БФУ им. И.Канта

Лекция 4:
«ИМЕНА. ПАКЕТЫ»

Дисциплина: «Язык программирования Java»

Преподаватель:
доцент кафедры математического моделирования и информационных систем,
к.т.н. Листопад Сергей Викторович

Калининград, 2013

Имена

Имена (names) используются в программе для доступа к объявленным (declared) ранее "объектам", "элементам", "конструкциям" языка (все эти слова-синонимы были использованы здесь в их общем смысле, а не как термины ООП, например). Конкретнее, в Java имеются имена:

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

Соответственно, все они должны быть объявлены специальным образом, что будет постепенно рассматриваться по ходу курса. Так же объявляются конструкторы

Напомним, что пакеты (packages) в Java – это способ логически группировать классы, что необходимо, поскольку зачастую количество классов в системе составляет несколько тысяч, или даже десятков тысяч. Кроме классов и интерфейсов в пакетах могут находиться вложенные пакеты. Синонимами этого слова в других языках являются библиотека или модуль.

Простые и составные имена. Элементы.

Имена бывают простыми, состоящими из одного идентификатора (они определяются во время объявления) и составными, состоящими из последовательности идентификаторов, разделенных точкой. У пакетов и ссылочных типов (классов, интерфейсов, массивов) есть элементы. Доступ к элементам осуществляется с помощью выражения, состоящего из имен, например, пакета и класса, разделенных точкой.

Элементами пакета являются содержащиеся в нем классы и интерфейсы, а также вложенные пакеты, чтобы получить составное имя такого пакета, необходимо к полному имени пакета, в котором он располагается, добавить точку, а затем его собственное простое имя (java.lang, java.lang.reflect).

Простое имя классов и интерфейсов дается при объявлении, например, Object, String, Point. Чтобы получить составное имя таких типов, надо к составному имени пакета, в котором находится тип, через точку добавить простое имя типа (java.lang.Object, java.lang.reflect.Method, com.myfirm.MainClass).

Для ссылочных типов элементами являются поля и методы, а также внутренние типы (классы и интерфейсы). Элементы могут быть как непосредственно объявлены в классе, так и получены по наследству от родительских классов и интерфейсов, если таковые имеются. Простое имя элементов также дается при инициализации. Например, toString(), PI, InnerClass. Составное имя получается путем объединения простого или составного имени типа, или переменной объектного типа с именем элемента. Например, ref.toString(), java.lang.Math.PI, OuterClass.InnerClass.

Имена и идентификаторы

Уточним разницу между идентификатором (вид лексемы) и именем. Простое имя состоит из одного идентификатора, а составное - из нескольких. Однако не всякий идентификатор входит в состав имени.

- в выражении объявления (declaration) идентификатор еще не является именем. Другими словами, он становится именем после первого появления в коде в месте объявления.
- существует возможность обращаться к полям и методам объектного типа не через имя типа или объектной переменной, а через ссылку на объект, полученную в результате выполнения выражения: country.getCity().getStreet(); В данном примере getStreet является не именем, а идентификатором, так как соответствующий метод вызывается у объекта, полученного в результате вызова метода getCity(). Причем country.getCity как раз является составным именем метода.

Идентификаторы также используются для названий меток (label):

```
num:
  for (int num = 2; num <= 100; num++) {
    int n = (int)Math.sqrt(num)+1;
    while (--n != 1) {
      if (num%n==0) {
        continue num;
      }
    }
    System.out.print(num+" ");
  }
```

Область видимости

Чтобы не заставлять программистов, совместно работающих над различными классами одной системы, координировать имена, которые они дают различным конструкциям языка, у каждого имени есть область видимости (scope). Если обращение, например, к полю, идет из части кода, попадающей в область видимости его имени, то можно пользоваться простым именем, если нет – необходимо применять составное:

```
class Point {
  int x,y;
  int getX() {
    return x; // простое имя
  }}
class Test {
  void main() {
    Point p = new Point();
    p.x=3; // составное имя
  }}
```

Видно, что к полю x изнутри класса можно обращаться по простому имени. К нему же из другого класса можно обратиться только по составному имени. Оно составляется из имени переменной, ссылающейся на объект, и имени поля.

Пакеты

Программа на Java представляет собой набор пакетов (packages). Каждый пакет может включать вложенные пакеты, то есть они образуют иерархическую систему.

