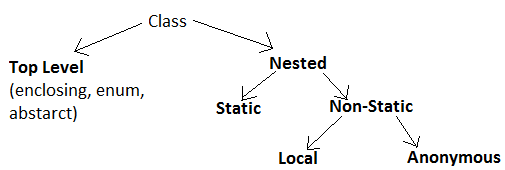
**LESSON 2**

**Виды классов**

Представим дерево классов в таком виде:



Или разобьём их на 5 групп:

1. Top Level
2. Static Nested
3. Non-static nested
4. Local
5. Anonymous

|  |  |  |
| --- | --- | --- |
|  | **Top Level (class)** | **Non-static (внутренний)** |
| Определение | Класс, который объявляется сразу в пакете. | Класс, определение которого содержится внутри другого класса. |
| Ключевые слова | 1. public, friendly 2. final, abstract, strictfp | 1. Все модификаторы доступа 2. **Не** может быть native, transient, synchronized, volatile |
| Область видимости | Public – доступен в любом пакете  Friendly – доступен внутри пакета | Public – доступен в любом пакете  Friendly – доступен внутри пакета  Protected – доступен внутри пакета и в наследниках  Private – доступен только внутри класса |
| К чему имеют доступ | 1. Получить доступ к элементам **внутреннего** класса можно только создав объект. А доступ к переменной final static можно через имя класса. 2. Вызвать статический метод **вложенного** можно через указание полного пути | Имеют прямой доступ к полям и методам внешнего класса через ссылку this |
| Можно ли создать объект напрямую | A a = new A() | Создание объекта:  Outer.Inner obj = new Outer().new Inner(); |
| От кого могут наследоваться, кого реализовывать | Без ограничений | Могут быть производными, базовыми в пределах внешнего, реализовывать интерфейсы |
| this  super | This – ссылка на самого себя  Super – ссылка на базовый класс | This – ссылка на самого себя  Super – ссылка на базовый класс  Имя\_внешнего\_класса.this – ссылка на внешний класс |
| Во что компилируются | В class . Сколько классов объявлено, столько и скомпилируется. | В байткоде внешнего класса будет отмечен как inner class. В собственном байткоде (причём файл имеет следующее название Car$MyClass.class) будет указана ссылка на sourcefile на внешний класс, и везде используется обозначение **Car$MyClass** |
| Когда используются | Всегда | Когда является неотъемлемой составляющей (подводная лодка –> двигатель для подводной лодки). Для хорошего singleTone |
| Примеры кода |  |  |
| Замечания |  | 1. Не содержат static полей, кроме final static |

|  |  |  |
| --- | --- | --- |
|  | **Static (вложенный)** | **Local (локальный)** |
| Определение | Внутренний static класс. | Внутренний класс, объявленный в методе или логическом блоке. |
| Ключевые слова | 1. Обязательно static 2. Все модификаторы доступа 3. Не может быть native, transient, synchronized, volatile | 1. Не имеет модификаторов доступа 2. Может быть strictfp, final, adbstract |
| Область видимости | Public – доступен в любом пакете  Friendly – доступен внутри пакета  Protected – доступен внутри пакета и в наследниках  Private – доступен только внутри класса | В данном логическом блоке или методе |
| К чему имеют доступ | Имеет прямой доступ к static полям и методам внешнего класса.  Для доступа к **не** static полям и методам внешнего должен создавать объект внешнего. | Имеют прямой доступ к полям и методам внешнего класса через ссылку this + к локальным переменным. |
| Можно ли создать объект напрямую | Outer.Inner obj = new Outer.Inner() | Нет |
| От кого могут наследоваться, кого реализовывать | Может быть базовым, производным и реализовывать интерфейс. | Может быть базовым, производным и реализовывать интерфейс в пределах «{ …. }» |
| this  super | Не имеет указателя this | This – ссылка на самого себя  Super – ссылка на базовый класс  Имя\_внешнего\_класса.this – ссылка на внешний класс |
| Во что компилируются | В байткоде внешнего класса будет отмечен как inner class. В собственном байткоде (причём файл имеет следующее название Car$MyClass.class) будет указана ссылка на sourcefile на внешний класс, и везде используется обозначение Car$MyClass | Пусть в классе А есть 2 локальных метода с названием B (конфликта не будет, так как они объявлены в разных местах) и локальный С. При компиляции создастся 3 файла class с именем A$1B.class , A$2B.class и A$1C.class |
| Когда используются | Предоставляет услугу внешнему классу. Для singleTone | История умалчивает |
| Примеры кода |  |  |
| Замечания | 1. Подкласс вложенного класса не наследует возможность доступа к членам внешнего класса, которым наделен суперкласс. 2. Класс вложенный в интерфейс static по умолчанию. 3. Можно объявить в классе, интерфейсе, абстрактном классе 4. Чтобы обратиться к статическому классу, вложенный в интерфейс, нужно указать Имя\_интерфейса.Имя\_класса 5. В static классе можно объявить всё, что угодно. | 1. Не содержат static полей, кроме final static |

