Создание и использование пакетов

* Имена в Java
* Пакеты в Java
* Объявления пакета
* Импорт-выражения
* Базовая система для пакетов
* Соглашения по именованию

Имена в Java

Имена используются в программе для доступа к объявленным ранее "объектам", "элементам", "конструкциям" языка:

* пакеты;
* классы;
* интерфейсы;
* элементы (member) ссылочных типов (поля; методы; внутренние классы и интерфейсы;)
* аргументы (методов; конструкторов; обработчиков ошибок; )
* локальные переменные.

Виды имен

Имена бывают:

* простыми (simple), состоящими из одного идентификатора (они определяются во время объявления)
* составными (qualified), состоящими из последовательности идентификаторов, разделенных точкой.

Пакеты в Java

* Программы на языке Java организованы как наборы пакетов.
* Каждый пакет имеет собственный набор имен для типов, который помогает предотвращать конфликты между именами.
* Пакеты - по существу средство организации классов вместе как группы.

Зачем нужны пакеты?

Классы и интерфейсы необходимо объединять в пакеты по ряду причин:

* Легко определить что типы связаны между собой.
* Легко найти необходимые типы для работы с конкретными задачами. Например найти типы для работы с графикой.
* Имена Ваших типов не будут конфликтовать с именами типов в других пакетах. Каждый пакет создает свое уникальное пространство имен.
* Вы можете гибко управлять правами доступа к типам. Например предоставить полный доступ к типам внутри пакета и запретить доступ за пределами пакета.

Члены пакета

Пакеты могут содержать:

* совокупность классов и интерфейсов.
* другие пакеты:
* каждый уровень, представляет меньшую, более определенную группировку классов.

Модуль компиляции

* Модуль компиляции (complication unit) хранится в текстовом .java-файле и является единичной порцией входных данных для компилятора.
* Он состоит из трех частей:
* Объявление пакета (package), содержащее полностью квалифицированное имя пакета, которому принадлежит модуль компиляции
* Объявление импорта (import), которое разрешает использовать простые имена типов из других пакетов
* Объявления классовых и интерфейсных типов

Объявление пакета

* Объявление пакета появляется внутри модуля компиляции, чтобы указать пакет, которому данный модуль принадлежит.
* Модуль компиляции, который не содержит никакого объявления пакета является частью не именованного пакета (пакета по умолчанию, безымянного пакета ...).

Ключевое слово package

* Объявление пакета записывается с помощью ключевого слова package, после которого указывается полное имя пакета.
* Например в файле java/lang/Object.java:

package java.lang;

* Здесь одновременно объявлен пакет lang, вложенный в пакет java, и указано, что класс Object находится в данном пакете.

Пакет по умолчанию

* Если это выражение объявления пакета отсутствует, то такой модуль компиляции принадлежит пакету по умолчанию.
* Этот пакет по умолчанию обязательно должен поддерживаться реализацией Java-платформы.
* Он не может иметь вложенных пакетов, так как составное имя пакета должно обязательно начинаться с имени пакета верхнего уровня.
* Пакет по умолчанию был введен в Java для облегчения написания очень небольших или временных приложений, для экспериментов.

Доступность пакета

Пакет доступен тогда и только тогда, когда выполняется любое из следующих двух условий:

* доступен модуль компиляции с объявлением этого пакета;
* доступен один из вложенных пакетов этого пакета.

Составные имена

* Чтобы получить составное имя пакета, необходимо к полному имени пакета, в котором он располагается, добавить точку, а затем его собственное простое имя:

java.lang

* Чтобы получить составное имя классов и интерфейсов, надо к составному имени пакета, в котором находится тип, через точку добавить простое имя типа:

java.lang.Object

Область видимости типа

* Область видимости объявления типа - пакет, в котором он располагается
* Внутри пакета допускается обращение к типу по его простому имени
* Из всех других пакетов необходимо обращаться по составному имени

Импорт-выражения

* import-выражения позволяют импортировать типы в модуль компиляции и далее обращаться к ним по простым именам.
* Существует два вида таких выражений:
* импорт одного типа;
* импорт пакета.
* Выражения импорта являются подсказкой для компилятора. Он пользуется ими, чтобы для каждого простого имени типа из другого пакета получить его полное имя, которое и попадает в компилированный код.
* Импортирующие выражения имеют эффект только внутри модуля компиляции, в котором они объявлены.