Кроме того, пакеты могут содержать классы и интерфейсы и таким образом группируют типы. Это необходимо сразу для нескольких целей. Во-первых, чисто физически невозможно работать с большим количеством классов, если они "свалены в кучу". Во-вторых, модульная декомпозиция облегчает проектирование системы. К тому же, как будет показано ниже, существует специальный уровень доступа, позволяющий типам из одного пакета более тесно взаимодействовать друг с другом, чем с классами из других пакетов. Таким образом, с помощью пакетов производится логическая группировка типов. Из ООП известно, что большая связность системы, то есть среднее количество классов, с которыми взаимодействует каждый класс, заметно усложняет развитие и поддержку такой системы. Используя пакеты, гораздо проще организовать эффективное взаимодействие подсистем друг с другом.

Наконец, каждый пакет имеет свое пространство имен, что позволяет создавать одноименные классы в различных пакетах. Таким образом, разработчикам не приходится тратить время на разрешение конфликта имен.

Элементы пакета

Элементами пакета являются вложенные пакеты и типы (классы и интерфейсы). Одноименные элементы запрещены, то есть не может быть одноименных класса и интерфейса, или вложенного пакета и типа. В противном случае возникнет ошибка компиляции.

Например, в JDK 1.0 пакет java содержал пакеты applet, awt, io, lang, net, util и не содержал ни одного типа. В пакет java.awt входил вложенный пакет image и 46 классов и интерфейсов.

Составное имя любого элемента пакета – это составное имя этого пакета плюс простое имя элемента. Например, для класса Object в пакете java.lang составным именем будет java.lang.Object, а для пакета image в пакете java.awt – java.awt.image.

Иерархическая структура пакетов была введена для удобства организации связанных пакетов, однако вложенные пакеты, или соседние, то есть вложенные в один и тот же пакет, не имеют никаких дополнительных связей между собой, кроме ограничения на несовпадение имен. Например, пакеты space.sun, space.sun.ray, space.moon и factory.store совершенно "равны" между собой и типы одного из этих пакетов не имеют никакого особенного доступа к типам других пакетов.

Платформенная поддержка пакетов

Простейшим способом организации пакетов и типов является обычная файловая структура. Рассмотрим пример, когда все пакеты, исходный и бинарный код располагаются в одном каталоге и его подкаталогах. В этом корневом каталоге должна быть папка java, соответствующая основному пакету языка, а в ней, в свою очередь, вложенные папки applet, awt, io, lang, net, util.

Как известно, исходный код располагается в файлах с расширением .java, а бинарный – с расширением .class. Таким образом, содержимое папки sunsystem может выглядеть следующим образом:

Moon.java
Moon.class
Sun.java
Sun.class
Test.java
Test.class

Другими словами, исходный код классов

space.sunsystem.Moon
space.sunsystem.Sun
space.sunsystem.Test

хранится в файлах

space\sunsystem\Moon.java
space\sunsystem\Sun.java
space\sunsystem\Test.java

а бинарный код – в соответствующих .class -файлах. Преобразование имен пакетов в файловые пути потребовало замены разделителя . (точки) на символ-разделитель файлов (для Windows это обратный слэш \). Такое преобразование может выполнить как компилятор для поиска исходных текстов и бинарного кода, так и виртуальная машина для загрузки классов и интерфейсов.

Classpath

Далеко не всегда удобно хранить все файлы в одном каталоге. Зачастую классы находятся в разных местах, а некоторые могут даже распространяться в виде архивов, для ускорения загрузки через сеть. Копировать все такие файлы в одну папку было бы крайне затруднительно.

Поэтому Java использует специальную переменную окружения, которая называется classpath. Ее значение должно состоять из путей к каталогам или архивам, разделенных точкой с запятой. С версии 1.1 поддерживаются архивы типов ZIP и JAR. Например, переменная classpath может иметь такое значение:

```
.;c:\java\classes;d:\lib\3Dengine.zip;d:\lib\fire.jar
```

В результате все указанные каталоги и содержимое всех архивов "добавляется" к исходному корневому каталогу. Java в поисках класса будет искать его во всех указанных папках и архивах по порядку. Такая конструкция таит в себе опасности: если разрабатываемые классы хранятся в некотором каталоге и он указан в classpath позже, чем некий другой каталог, в котором обнаруживаются одноименные типы, разобраться в такой ситуации будет непросто. В классы будут вноситься изменения, которые никак не проявляются при запуске из-за того, что Java на самом деле загружает одни и те же файлы из посторонней папки.