|  |  |  |
| --- | --- | --- |
|  | **Анонимный (Anonymous)** | **Перечисления (Enum)** |
| Определение | Расширяет другой класс или реализует интерфейс при объявлении одного единственного объекта. | Классы специального вида, которые: |
| Ключевые слова | Нет | Public – доступен в любом пакете  Friendly – доступен внутри пакета |
| Область видимости | Нет |  |
| К чему имеют доступ | **Не** имеет доступ к private |  |
| Можно ли создать объект напрямую | Нет. Объявление анонимного класса выполняется одновременно вместе с созданием объекта с помощью оператора new. | Нельзя создать через new. Объект перечисления создаётся один раз при загрузке перечисления в память. |
| От кого могут наследоваться, кого реализовывать | Нет | Не могут иметь наследников.  Наследуются от java.lang.Enum и реализуют Comparable и Serializable. |
| this  super | This – ссылка на себя  Super – ссылка на класс, который переопределяем |  |
| Во что компилируются | В отдельный файл с названием Main$1.class |  |
| Когда используются | Когда необходимо для одного объекта переопределить метод.  Для Runnable, Comparator, лямбда-выражения | Предоставляют типизированный, безопасный способ задания фиксированных наборов значений. |
| Примеры кода | **Runnable**    **Comparator** | **Анонимные перечисления** ( перечисления, в которых отдельные элементы могут реализовывать свое собственное поведение.) |
| Замечания | 1. Конструктор ни определить, ни переопределить нельзя. 2. Допускается вложенность друг в друга. 3. По ссылке ничего не сможем получить, чего нет в базовом 4. Если анонимный класс определяется не на классе, а на интерфейсе, то нужно переопределить все методы | 1. Элементы перечисления по умолчанию public static final 2. Если содержит абстрактный метод, то должен переопределить 3. Могут содержать вложенные классы 4. Конструктор только private 5. Enum может содержать non-static, static classes, interface, static{}, {}, enum 6. Пример enum в enum календарь: 12месяцев, 7 дней недели |