Выражения импорта

* Выражение, импортирующее один тип, записывается с помощью ключевого слова import и полного имени типа:

import java.net.URL;

* Выражение, импортирующее пакет, включает в себя полное имя пакета:

import java.awt.\*;

* Импортировать вложенный пакет нельзя:

import java.awt.image;

Приведение простого имени к полному

Приведение простого имени к полному состоит из трех шагов:

* сначала просматриваются выражения, импортирующие типы;
* затем другие типы, объявленные в текущем пакете, в том числе в текущем модуле компиляции;
* наконец, просматриваются выражения, импортирующие пакеты.

Пример импорта типа 1

// пример вызовет ошибку компиляции package packagedemo;

import javax.swing.JTable; public class JTable {

}

Пример импорта типа 2 (1 из 3)

package packagedemo; public class Point {

}

Пример импорта типа 2 (2 из 3)

package packagedemo; import java.awt.Point;

class Line {

public static void main(String... args) { System.out.println(Point.class.getName());

}

}

Пример импорта типа 2 (3 из 3)

* Результат java.awt.Point

Пример импорта пакета

* Модифицируем часть предыдущего примера package packagedemo;

import java.awt.\*;

public class LinePackagelmport { public static void main(String... args) { System.out.println(Point.class.getName()); System.out.println(Rectangle.class.getName());

}

}

* Результат packagedemo.Point java.awt.Rectangle

import static

* Константы и статические методы класса можно использовать без указания принадлежности к классу, если применить статический импорт

import static java.lang.Math.\*;

■ ■ ■

System.out.println(2 \* PI \* radius);

Базовая система для пакетов

* Пакеты могут хранится в локальной файловой системе в простых реализациях Java.
* Другие реализации могут использовать распределенную файловую систему или некоторую форму базы данных, чтобы хранить исходные тексты программ и/или двоичный код.

Хранение пакетов в файловой системе (1 из 2)

* Если пакеты, исходный и бинарный код хранятся в файловой системе, то Java может накладывать ограничение на объявления классов в модулях компиляции.
* Создается ошибка компиляции в случае, если описание типа не обнаруживается в файле с названием, составленным из имени типа и расширения (например, java), и при этом:
* тип объявлен как public и, значит, может использоваться из других пакетов;
* тип используется из других модулей компиляции в своем пакете.
* В модуле компиляции может быть максимум один тип отвечающий этим условиям.

Хранение пакетов в файловой системе (2 из 2)

* Для хранения пакетов система Java использует каталоги файловой системы.
* Например файлы .class любых классов, объявленных в качестве составной части пакета MyPackage, должны храниться в каталоге MyPackage.
* Регистр символов имеет значение, а имя каталога должно в точности совпадать имени пакета.

Хранение пакетов в базе данных

* При использовании базы данных для хранения пакетов, нет ограничения на число классов или интерфейсов, объявленных с модификатором public, в каждом модуле компиляции.
* Системы, которые используют базу данных, должны, однако, обеспечить опцию, чтобы преобразовать программу на Java в форму, которая удовлетворяет условиям ограничений, для целей экспорта реализации с файловой основой.

Соглашения по именованию

Соглашения регулируют именование следующих конструкций:

* пакеты;
* типы (классы и интерфейсы);
* методы;
* поля;
* поля-константы;
* локальные переменные и параметры методов и др.

Имена пакетов

* Имя каждого пакета начинается с маленькой буквы и представляет собой, как правило, одно недлинное слово.
* Если требуется составить название из нескольких слов, можно воспользоваться знаком подчеркивания или начинать следующее слово с большой буквы.
* Имя пакета верхнего уровня обычно соответствует доменному имени первого уровня.
* Названия java и javax (Java extension) зарезервированы компанией Sun (читай Oracle) для стандартных пакетов Java.

Имена типов

* Имена типов начинаются с большой буквы и могут состоять из нескольких слов, каждое следующее слово также начинается с большой буквы.
* Human,ArrayIndexOutOfBoundsException
* Аналогично задаются имена интерфейсов, хотя они не обязательно должны быть существительными.
* Часто в именах интерфейсов используется английский суффикс "able":
* Runnable, Serializable, Cloneable

Имена методов

* Имена методов должны быть глаголами и обозначать действия, которые совершает данный метод.
* Имя должно начинаться с маленькой буквы, но может состоять из нескольких слов, причем каждое следующее слово начинается с заглавной буквы.
* Существует ряд принятых названий для методов:
* если методы предназначены для чтения и изменения значения переменной, то их имена начинаются, соответственно, с get и set, например, для переменной size это будут getSize() и setSize();
* метод, возвращающий длину, называется length(), например, в классе String;
* имя метода, который проверяет булевское условие, начинается с is, например, isVisible() у компонента графического пользовательского интерфейса;
* метод, который преобразует величину в формат X, называется toX(), например, метод toString().

