Статическая инициализация

Поля и методы в Java могут принадлежать классу или объекту. Поля, принадлежащие классу, являются статическими, а поля, принадлежащие объекту, - нестатическими.

Статические поля доступны без создания объекта класса и должны инициализироваться до создания объекта класса. Нестатические поля должны инициализироваться после создания объекта класса.

Независимо от вида поля существуют такие способы инициализации:

|  |  |  |
| --- | --- | --- |
| **Название** | **Применимость** | **Описание** |
| Инициализация в месте объявления поля | Статические поля, поля объекта | Применяется, если инициализация может быть произведена коротким выражением и доступен контекст, необходимый для ее проведения |
| Инициализационный блок | Статические поля, поля объекта | Применяется, если инициализационный код неудобно записывать одним выражением или же, например, нужна обработка проверяемых исключений. В случае объектов может применяться для инициализации полей объектов анонимных классов. |
| Конструктор класса | Поля объекта | Применяется, если для инициализации нужны параметры конструктора |

Инициализация статических полей в месте объявления или в статическом инициализационном блоке называется статической инициализацией.

Нужно учитывать, что в статическом инициализационном блоке доступны только к статические поля и методы.

Пример статической инициализации переменной:

class Integer {

...

public static final int SIZE = 32;

...

}

Пример статического инициализационного блока:

static List<Character> alphabet;

static {

alphabet = new ArrayList<Character>();

for (char c='a'; c<='z'; c++) alphabet.add(c);

}

**Порядок инициализации**

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

**Применение статической инициализации**

* Для инициализации статических переменных. Статическая инициализация выполняется один раз при загрузке класса, в том порядке, как переменные описаны в коде. Это лучше всего подходит для переменных принадлежащих классу и существующих столько же сколько сам класс (не объект). Не тратятся ресурсы на повторную инициализацию;
* Сделать инициализацию до выполнения конструктора;
* В момент загрузки Java обеспечивает потокобезопасность для статических переменных. Этот факт тоже можно использовать;
* В статическом блоке можно создать объект безымянного внутреннего класса;

Перечисления

Часто в программировании возникает задача создать тип данных множество значений которого ограничено. Например, день недели может иметь 7 разных значений, месяц в году – 12 и т. п.

В Java, начиная с версии 1.5 предусмотрен специальный тип данных - перечисление (enum). Перечисление в Java – это не просто набор констант (как, например, в С++).

Перечисления в Java определяют тип класса. Каждый элемент перечисления – это экземпляр enum-класса, доступный статически.

Пример объявления перечисления:

public enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

THURSDAY, FRIDAY, SATURDAY

}

В этом примере SUNDAY – это объект типа Day.

Хотя перечисления представляют тип класса, нельзя создать объект такого класса с помощью new. Перечисления создаются также как переменные примитивных типов.

**Особенности перечислений**

* Невозможно назначить переменной перечисления другое значение, кроме перечисленных в enum констант;
* Все перечисления неявно наследуются от класса java.lang.Enum а значит содержат методы родительского класса name(), ordinal(),valueOf(), и т.д;
* В классе перечисления имеется возможность задавать конструкторы (только приватные), поля и методы;
* Перечисления могут реализовывать любые интерфейсы;
* Методы в перечислении могут быть абстрактными, а конкретные экземпляры констант могут определять такие методы (как, впрочем, и переопределять уже определенные);
* Элементы перечисления неявно являются static и final и могут быть инициализированы только раз;

**Предопределенные методы перечислений**

|  |  |
| --- | --- |
| **public final** String name() | Возвращает имя данного элемента перечисления, как оно задано при объявлении перечисления. |
| **public final int** ordinal() | Возвращает порядок данного элемента перечисления, как он задано при объявлении перечисления. Нумерация начинается с нуля |
| **public static** EnumClass valueOf(String name) | Возвращает элемент перечисления по имени, как оно задано при объявлении перечисления |
| **public static** EnumClass[] values() | Получение всех элементов перечисления |

**Наследование в enum**

Наследовать тип класса перечисления нельзя, поскольку множественное наследование в Java запрещено.

Но каждый элемент перечисления может представлять собой класс производный от класса перечисления. Каждый класс элемента будет доступен статически, в одном экземпляре. При этом элементы enum могут содержать собственные конструкторы.

Пример:

**enum** Type {  
 ***INT***(**true**) {  
 **public** Object parse(String string) {**return** Integer.*valueOf*(string);}  
 },  
 ***INTEGER***(**false**) {  
 **public** Object parse(String string) {**return** Integer.*valueOf*(string);}  
 },  
 ***STRING***(**false**) {  
 **public** Object parse(String string) {**return** string;}  
 };  
  
 **boolean primitive**;  
 Type(**boolean** primitive) {**this**.**primitive** = primitive;}  
  
 **public boolean** isPrimitive() {**return primitive**;}  
 **public abstract** Object parse(String string);  
}

Здесь объявляется перечисление Type с тремя элементами INT, INTEGER и STRING. Компилятор создаст следующие классы и объекты:

* Type - класс производный от java.lang.Enum;
* INT - объект 1-го класса производного от Type;
* INTEGER - объект 2-го класса производного от Type;
* STRING - объект 3-го класса производного от Type;

1. <http://www.quizful.net/post/java-fields-initialization>
2. <http://info.javarush.ru/translation/2014/04/15/10-%D0%B7%D0%B0%D0%BC%D0%B5%D1%82%D0%BE%D0%BA-%D0%BE-%D0%BC%D0%BE%D0%B4%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82%D0%BE%D1%80%D0%B5-Static-%D0%B2-Java.html>
3. <http://www.quizful.net/post/java_enums>
4. <https://habrahabr.ru/post/101280/>
5. <http://info.javarush.ru/translation/2015/10/13/15-%D0%B2%D0%BE%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2-%D0%B4%D0%BB%D1%8F-%D1%81%D0%BE%D0%B1%D0%B5%D1%81%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA%D0%BE%D0%B2-%D0%BE%D1%82%D0%BD%D0%BE%D1%81%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE-Enum-%D0%B2-%D0%94%D0%B6%D0%B0%D0%B2%D0%B5-%D1%81-%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0%D0%BC%D0%B8-.html>