**Методы Enum**

|  |  |
| --- | --- |
| clone() | Throws CloneNotSupportedException. This guarantees that enums are never cloned, which is necessary to preserve their "singleton" status. |
| compareTo() | Compares this enum with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object. Enum constants are only comparable to other enum constants of the same enum type. The natural order implemented by this method is the order in which the constants are declared. |
| equals() | Returns true if the specified object is equal to this enum constant. |
| finalize() | enum classes cannot have finalize methods |
| getDeclaringClass() | Returns the Class object corresponding to this enum constant's enum type. Two enum constants e1 and e2 are of the same enum type if and only if e1.getDeclaringClass() == e2.getDeclaringClass(). (The value returned by this method may differ from the one returned by the Object.getClass() method for enum constants with constant-specific class bodies.) |
| hashCode() | Returns a hash code for this enum constant. |
| name() | Returns the name of this enum constant, exactly as declared in its enum declaration. Most programmers should use the toString() method in preference to this one, as the toString method may return a more user-friendly name. This method is designed primarily for use in specialized situations where correctness depends on getting the exact name, which will not vary from release to release. |
| ordinal() | Returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero). Most programmers will have no use for this method. It is designed for use by sophisticated enum-based data structures, such as EnumSet and EnumMap. |
| toString() | Returns the name of this enum constant, as contained in the declaration. This method may be overridden, though it typically isn't necessary or desirable. An enum type should override this method when a more "programmer-friendly" string form exists. |
| valueOf(Class<T> enumType, String name) | Returns the enum constant of the specified enum type with the specified name. |
| values() | Returns all the enum constant. |

**Enum** vs **Constant**

|  |  |
| --- | --- |
| **Enum** | **Constant** |
| При использовании оператора switch. Если вдруг нам на вход придёт неправильный enum, то мы получим ошибку. | При использовании оператора switch. Если вдруг нам на вход придёт неправильная константа, то мы попадём в блок default |
| Инициализирует все объекты перечисления сразу, при загрузке класса в память. А если у нас большое количество перечислений и на данный момент мы пользуемся только небольшой часть, то для нас это потеря памяти и сокрости. |  |

|  |  |
| --- | --- |
|  | **Параметризованный (Generic)** |
| Определение | С помощью шаблонов можно создать параметризованные классы и методы, что позволяет использовать более строгую типизацию |
| Ключевые слова |  |
| Область видимости | Public – доступен в любом пакете  Friendly – доступен внутри пакета |
| К чему имеют доступ |  |
| Можно ли создать объект напрямую |  |
| От кого могут наследоваться, кого реализовывать |  |
| this  super |  |
| Во что компилируются | .class |
| Когда используются |  |
| Примеры кода |  |
| Замечания | 1. Если не указать параметр, то компилятор подставит Object 2. Чтобы расширить возможности параметризованных членов используется extends |

*Замечания:*

1. глубина вложенности ограничивается длиной имени файла
2. в абстрактном классе не может быть единственного private конструктора



**Абстрактный класс**

Абстрактный класс – это класс, объявленный с модификатором abstract.

Экземпляры такого класса создать нельзя, но можно создать абстрактные ссылки, которые будут ссылаться на класс, наследующий данный абстрактный. Причём класс, который наследует абстрактный класс, должен реализовывать все абстрактные методы, иначе будет объявлен тоже абстрактным.

Abstract Method -> Abstract Class

Abstract Class -> Abstract Method //нет

*Зачем нужен?*

В результате рефакторинга кода может возникнуть ситуация, когда выделяются классы, которые начинают играть абстрактную роль и реализовывать в них некоторые методы не имеет смысла. Но и удалить такие методы нельзя, так как при использовании базовых объектных ссылок на объекты производных классов теряется возможность доступа к таким методам. В таком случае их называют abstract.

**Интерфейс**

Интерфейс – полностью абстрактный класс.

*English*: *There are a number of situations in software engineering when it is important for disparate groups of programmers to agree to a "contract" that spells out how their software interacts. Each group should be able to write their code without any knowledge of how the other group's code is written. Generally speaking, interfaces are such contracts.*

Интерфейсы в джава используются для добавления новых возможностей классам, которых нет и не может быть в базовом.

Интерфейсы говорят о том, что класс должен делать, но не говорят как. Таким образом интерфейс – это есть некий контракт, который обязаны соблюдать «реализаторы».

*Объявление интерфейса*

**

Модификаторы: public, friendly

Поля по умолчанию **public final static,** методы – **public**, классы – **static**.

В интерфейс нельзя вложить логический или статический блок.

В интерфейс можно вложить класс, но он будет **public static** по умолчанию. Можно вложить интерфейс, который будет тоже **public** по умолчанию, перечисления.

