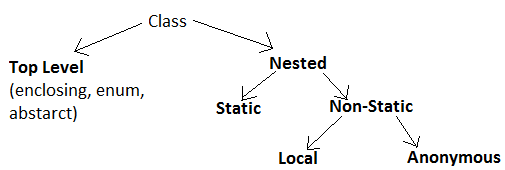
**LESSON 2**

**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 | 1. Не содержат static полей, кроме final static |

|  |  |  |
| --- | --- | --- |
|  | **Анонимный** | **Перечисления** |
| 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 |

|  |  |
| --- | --- |
|  | **Параметризованный** |
| 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



1. В статическом классе может быть всё, что угодно
2. Чтобы обратиться к статическому классу, вложенный в интерфейс, нужно указать Имя\_интерфейса.Имя\_класса
3. Enum может содержать non-static, static classes, interface, static{}, {}

**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 |

**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?

**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

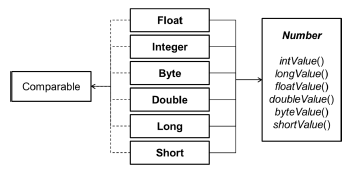
Они хранят те же значения, что и соответствующие им базовые.

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.

Значения



*Зачем реализуют Comparable?*

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

*Почему они Immutable?*

*Что такое ссылка?*

Значения ссылок (чаще просто ссылки) – это указатели на эти объекты, и специальная ссылка null, которая означает отсутствие объекта.

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

**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 |  |

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

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