Имена констант

* Имена констант состоят из последовательности слов, сокращений, аббревиатур. Записываются они только большими буквами, слова разделяются знаками подчеркивания:
* PI, MIN\_VALUE, MAX\_VALUE
* Иногда константы образуют группу, тогда рекомендуется использовать одно или несколько одинаковых слов в начале имен:
* COLORRED, COLORGREEN, COLOR\_BLUE

Имена полей

* Поля класса имеют имена, записываемые в том же стиле, что и для методов, начинаются с маленькой буквы, могут состоять из нескольких слов, каждое следующее слово начинается с заглавной буквы.
* Имена должны быть существительными, например:
* name, size

Права доступа в Java

* Член класса может быть закрытым (private), защищенным (protected), открытым (public) и по умолчанию.

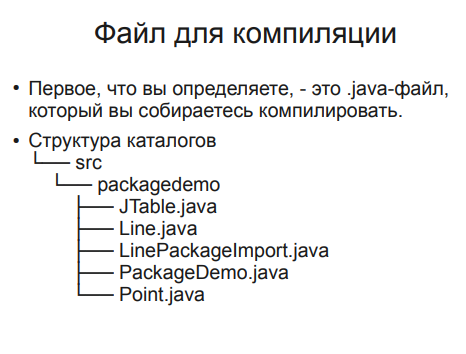


Да что же с этим компилятором ?!!!

* Компиляция Java приложения
* Запуск Java приложения

Компилирование кода Java

* При компилировании кода Java Вам нужно отслеживать несколько связанных, но разных вещей:
* Целевой файл, который вы компилируете.
* Директорию, где компилятор ищет .java-файлы, которые импортирует целевой файл.
* Директорию, где компилятор ищет .class-файлы, которые импортирует целевой файл.
* Директорию, куда компилятор помещает скомпилированный вывод.



Компиляция файла

* Команда

javac src/packagedemo/LinePackagelmport.java

скомпилирует файл LinePackagelmport.java

* или не скомпилирует (если файла нет)

javac: file not found: packagedemo/LinePackagelmport.java

Ошибки компиляции (1 из 2)

Скомпилируем public static void main(String... argsO {

System.println(Point.class.getName());

System.out.println(Rectangle.class.getName());

}

Ошибки компиляции (2 из 2)

src/packagedemo/LinePackageImport.java:11:error: ')' expected public static void main(String... argsO {

1 error

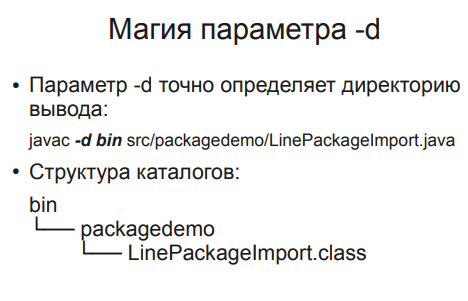
Исправим первую ошибку, и снова скомпилируем

src/packagedemo/LinePackageImport.java:12: error: cannot find symbol System.println(Point.class.getName());

symbol: method println(String) location: class System 1 error

Куда идет вывод

* javac располагает скомпилированный .class-файл в той же директории, где находится .java файл.
* Смешивание файлов .class и .java затрудняет очистку скомпилированных файлов без случайного удаления .java-файлов.
* Это делает чистую сборку проблематичной и ведет к возникновению различных проблем.
* Это также затрудняет упаковку в jar только что скомпилированных .class-файлов при распространении двоичного файла.



Зависимости классов (1 из 2)

Скомпилируем класс B (наследник класса А в другом пакете)

package packagedemo.p2; import packagedemo.p1.A; public class B extends A { public static void main(String... args) { System.out.println(”Hello from B.class”);

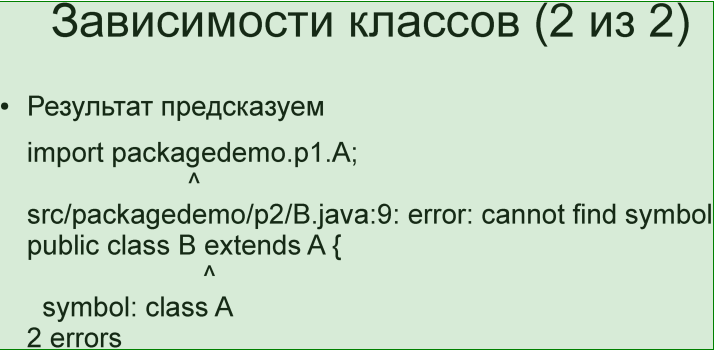
}

}

package packagedemo.p1;