Модуль компиляции

Модуль компиляции (compilation unit) хранится в текстовом .java - файле и является единичной порцией входных данных для компилятора. Он состоит из трех частей:

- объявление пакета ;
- import -выражения;
- объявления верхнего уровня.

Объявление пакета одновременно указывает, какому пакету будут принадлежать все объявляемые ниже типы. Если данное выражение отсутствует, значит, эти классы располагаются в безымянном пакете (другое название – пакет по умолчанию). Import -выражения позволяют обращаться к типам из других пакетов по их простым именам, "импортировать" их. Эти выражения также необязательны. Наконец, объявления верхнего уровня содержат объявления одного или нескольких типов. Название "верхнего уровня" противопоставляет эти классы и интерфейсы, располагающиеся в пакетах, внутренним типам, которые являются элементами и располагаются внутри других типов. Как ни странно, эта часть также является необязательной, в том смысле, что в случае ее отсутствия компилятор не выдаст ошибки. Однако никаких .class -файлов сгенерировано тоже не будет.

Доступность модулей компиляции определяется поддержкой платформы, т.к. утилиты Java являются обычными программами, которые исполняются операционной системой по общим правилам.

Объявление пакета

Объявление пакета записывается с помощью ключевого слова package, после которого указывается полное имя пакета: package java.lang; Это одновременно служит объявлением пакета lang, вложенного в пакет java, и указанием, что объявляемый ниже класс Object находится в данном пакете. Так складывается полное имя класса java.lang.Object.

Если это выражение отсутствует, то такой модуль компиляции принадлежит безымянному пакету, который не может иметь вложенных пакетов, так как составное имя пакета должно обязательно начинаться с имени пакета верхнего уровня.

Таким образом, самая простая программа может выглядеть следующим образом:

```
class Simple {
  public static void main(String s[]) {
    System.out.println("Hello!");
  }}
```

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

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

Таким образом, для следующего кода:

```
package space.star;
class Sun {}
```

если файл, который хранит этот модуль компиляции, доступен Java-платформе, то пакеты space и вложенный в него star (полное название space.star) также становятся доступны для Java.

Если пакет доступен, то область видимости его объявления – все доступные модули компиляции. Проще говоря, все существующие пакеты доступны для всех классов, никаких ограничений на доступ к пакетам в Java нет. Требуется, чтобы пакеты java.lang и java.io, а значит, и java, всегда были доступны для Java-платформы, поскольку они содержат классы, необходимые для работы любого приложения.

Импорт-выражения

Область видимости объявления типа - пакет, в котором он располагается, т.е. внутри данного пакета допускается обращение к типу по его простому имени. Из всех других пакетов необходимо обращаться по составному имени. Поскольку пакеты могут иметь довольно длинные имена, а тип может многократно использоваться в модуле компиляции, такое ограничение может привести к усложнению исходного кода и сложностям в разработке. Для решения этой проблемы вводятся import -выражения, позволяющие импортировать типы в модуль компиляции и далее обращаться к ним по простым именам. Существует два вида таких выражений:

- импорт одного типа import java.net.URL;
- импорт пакета import java.awt.*.

Допускается одновременно импортировать пакет и какой-нибудь тип из него:

```
import java.awt.*;
import java.awt.Point;
```

Может возникнуть вопрос, как же лучше поступать – импортировать типы по отдельности или весь пакет сразу? Есть ли какая-нибудь разница в этих подходах? Разница заключается в алгоритме работы компилятора, который приводит каждое простое имя к полному:

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

Таким образом, если тип явно импортирован, то невозможно ни объявление нового типа с таким же именем, ни доступ по простому имени к одноименному типу в текущем пакете:

```
// пример вызовет ошибку компиляции
package my_geom;I
mport java.awt.Point;
class Point {}
```

Импорт пакетов

Импорт пакета называют "импорт по требованию", т.к. никакой "загрузки" типов импортированного пакета при указании импортирующего выражения не происходит, их полные имена подставляются по мере использования простых имен в коде. Можно импортировать пакет и задействовать только один тип (или ни одного) из него:

```
package my_geom;
import java.awt.*;
class Line {
  void main() {
    System.out.println(new Point());
    System.out.println(new Rectangle());
  }}
//результатом будет: my_geom.Point@92d342
//java.awt.Rectangle[x=0,y=0,width=0,height=0]
```