Нельзя создать экземпляр интерфейса через оператор new. Но можно создать интерфейсные ссылки, причём они должны ссылаться на классы, которые реализуют этот интерфейс. Через интерфейсную ссылку можно вызвать только те методы, что объявлены в интерфейсе.

*Какие типы интерфейсов бывают?*

Маркерные - интерфейсы, которые не содержат методов для реализации, а специальным образом обрабатываются JVM. Это интерфейсы:

|  |  |
| --- | --- |
| **java.lang.Cloneable** | A class implements the Cloneable interface to indicate to the [Object.clone()](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html" \l "clone()) method that it is legal for that method to make a field-for-field copy of instances of that class. |
| **java.io.Serializable** | Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable. |
| **java.rmi.Remote** | The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a "remote interface", an interface that extends java.rmi.Remote are available remotely. |

Функциональные (Java 8) – это интерфейсы с одним единственным методом. Они могут быть представлены при помощи лямбда-выражений, ссылками на методы и конструкторами ссылок. Примеры:

|  |  |
| --- | --- |
| **java.lang.Runnable** | The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run. |
| **java.util.Comparator** | A comparison function, which imposes a total ordering on some collection of objects. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps), or to provide an ordering for collections of objects that don't have a natural ordering. |
| **java.util.concurrent.Callable** | A task that returns a result and may throw an exception. Implementors define a single method with no arguments called call.  The Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. A Runnable, however, does not return a result and cannot throw a checked exception. |

**Interface** vs **Abstract** vs **Class**

|  |  |  |
| --- | --- | --- |
| **Interface** | **Abstract** | **Class** |
| Задаёт поведение |  |  |
| Возможно реализовать «множественное наследование» | Нет | Нет |

*Что, если класс реализует два интерфейса, в которых есть абсолютно идентичные методы (сигнатура и возвращаемый тип совпадают)?*

В таком случае конфликта не произойдёт. Иначе:



**Типы данных**

**Базовые типы** данных разделяют на 4 группы:

|  |  |  |  |
| --- | --- | --- | --- |
| **Группы** | **Название** | **Размерность (бит)** | **Значение по умолчанию** |
| Целые | byte | 8 | (byte)0 |
| short | 16 | (short)0 |
| int | 32 | 0 |
| long | 64 | 0L |
| С плавающей точкой | float | 32 | 0.0f |
| double | 64 | 0.0d |
| Символы | char | 16 | ‘\u0000’ null |
| Логические | boolean | - | false |

Они хранятся в стеке.

Размер примитивных типов одинаков для всех платформ, за счёт этого становится возможным переносимость кода.

В метод как параметр передаются по значению.

Java запрещает смешивать в выражении величины разных типов, поэтому есть преобразование типов (явное и неявное). Заметим, что нет преобразования между Boolean и другими типами.

**Неявное** (повышающее, разрешённое) преобразование - преобразование, при котором результирующий тип имеет больший диапазон значений, чем исходный. Например, int x =2000; long y =x;

Также такое образование осуществляется автоматически, даже в случае потери данных. Потеря данных может произойти, если: int -> float , long -> float , float -> double.

*Почему происходит потеря данных?*

*Как избежать потери данных?*

Использовать BigDecimal, BigInteger

**Явное** (понижающее) преобразование – преобразование, при котором результирующий тип имеет меньший диапазон, чем исходный. Пример, long x =2000; int y = (int) x;

Null нельзя привести к нулю.

Слишком большое дробное число при приведении к целому превращается в Integer.MAX\_VALUE или Integer.MIN\_VALUE.

Слишком большой double при приведении к float превращается в Float.POSITIVE\_INFINITY или Float.NEGATIVE\_INFINITY.