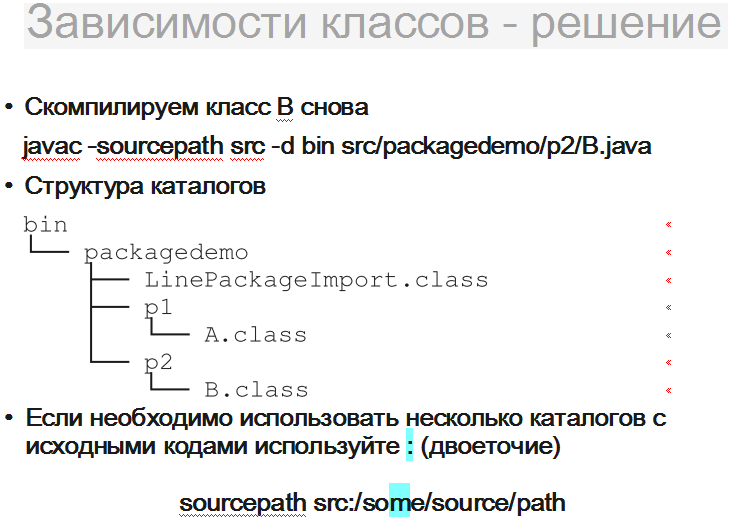
public class A {

}



Параметр sourcepath

* Директория, где Java ищет исходные файлы называется sourcepath.
* Это директория, которая содержит иерархию исходных файлов, организованных в их собственных директориях.
* Это не директория com или src/com/elharo/math.



Запуск приложения

* Для запуска приложения используется команда java

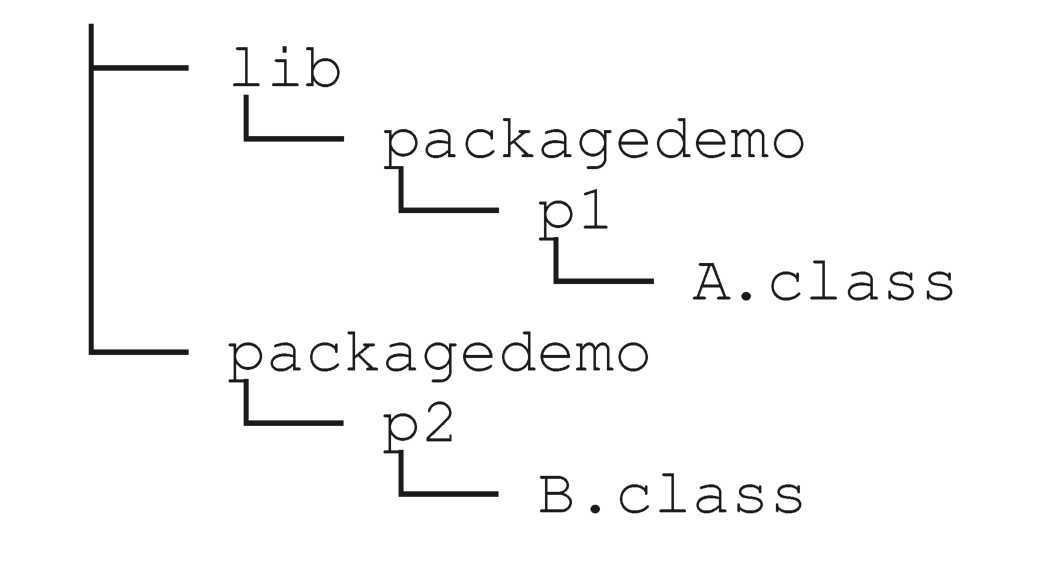
java packagedemo.p2.B

* Результат

Hello from B.class

Внешние зависимости

Перенесем зависимости класса в другой каталог:



3..2..1.. Хьюстон, у нас проблемы

Запустим приложение

java packagedemo.p2.B

Результат

Exception in thread "main" java.lang.NoClassDefFoundError: packagedemo/p1/A at java.lang.ClassLoader.defineClass1(Native Method)

Caused by: java.lang.ClassNotFoundException: packagedemo.p1.A at java.net.URLClassLoader$1 .run(URLClassLoader.java:366)

Запуск программы

* Во время запуска программы вам нужно определить только две вещи:
* classpath.
* Полное (с учетом пакета) имя класса, которое содержит ваш main() method.

