

Надо понимать, что процесс инициализации всех полей заданными значениями происходит всякий раз, как создается очередной объект типа Dog, соответственно, любая созданная собака сразу же после инициализации будет иметь данные параметры.

Поэтому данный вид инициализации используется тогда, когда необходимо задать некоторым полям класса значения «по умолчанию». Если нет значения «по умолчания», Java-машина инициализирует поля своими значениями «по умолчанию», при этом:

|  |  |
| --- | --- |
| Тип  данных | Значение по умолчанию |
| byte, short, int, long, char | 0 |
| float, double | 0.0 |
| boolean | false |
| Массивы, классы | null |

*(!)* ***null*** *– это ключевое слово в Java (регистр имеет значение, так при использовании Null или NULL в качестве null возникнет ошибка компиляции).* ***null – это не объект специального вида,*** *но* ***null*** *может быть назначен только ссылочному типу (невозможно назначить****null****примитивной переменной). Использование* ***null****, как значения, вызов метода на переменной, значение которой* ***null****, приводит к генерации ошибки NullPointerException во время исполнения кода.*

|  |
| --- |
| **public** **class** Engine {  **public** **static** **void** main(String[] args) {  Dog businka = **null**;  System.***out***.println("Dog:" + businka);  }  } |
|  |
| Пример 1. Задание переменной ссылочного типа значения null и вывод этой переменной |
|  |
| **public** **class** Engine {  **public** **static** **void** main(String[] args) {  Dog businka = **null**;  System.***out***.println("Dog's name:"+businka.getName());  }  } |
|  |
| Пример 2. Вызов метода на объекте, значение которого null |

# Инициализация полей объекта в нестатическом блоке

Использование инициализационных блоков является альтернативой предыдущему способу инициализации.

|  |
| --- |
| **public** **class** Dog {  String name;  **int** age;  String breed;  {  System.***out***.println("+ initialization block");  breed = "UNKNOWN";  System.***out***.println("+ initialization block");  }  } |
| Пример 3. Нестатический блок инициализации |

# Конструктор класса

Фактически правая часть выражения в формуле 1 – это вызов ***конструктора*** класса, с помощью которого будет создан объект типа этого класса.

***Конструктор*** – это специальный метод класса, который вызывается при создании нового объекта. Именно конструктор отвечает за начальное состояние объекта (состояние полей класса на этапе инициализации). Наличие конструктора удобно также тем, что не всегда есть необходимость инициализировать все поля класса при создании его экземпляра (например, часть полей должна быть заполнена значениями по умолчанию).

|  |
| --- |
| [модификатор\_доступа] Имя\_конструктора([список\_параметров])  {  // инициализация полей класса  } |
| Формула 2. Объявление конструктора класса в Java |

Конструктор срабатывает непосредственно во время инициализации нового объекта. Имя конструктора совпадает в точности с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения. Конструктор в классе есть всегда (явный или неявный).

***Неявный*** конструктор есть в каждом классе, в котором не объявлен ни один конструктор. Он называется «конструктор по умолчанию», не принимает ни одного параметра, и не требует явного описания в классе (Java-машина автоматически встраивает конструктор по умолчанию в класс).

|  |
| --- |
| [модификатор\_доступа] Имя\_конструктора()  {  // инициализация полей класса  } |
| Формула 3. Неявный конструктор класса в Java |

Например, в первой реализации класса Dog, явно ни один конструктор описан не был. Однако, при создании нового объекта, в качестве конструктора был выбран единственный доступный – без параметров.

|  |
| --- |
| **public** **class** Engine {  **public** **static** **void** main(String[] args) {  Dog businka = **new** Dog(); // вызов конструктора по умолчанию, // хотя явно он был объявлен  }  } |
| Пример 4. Вызов конструктора по умолчанию |
|  |

Несмотря на то, что конструктор по умолчанию автоматически встраивается во время компиляции класса, в случае, если необходимо определить действия, производимые на этапе создания объекта, конструктор можно описать явно. Конструкторы, описанные в классе, называются ***явными***. Если в классе есть хоть один явный конструктор, то Java-машина не встраивает конструктор по умолчанию в класс.

|  |
| --- |
| **public** **class** Dog {  String name;  **int** age;  String breed;    **public** Dog() { //явный конструктор без параметров  System.***out***.println("+ Dog()");  breed = "UNKNOWN";// каждому новому объекту задается значение породы  System.***out***.println("- Dog()");  }  } |
|  |
| Пример 5. Явный конструктор без параметров в классе Dog |

В классе может быть объявлено несколько конструкторов. Единственное правило, чтобы у них не был одинаковый список параметров. В таком случае возникает ситуация неоднозначности – для Java-машины не понятно, какой конструктор использовать для создания объекта. Собственно, именно по списку параметров происходит выбор конструктора класса для инициализации объекта.

|  |
| --- |
| **public** **class** Dog {  String name;  **int** age;  String breed;  **public** Dog(String name) {  System.***out***.println("+ Dog(name)");  **this**.name = name;  System.***out***.println("- Dog(name)");  }    **public** Dog(**int** aAge) {  System.***out***.println("+ Dog(aAge)");  **this**.age = aAge;  System.***out***.println("- Dog(aAge)");  }  } |
| Пример 6. Несколько конструкторов в классе Dog |

Ситуация, когда имена конструкторов совпадают, но различаются по списку параметров, называется ***перегрузкой*** конструкторов.

В примере 6 в классе Dog объявлено 2 конструктора. Первый в качестве параметра получает имя (переменная типа String) и присваивает полю name переданное значение. Второй – переменную типа int (возраст) и инициализирует этим значением поле age. Эти конструкторы различаются, так как на вход первого поступает аргумент типа String, на вход второго – int.

*(!)* ***Стоить заметить, что при попытке в примере 5 вызвать конструктор без параметров, компилятор сообщит об ошибке: в списке явных конструкторов нет конструктора без параметров****.*

Конструкторы могут вызывается друг из друга. В этом случае, каждый из них выполняет свои действия по инициализации. Есть единственное правило: вызов другого конструктора должен быть первым действием в вызываемом конструкторе.

|  |
| --- |
| **public** **class** Dog {  String name;  **int** age;  String breed;  **public** Dog(String name) {  System.***out***.println("+ Dog(name)");  **this**.name = name;  System.***out***.println("- Dog(name)");  }    **public** Dog(**int** aAge) {  **this**("BUSINKA"); // вызов конструктора, который на вход получает  // параметр типа String  System.***out***.println("+ Dog(aAge)");  age = aAge;  System.***out***.println("- Dog(aAge)");  }  } |
| **public** **class** Engine {  **public** **static** **void** main(String[] args) {  Dog businka = **new** Dog(**5**);  }  } |
|  |
| Пример 7. Вызов конструктора из конструктора |

# Порядок инициализации объекта

1. Инициализация полей в месте объявления и в инициализационном блоке происходит до инициализации в конструкторе.
2. Инициализации полей в месте объявления и в инициализационных блоках выполняются в порядке их объявления в классе.
3. Инициализация полей базового класса происходит полностью до инициализации производного класса, т.е. сначала выполняются все инициализаторы базового класса, а потом все инициализаторы производного класса.

(Пункт 3 будет отдельно рассмотрен далее)

**Самостоятельно:**

Применить все 3 способа инициализации полей в классе Dog и убедиться в их очередности.