JRE, JDK, JVM, Console

**Java Runtime Environment**, сокращенно JRE – это исполнительная среда Java в которой выполняются программы, написанные на этом языке. Среда состоит из виртуальной машины – Java Virtual Machine(JVM) и библиотеки Java классов. JRE является частью JDK. По сути это минимальная реализация виртуальной машины, необходимая для исполнения Java приложений, без компилятора и других средств разработки. Хотя JRE входит в состав JDK, фирма SUN распространяет этот набор и отдельным файлом. Это вызвано тем, что установка JRE является необходимым и достаточным условием для выполнения Java программ.

**Java Virtual Machine**, сокращенно JVM – это виртуальная машина Java — основная часть исполняющей среды JRE. Виртуальная машина Java интерпретирует и исполняет байт-код Java. Байт код получают посредством компиляции исходного кода программы с помощью компилятора Java (стандартный - javac).  JVM называется «виртуальной», потому что она предоставляет машинный интерфейс, не зависящий от операционной системы и хардвэйр архитектуры.

**Java Development Kit**, сокращенно JDK – это комплект разработчика приложений на языке Java, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java (JRE). В состав JDK не входит интегрированная среда разработки на Java (IDE), поэтому разработчик, использующий только JDK, вынужден использовать внешний текстовый редактор и компилировать свои программы, используя утилиты командной строки.

**Console app run**

javac (имя класса).java

Эта команда запускает компилятор javac.exe.

После этого компилятор создает файл .class с соответствующим именем. Запустить его можно с помощью команды java (имя класса).

javac -d bin src/(имя класса).java - сохранить скомпилированные классы в отдельную папку

Класс

Класс может быть определен как шаблон, описывающий поведение и/или состояние объекта типа этого класса.

Используется для описания общих характеристик и паттернов поведения некоторой совокупности объектов. Все экземпляры одного класса (объекты, порожденные от одного класса) имеют один и тот же набор свойств и общее поведение, то есть одинаково реагируют на одинаковые сообщения.

Для объявления класса необходимо ключевое слово class. Тело класса заключается в фигурные скобки. Оно содержит код – конструкторы (для инициализации новых объектов), поля (описывают состояние класса/объекта), методы (определяют поведение объектов класса).

В целом, объявление класса может содержать следующие компоненты (в следующем порядке):

Модификаторы доступа (public/дефолт для внешних классов, (любые для вложенных классов)) и другие модификаторы класса.

1.       Имя класса (в соответствии с конвенциями, должно начинаться с заглавной буквы): может состоять из любых java letters (символы юникода + $ + \_ + цифры). $ не рекомендуется использовать, т.к используется при комплияции классов в названиях файлов. С цифры нельзя начинать имя класса. Макс. длинна - 65535 символов (ограничение по байтам, не по символам)

2.       Имя суперкласса, если есть, следующее за ключевым словом extends. Класс может наследоваться только от одного суперкласса.

3.       Перечисление (через запятую) интерфейсов, которые реализует данный класс. Класс может реализовывать несколько интерфейсов.

4.       Тело класса, заключенное в фигурные скобки {}.   
   
**Файл .class:**   
   
. class файл генерируется джава-компилятором. .class файл содержит байт-код. Байт код – это платформно-независимая инструкция для JVM и не является машинным кодом.    
   
. class файл вложенных классов содержит символ $ в имени. В случае, если джава-файл содержит два класса, один из которых вложенный, то компилятор создаст два файла с расширением class.   
   
.class файл идентифицируется по «магическому числу» в заголовке, которое состоит из четырех байт 4 CA FE BA BE ( in hex). Это «магическое число» - первый элемент файла с расширением class, за которым следует версия файла class.   
    