 => 

**Литералы** – это константы, которые записаны по правилам языка Java

|  |  |  |
| --- | --- | --- |
| Целочисленные | Десятичные | 1, 24 (int),  1L , 42L (long) |
| Восьмеричные | 01, 017 (int),  01L (long) |
| Шестнадцатеричные | 0x12, 0xfffff (int),  0x12L (long) |
| Литералы с плавающей точкой | Стандартная форма | 2.0 , 3.1415 (double),  2.0f , 3.1415f (float) |
| Научная форма | 6.022e23 (double),  6.022e23f (float) |
| Булевские константы |  | true |
|  | false |
| Символьные литералы | Непосредственный вид | ‘a’  ‘\ddd’ восьмеричный символ ddd  ‘\uxxxx’ шестнадцатеричный символ хххх |
| Ввод с помощью управляющей последовательностью | ‘\’’ одинарная кавычка  ‘\n’ перевод на новую строку |

**Классы-оболочки**

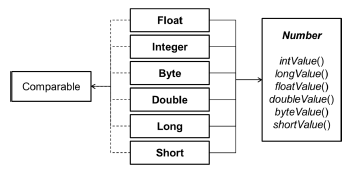
Byte, Short, Integer, Long, Float, Double, Character, Boolean

Они хранят те же значения, что и соответствующие им базовые.

Объекты данных классов являются ссылками на участки динамической памяти.

Переменная базового типа передаётся в метод по ссылке.

Классы оболочки являются immutable. Это означает, что их состояние изменить нельзя.



*Зачем реализуют Comparable?*

Для того, чтобы сравнивать объекты не по ссылкам (как это происходит в стандартном определении), а по значению.

*Почему классы-оболочки Immutable?*

Их можно использовать в коллекциях в качестве ключа. Так как в зависимости от хэшкода ключа создаётся бакет. Если бы можно было изменить значение ключа, следовательно, изменился бы хэшкод.

Также, имеем преимущества при синхронизации данных. Ведь, если в новом потоке изменить значение такой переменной (то есть мы создадим новый объект), то в другом потоке она останется прежней.

И ещё, для классов-оболочек есть пул констант, то есть значения хэшируются.

*В чём отличие в создании строки как new() или литералом (при помощи двойных кавычек)?*

Если создавать через new() , то создаться строка в куче(динамической памяти) и в пуле строк. А строка, созданная при помощи литерала, хранится только в пуле строк.

*Что такое ссылка?*

Ссылка -  множество величин, объединенных определенной совокупностью допустимых операций.

Integer ref; // ссылка с именем ref , а типом Integer, неинициализированная

|  |  |  |  |
| --- | --- | --- | --- |
| **Strong Reference** | **Weak Reference** | **Soft Reference** | **Phantom Reference** |
| Если объект доступен через цепочку ссылок (как например объект, на который ссылается переменная buffer), то такая ссылка называется жесткой и сборщик мусора не станет уничтожать такой объект.  Минусы:  Утечка памяти (должны перенаправить на null, чтобы сборщик мусора её обработал)  Кэширование (сборщик мусора не сможет освободить память, которую занимает закэшированная ссылка) | Слабая ссылка - это ссылка, которая недостаточно сильна чтобы объект не собирался сборщиком мусора. | По своей природе данный вид ссылок очень похож на WeakReference, с одним очень существенным отличием: объекты по ссылкам уничтожаются в том случае, когда память вашей программы заполнена и появляется вероятность получить OutOfMemoryError  Рекомендуется для кэширования | Связь с объектами в этих ссылках такая слабая, что вы даже не сможете получить эти объекты - get() метод всегда будет возвращать null.  Область применения этих ссылок в отслеживании момента, когда ссылка помещается в очередь ReferenceQueue |
| StringBuffer buffer = new StringBuffer(); | WeakReference<Thing> weakThing = new WeakReference<Thing>(thing); | SoftReference<Thing> thing = new SoftReference<Thing>(new Thing()); | PhantomReference<Thing> thing = new PhantomReference<Thing>(new Thing(), queue); |