Магический classpath

* Classpath в Java — это переменная среды, которая используется виртуальной машиной Java для поиска классов во время их загрузки
* Вы можете переопределить значение Classpath
* Установив переменную среды CLASSPATH
* Использовав параметр коммандной строки -cp или -classpath во время запуска приложения
* Если в Classpath находятся два класса с одинаковым именем то будет загружен тот что идет в Classpath раньше.
* По умолчанию CLASSPATH указывает на текущую директорию.
* ClassNotFoundException — исключение возникающие при попытке динамической загрузки класса не найденного в Classpath
* NoClassDefFoundError - исключение возникающие при попытке загрузки класса доступного при компиляции но не найденного в Classpath при выполнении

Другие места, где располагаются классы

* Текущая рабочая директория
* jre/lib/ext
* jre/lib/endorsed

Повторный запуск приложения

java -cp .:lib packagedemo.p2.B

* Результат Hello from B.class

Собираем приложение 45 минут (1 из 3)

* Создайте два класса A и B:

package org.demo.packagel; public class A { public void print(){

System.out.println("Very Usefull Print");

}

}

Собираем приложение 45 минут (2 из 3)

package org.demo.package2; import org.demo.packagel .A;

public class B { public static void main(String[] args){

A a = new A(); a.print();

}

}

Собираем приложение 45 минут (3 из 3)

* Разместите классы в каталоге исходного кода src
* Разместите классы в каталогах согласно пакетам
* Скомпилируйте приложение так чтобы скомпилированные классы разместились в каталоге bin
* Переместите класс А вместе со структурой каталогов в каталог lib каталога bin
* Запустите класс В

Выполнение приложений Java

* Виртуальная машина Java
* Байт-код Java
* Загрузка классов
* Память программ
* Выполнение программ

Виртуальная машина Java

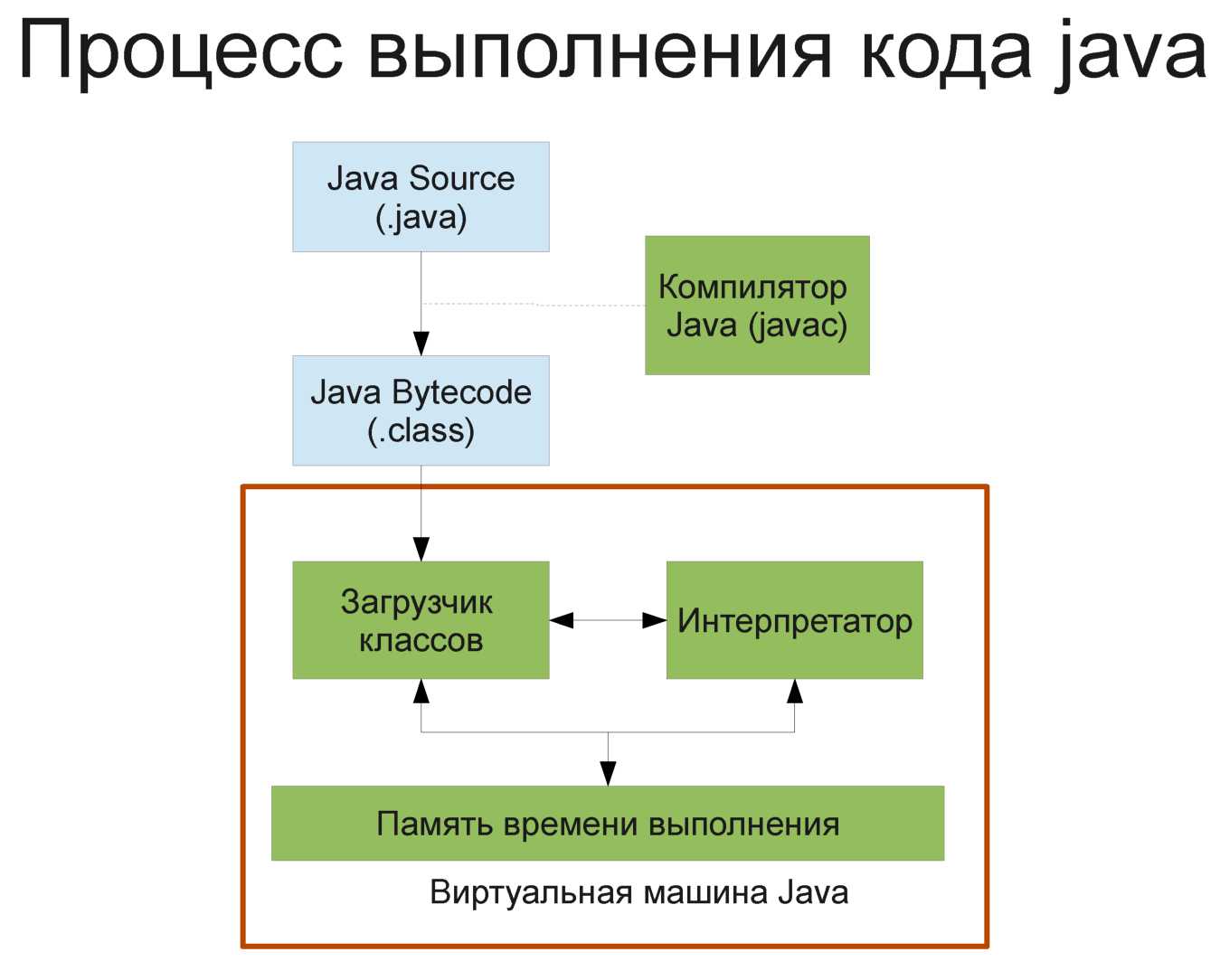
* Виртуальная машина — программная реализация машины (компьютера) способная выполнять команды аналогично физическому компьютеру.
* Основная цель создания виртуальной машины - Write Once Run Anywhere