Джава файл с расширением класс состоит из 10 базовых компонентов:   
   
Magic Number : 0xCAFEBABE   
Major and Minor версии .class файла: Эти поля указывают на номер версии компилятора языка Ява, которым был создан этот файл класса.   
Constant pool - constants for this class: Константный пул - это таблица значений. Эти значения являются различными строковыми константами, именами классов, именами полей и другими, которые обозначены структурой класса или кодом.   
Access modifiers - to indicate whether class is static or abstract: Это поле включает трафарет до 16 модификаций, используемых в описаниях классов, методов и полей.    
This class - Это поле-индекс в константном пуле; constant\_pool[this\_class] должен быть CONSTANT\_class.   
Super class - имя суперкласса. Это поле-индекс в константном пуле. Если значение super\_class не равно нолю, то constant\_pool[super\_class] должен быть классом, и представляет индекс суперкласса класса в константном пуле. Если значение super\_class равно нолю, то класс должен быть java.lang.Object, и он не имеет суперкласса.   
Interfaces   
Fields   
Methods   
Attributes   
    
Имя.class файла – это имя внешнего (невложенного) класса в джава-файле (Hello.java создаст Hello.class)

Конструктор, переопределение и перегрузка

**Конструктор**

Конструктор — это частный случай метода, который используется для инициализации состояния объекта. Имя конструктора всегда совпадает с именем класса, в котором он расположен. У конструкторов нет типа возвращаемого результата - никакого, даже void.

Если в классе нет никакого конструктора, джава создает конструктор без параметров. Если есть конструктор с параметрами, конструктор без параметров автоматически не создается.

Как и методы, конструкторы могут иметь любые модификаторы доступа: public, protected, private, или модификатор может отсутствовать (часто имеется ввидуpackage или friendly). Но в отличии от методов, конструкторы могут иметь модификаторы только доступа. Поэтому, конструктор не может бытьabstract, final, native, static, or synchronized.

В каждый конструктор компилятор автоматически вставляет вызов конструктора суперкласса без параметров. Конструкторы можно напрямую вызывать только из других конструкторов при помощи this(); и super();

Super() всегда идет первой строчкой, явно или нет.

Конструкторы могут бросать exception.

**Метод**- набор действий, объявленный под одним именем и характеризующий поведение.

**Переопределение и перегрузка**

Методы могут быть перегружены и переопределены. Конструкторы — только перегружены.

Нельзя переопределять static, private, final методы.

**При переопределении методы**

         Должны иметь неизменный список агрументов.

         Должны иметь неизменный тип возвращаемого значения.

         Уровень доступа не может быть ниже наследованного метода.

         Не должны бросать новых exception'ов.

         Переопределяться могут только наследованные методы.

**При перегрузке методы**

         Должны иметь новый список аргументов.

         Могут иметь новый тип возвращаемого значения.

         Могут иметь новый уровень доступа без ограничений.

         Могут бросать новые exception'ы

         Методы суперкласса могут быть перегружены в наследнике.

         Полиморфизм применим только к переопределению — не к перезагрузке.

Вложенные и внутренние классы

Вложенные и внутренние классы

Определение класса может размещаться внутри определения другого класса. Такие классы называются вложенными или внутренними. Область видимости вложенного класса ограничена областью видимости внешнего класса.

Вложенные классы

Используются для:

Это хороший способ группировки классов, которые используются только в одном месте: если класс полезен только для одного другого класса, то логично будет держать их вместе. Вложение таких вспомогательных классов делает код более удобным.

Инкапсуляция: допустим, есть два класса A и B, классу B требуется доступ к свойству класса A, которое может быть приватным. Вложение класса B в класс A решит эту проблему, более того сам класс B можно скрыть от внешнего использования.

Если связь между объектом внутреннего класса и объектом внешнего класса не нужна, можно сделать внутренний класс статическим (static). Такой класс называют вложенным (nested).

Применение статического внутреннего класса означает следующее:

         для создания объекта статического внутреннего класса не нужен объект внешнего класса

         из объекта вложенного класса нельзя обращаться к нестатическим членам внешнего класса

Вложенный класс имеет доступ к членам своего внешнего класса, в том числе и к закрытым членам. Однако, внешний класс не имеет доступа к членам вложенного класса. Вложенный класс при этом является членом внешнего класса.