*Что можно отправить в оператор switch?*

char, byte, short, int, Character, Byte, Short, Integer, String, Enum, Const

**Primitive** vs **Reference**

|  |  |  |
| --- | --- | --- |
| Критерии | **Базовые типы** | **Ссылочные типы** |
| Место в памяти | Стек | Куча (динамическая память) |
| Как передаётся в метод | По значению | По ссылке |
|  | Обеспечивают более высокую производительность вычислений | Используются в коллекции |
|  |  | Нет проблем с синхронизацией |
|  |  | Кэшируются (пул литералов) |

**Автоупаковка** – процесс автоматической инкапсуляции данных басовых типов в соответствующие объекты-оболочки.

**Автораспаковка** – процесс извлечения из объекта-оболочки значения базового типа.

**Массив**

Массивом называется именованное множество переменных одного типа. Массив используется для хранения нескольких однотипных значений.

*English:* An *array* is a container object that holds a fixed number of values of a single type.

Индексация начинается с нуля. Размер массива не меняется (даже если с середины массива удалить элемент).

Чтобы узнать длину массива: имя\_массива.length.

Все массивы хранятся в куче.

Массив объектов представляет собой массив ссылок, проинициализированных по умолчанию null.

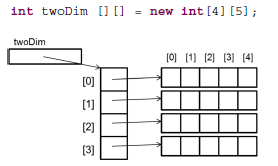
Способы объявления (int [] a === int a []):

1. int [] a = new int[10];
2. int [] a = new int [] {1,3,2};
3. int [] a = {1,3,2}; *//этим мы только себе упрощаем жизнь, а компилятор всё равно подставит new*
4. int[] a = (int[]) Array.newInstance(int.class, 10);

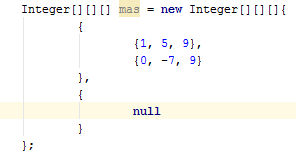
Многомерных массивов не существует, но можно объявить массив массивов.

Способы объявления:

1. Int[][] a = new int[2][5];
2. Int[][] a = new int[2][]; a[0] = new int[2]; a[1] = new int[5];
3. Int[][] a = {{1}, {2,3}, {4,5,6}, null};



Массив массивов массивов



*Какие ошибки могут возникнуть при работе с массивами?*

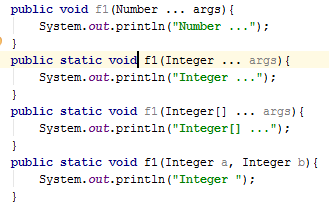
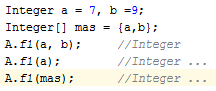
При обращении к несуществующему индексу - java.lang.ArrayIndexOutOfBoundsException.



При попытке поместить в массив неподходящий элемент - java.lang.ArrayStoreException.

|  |  |
| --- | --- |
| **Array** | **Arrays** |
| The Array class provides static methods to dynamically create and access Java arrays. | This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists. |
| java.lang.reflect.Array | java.util.Arrays |
| 1. newInstance(Class<?> componentType, int length) - creates a new array with the specified component type and length. | 1. asList(T... a) - returns a fixed-size list backed by the specified array. 2. binarySearch(byte[] a, byte key) - searches the specified array of bytes for the specified value using the binary search algorithm. 3. sort(T[] a, Comparator<? super T> c) - sorts the specified array of objects according to the order induced by the specified comparator. |

**Varargs**

** **

Замечания:

1. Нельзя Integer … перегрузить Integer []
2. Аргумент с переменным числом параметров задаётся последним в списке аргументов

**Final переменные**

When an anonymous inner class is defined within the body of a method, all variables declared final in the scope of that method are accessible from within the inner class. For scalar values, once it has been assigned, the value of the final variable cannot change. For object values, the reference cannot change. This allows the Java compiler to "capture" the value of the variable at run-time and store a copy as a field in the inner class. Once the outer method has terminated and its stack frame has been removed, the original variable is gone but the inner class's private copy persists in the class's own memory.