Особенности JVM

* JVM стековая: в ней отсувствуют регистры для хранения переменных.
* Используются символические ссылки: все типы (классы и интерфейсы за исключением примитивов) доступны через символические ссылки, ане через указатель на область памяти.
* Сборка мусора: явно созданные классы уничтожаются сборщиком мусора.
* Все типы в JVM имеют постоянные размер вне завистимости от платформы.
* Порядок следования байтов ориентирован на передачу по сети: JVM использует порядок от старшего к младшему (big endian)

Байт код

* Байт-код или байткод (англ. byte-code) — машинно­независимый код низкого уровня, генерируемый транслятором и исполняемый интерпретатором.
* Большинство инструкций байт-кода эквивалентны одной или нескольким командам ассемблера.
* Трансляция в байт-код занимает промежуточное положение между компиляцией в машинный код и интерпретацией.



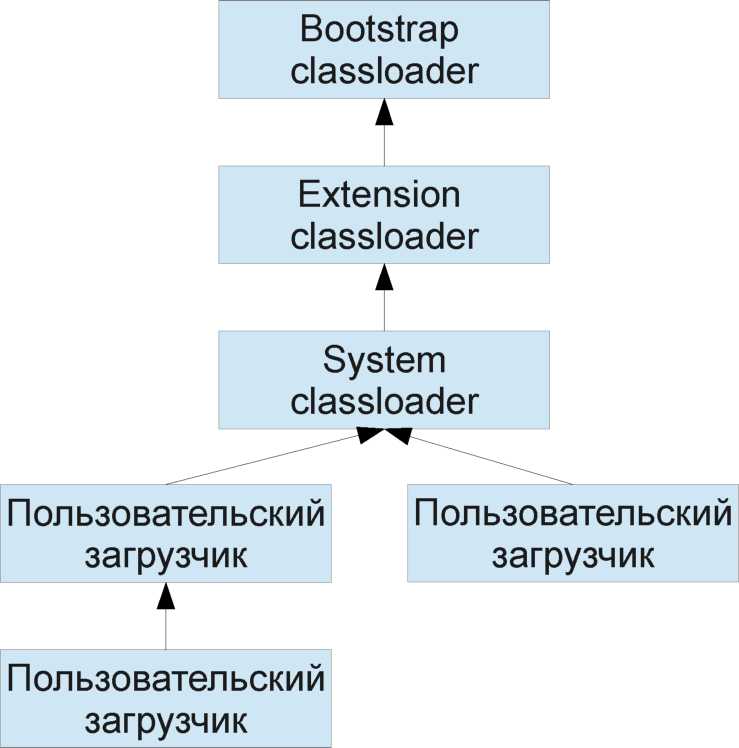
Загрузчик классов

* Java предусматривает динамическую загрузку

классов при помощи загрузчика классов

* Особенности загрузчиков классов:
* Все загрузчика находятся в иерархии
* Предусмотрено делегировании загрузки: при загрузке проверяется не загружен ли класс родительским загрузчиком классов
* Ограничение видимости: загрузчик потомок может использовать классы загруженные родителем но не наоборот
* Выгрузка классов запрещена

Иерархия загрузчиков



Данные времени выполнения



Стек

* Область памяти выделенная Вашему процессу ОС;
* Размер стека фиксирован и определяется на этапе компиляции исходя опций компилятора и задекларированных переменных;
* После старта процесса стек остается неизменным;
* В большинстве случаев стек используется для хранения аргументов методов и локальных переменных.
* Каждый метода имеет свой стек (область в общем стеке процесса).
* Стек метода существует с момента вызова метода и до момента возврата из метода.