Статический класс объявляется ключевым словом static. При этом класс должен обращаться к нестатическим членам своего внешнего класса при помощи объекта, т.е. он не может обращаться напрямую на нестатические члены своего внешнего класса.

Вложенный (nested) в интерфейс класс является открытым (public) и статическим (static) даже без явного указания этих модификаторов. Помещая класс внутрь интерфейса, мы показываем, что он является неотъемлемой частью API этого интерфейса и более нигде не используется.

Внутренние классы в Java делятся на виды:

         внутренние классы-члены (member inner classes);

         локальные классы (local classes);

         анонимные классы (anonymous classes).

Собственно нестатические вложенные классы имеют и другое название - внутренние классы (inner classes).

Внутренние классы-члены ассоциируются не с самим внешним классом, а с его экземпляром. При этом они имеют доступ ко всем его полям и методам.

Inner class не может иметь статических объявлений. Вместо этого можно объявить статические методы у обрамляющего класса. Кроме этого, внутри таких классов нельзя объявлять перечисления.

Еще одна важная особенность - интерфейсы в Java не могут быть инстанциированы, соответственно объявить нестатический внутренний интерфейс нельзя (статический можно), так как элементарно не будет объекта для ассоциации с экземпляром внешнего класса.

Для создания объекта:

OuterClass.InnerClassinnerObject = outerObject.new InnerClass();

**Локальные классы (local inner classes)**

Локальные классы (local classes) определяются в блоке Java кода. На практике чаще всего объявление происходит в методе некоторого другого класса. Хотя объявлять локальный класс можно внутри статических и нестатических блоков инициализации.

Как и member классы, локальные классы ассоциируются с экземпляром обрамляющего класса и имеют доступ к его полям и методам. Кроме этого, локальный класс может обращаться к локальным переменным и параметрам метода, если они объявлены с модификатором final.

У локальных классов есть множество ограничений:

         они видны только в пределах блока, в котором объявлены;

         они не могут быть объявлены как private, public, protected или static;

         они не могут иметь внутри себя статических объявлений (полей, методов, классов); исключением являются константы (static final);

**Анонимные классы (anonymous inner classes)**

Анонимный класс (anonymous class) - это локальный класс без имени.

Синтаксис создания анонимного класса базируется на использовании оператора new с именем класса (интерфейса) и телом новосозданного анонимного класса.

Основное ограничение при использовании анонимных классов - это невозможность описания конструктора, так как класс не имеет имени. Аргументы, указанные в скобках, автоматически используются для вызова конструктора базового класса с теми же параметрами.

Так как анонимный класс является локальным классом, он имеет все те же ограничения, что и локальный класс.

Использование анонимных классов оправдано во многих случаях, в частности когда:

         тело класса является очень коротким;

         нужен только один экземпляр класса;

         класс используется в месте его создания или сразу после него;

         имя класса не важно и не облегчает понимание кода.

Анонимный класс может быть как статическим, так и нестатическим. Это напрямую зависит от того, статическим или нестатическим является блок, в котором анонимный класс был объявлен. В примере, который указан выше, анонимный класс будет статическим.

Модификаторы

**Модификаторы:**

         private члены класса доступны только внутри класса

         package-private или default (по умолчанию) члены класса видны внутри пакета

         protected члены класса доступны внутри пакета и в классах-наследниках

         public члены класса доступны всем

**Модификаторы static, abstract и final**

**Static**

         Применяется к внутренним классам, методам, переменным и логическим блокам

         Статические переменные инициализируются во время загрузки класса

         Статические переменные едины для всех объектов класса (одинаковая ссылка)

         Статические методы имеют доступ только к статическим переменным

         К статическим методам и переменным можно обращаться через имя класса

         Не static методы не могут быть переопределены как static

         Локальные переменные не могут быть объявлены как static

         Абстрактные методы не могут быть static

         Static поля не сериализуются

**Abstract**

         Применяется только для методов и классов

         У абстрактных методов нет тела метода

         Является противоположностью final: final класс не может наследоваться, abstract класс обязан наследоваться

         Класс должен быть объявлен как abstract если:

1.       он содержит хотя бы один абстрактный метод

2.       он не предоставляет реализацию наследуемых абстрактных методов

3.       он не предоставляет реализацию методов интерфейса, реализацию которого он объявил

4.       необходимо запретить создание экземпляров класса

**Final**

         Поля не могут быть изменены, методы переопределены

         Классы нельзя наследовать

         Этот модификатор применяется только к классам, методам и переменным (также и к локальным переменным)

         Аргументы методов, обозначенные как final, предназначены только для чтения, при попытке изменения будет ошибка компиляции

         Переменные final не инициализируются по умолчанию, им необходимо явно присвоить значение при объявлении или в конструкторе, иначе – ошибка компиляции

         Если final переменная содержит ссылку на объект, объект может быть изменен, но переменная всегда будет ссылаться на тот же самый объект

         Также это справедливо и для массивов, потому что массивы являются объектами, – массив может быть изменен, а переменная всегда будет ссылаться на тот же самый массив

         Если класс объявлен final и abstract (взаимоисключающие понятия), произойдет ошибка компиляции

         Так как final класс не может наследоваться, его методы никогда не могут быть переопределены

**Конструктор не может быть static, abstract или final**

**Модификаторы strictfp, transient, volatile, synchronized, native**

**Strictfp**

         Применяется для методов и классов

         Обеспечивает выполнение операций над числами типа float и double (с плавающей запятой) по стандарту IEEE 754

**Transient**

         Применяется только для переменных уровня класса (локальные переменные не могут быть объявлены как transient)

         Transient переменные могут не быть final или static.

         Transient переменные не сериализуются

**Volatile**

         Используется только с переменными

         Может использоваться со static переменными

         Не используется с final переменными - Значение переменной, объявленной как volatile, измененное одним потоком, асинхронно меняется и для других потоков

         Применяется в многопоточных приложениях

**Synchronized**

         Применяется только к методам или частям методов

         Используется для контроля доступа к важным частями кода в многопоточных программах

**Native**

         Используется только для методов

         Обозначает, что метод написан на другом языке программирования

         Классы в Java используют много native методов для повышения производительности и доступа к аппаратным средствам

         Можно предавать/возвращать Java объекты из native методов

         Сигнатура метода должна заканчиваться “;”, фигурные скобки вызовут ошибку компиляции

**Особенности в интерфейсах**

         Методы всегда public и abstract, даже если это не объявлено

         Методы не могут быть static, final, strictfp, native, private, protected

         Переменные только public static final, даже если это не объявлено

         Переменные не могут быть strictfp, native, private, protected

         Может только наследовать (extends) другой интерфейс, но не реализовывать интерфейс или класс (implements).

Табличка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Класс | Внутренний класс | Переменная | Метод | Конструктор | Логический блок |
| public | Да | Да (кроме локальных и анонимных классов) | Да | Да | Да | Нет |
| protected | Нет | Да (кроме локальных и анонимных классов) | Да | Да | Да | Нет |
| default | Да | Да | Да (и для локальной переменной) | Да | Да | Да |
| private | Нет | Да (кроме локальных и анонимных классов) | Да | Да | Да | Нет |
| final | Да | Да (кроме анонимных классов) | Да (и для локальной переменной) | Да | Нет | Нет |
| abstract | Да | Да (кроме анонимных классов) | Нет | Да | Нет | Нет |
| static | Нет | Да (кроме локальных и анонимных классов) | Да | Да | Нет | Да |
| native | Нет | Нет | Нет | Да | Нет | Нет |
| transient | Нет | Нет | Да | Нет | Нет | Нет |
| synchronized | Нет | Нет | Нет | Да | Нет | Да (только как часть метода) |
| volatile | Нет | Нет | Да | Нет | Нет | Нет |
| strictfp | Да | Да | Нет | Да | Нет | Нет |