Куча(heap)

* Область памяти которая управляется ОС и используется процессом для получения дополнительной памяти во время выполнения;
* Эта область глобальна в пределах процесса;
* Цель этой области памяти предоставить дополнительные ресурсы процессу в виде памяти во время выполнения;
* Объем памяти определяется вызовами вроде new, которые используются для динамического выделения памяти в куче.

Что хранится в куче и стеке?

Рассмотрим пример

class Student{

int age; //поле

String name; //поле

public Student() { this.age = 20; setName("Anonymous");

}

public void setName(String Name) { this.name = Name;

}

}

public class Main{

public static void main(String[] args) {

Student s = new Student(); //локальная переменная

int noStudents = 1; //локальная переменная

}

}

Анализ примера: метод main

public static void main(String[] args) {

Student s = new Student(); int noStudents = 1;

}

В стеке будут находится

* Ссылка args;
* Ссылка на Student, переменная s;
* Целое (4 байта) значение, переменная noStudents;

Анализ примера: конструктор Student()

public Student() { this.age = 20; setName("Anonymous");

}

В стеке будут находится:

* Локальная переменная this
* Все остальные переменные имеют отношения к объекту и будут находится в куче

Анализ примера: метод setName(String Name)

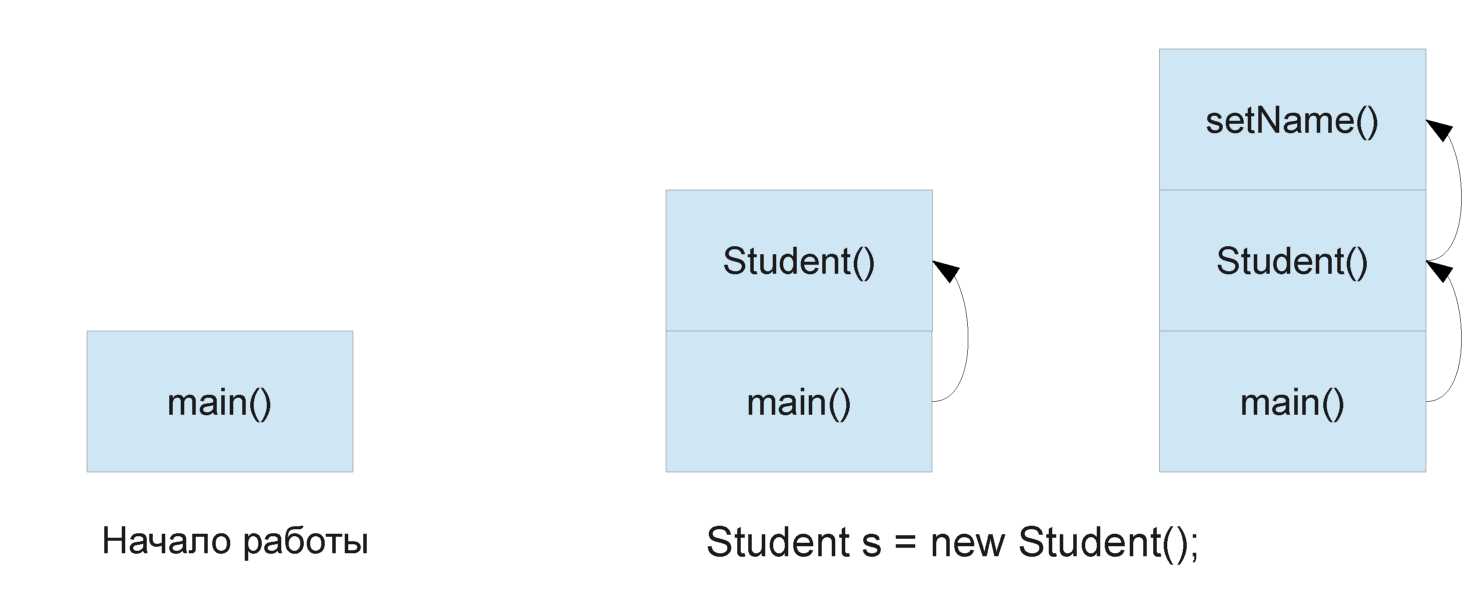
public void setName(String Name) { this.name = Name;

}

В стеке будут находится:

* Ссылка на текущий объект (ссылка на вызывающий объект) (все нестатические методы методы класса вызываются через объект и ссылка на него доступна в методе как this)
* Аргумент Name, ссылка на строку

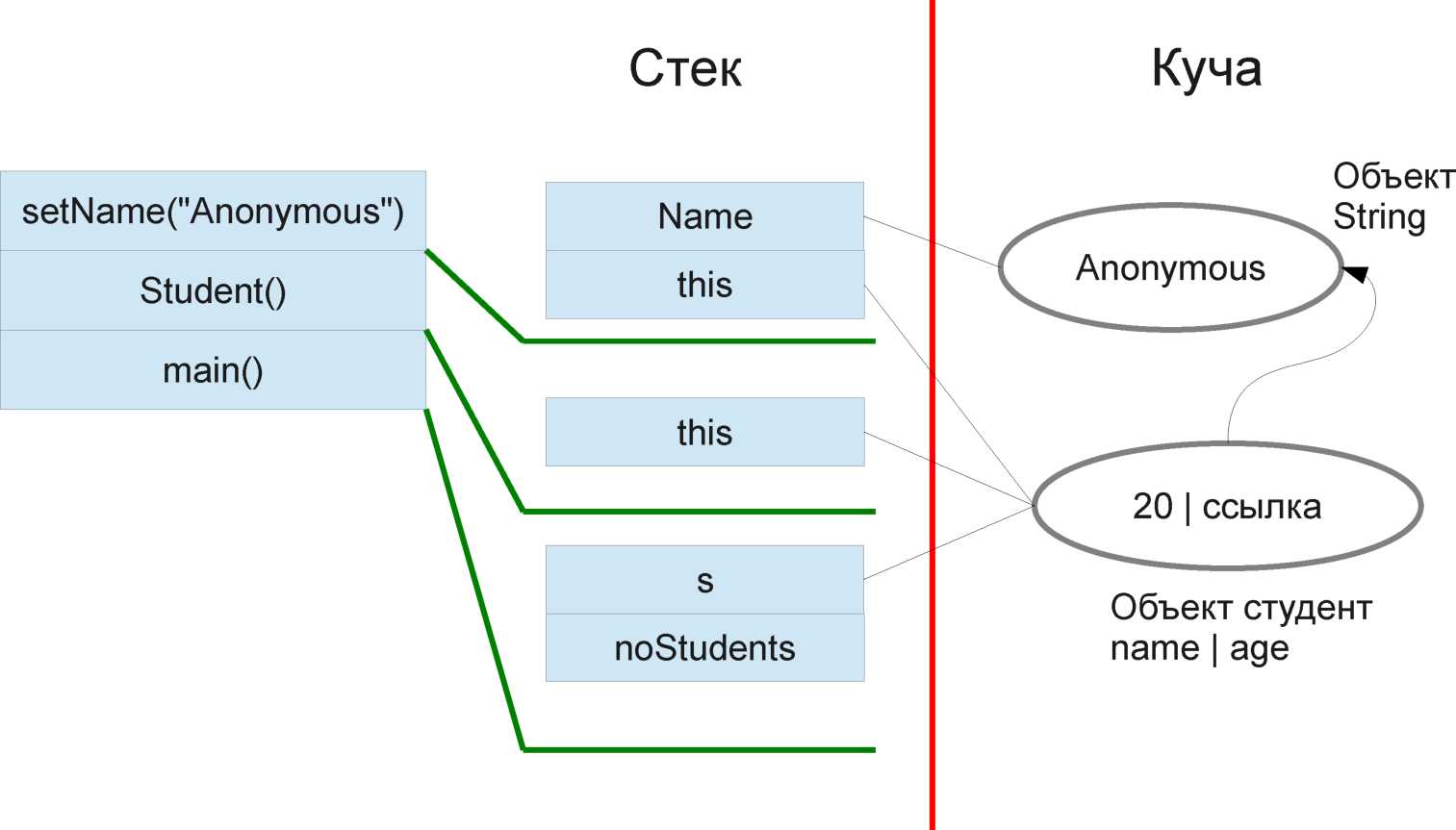
Стек вызовов



Использование кучи

* В куче создаются:
* Память под Student после вызова оператора new и конструктора класса;
* Объект String во время инициализации объекта Student внутри конструктора класса.

Стек вызовов и куча



Интерпретатор

* Интерпретатор — загружает и последовательно исполняет инструкции
* JIT (Just-In-Time) компилятор — преобразует байт-код в инструкции процессора, тем самым ускоряя выполнение.