Тип Point нашелся в текущем пакете, поэтому компилятору не пришлось выполнять поиск по пакету java.awt. Второй объект порождается от класса Rectangle, которого не существует в текущем пакете, зато он обнаруживается в java.awt. Также корректен теперь пример:

```
package my_geom;
import java.awt.*;
class Point {}
```

Импорт пакета не препятствует объявлению новых типов или обращению к существующим типам текущего пакета по простым именам. Если нужно работать именно с внешними типами, то можно воспользоваться импортом типа, или обращаться к ним по полным именам. Считается, что импорт конкретных типов помогает при прочтении кода сразу понять, какие внешние классы и интерфейсы используются в этом модуле компиляции. Однако полностью полагаться на такое соображение не стоит, так как возможны случаи, когда импортированные типы не используются и, напротив, в коде стоит обращение к другим типам по полному имени.

Объявление верхнего уровня

Модуль компиляции может содержать одно или несколько объявлений классов и интерфейсов:

```
package first;
class FirstClass {}
interface MyInterface {}
```

Область видимости типа - пакет, в котором он описан. Из других пакетов к типу можно обращаться либо по составному имени, либо с помощью импортирующих выражений. По умолчанию тип объявляется доступным только для других типов своего пакета. Чтобы другие пакеты могли использовать его, можно указать ключевое слово public:

```
package second;
public class OpenClass {}
public interface PublicInterface {}
```

Объявления верхнего уровня описывают классы и интерфейсы, хранящиеся в пакетах. В версии Java 1.1 были введены внутренние (inner) типы, которые объявляются внутри других типов и являются их элементами наряду с полями и методами. Если пакеты, исходный и бинарный код хранятся в файловой системе, то Java может накладывать ограничение на объявления классов в модулях компиляции. Это ограничение создает ошибку 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.

Т.е. в модуле компиляции может быть максимум один public тип, и его имя и имя файла должны совпадать. Если же в нем есть не-public типы, имена которых не совпадают с именем файла, то они должны использоваться только внутри этого модуля компиляции.

Уникальность имен пакетов

Поскольку Java создавался как язык, предназначенный для распространения приложений через Internet, а приложения состоят из структуры пакетов, необходимо предпринять некоторые усилия, чтобы не произошел конфликт имен. Имена двух используемых пакетов могут совпасть по прошествии значительного времени после их создания.

Поэтому создатели Java предлагают следующий способ уникального именования пакетов. Если программа создается разработчиком, у которого есть Internet-сайт, либо же он работает на организацию, у которой имеется сайт, и доменное имя такого сайта, например, company.com, то имена пакетов должны начинаться с этих же слов, выписанных в обратном порядке: com.company. Дальнейшие вложенные пакеты могут носить названия подразделений компании, пакетов, фамилии разработчиков, имена компьютеров и т.д.

Таким образом, пакет верхнего уровня всегда записывается ASCII-буквами в нижнем регистре и может иметь одно из следующих имен:

- трехбуквенные com, edu, gov, mil, net, org, int (этот список расширяется);
- двухбуквенные, обозначающие имена стран, такие как ru, su, de, uk и другие.

Если имя сайта противоречит требованиям к идентификаторам Java, то можно предпринять следующие шаги:

- если в имени стоит запрещенный символ, например, тире, то его можно заменить знаком подчеркивания;
- если имя совпадает с зарезервированным словом, можно в конце добавить знак подчеркивания;
- если имя начинается с цифры, можно в начале добавить знак подчеркивания.

Примеры имен пакетов, составленных по таким правилам:

- com.sun.image.codec.jpegorg.omg.CORBA.ORBPackageoracle.jdbc.driver.OracleDriver

Однако, конечно, никто не требует, чтобы Java-пакеты были обязательно доступны на Internet-сайте, который дал им имя. Скорее была сделана попытка воспользоваться существующей системой имен вместо того, чтобы создавать новую для именования библиотек.

Область видимости имен

Областью видимости объявления некоторого элемента языка называется часть программы, откуда допускается обращение к этому элементу по простому имени.

Область видимости доступного пакета – вся программа, т.е. любой класс может использовать доступный пакет. Однако необходимо помнить, что обращаться к пакету можно только по его полному составному имени. К пакету java.lang ни из какого места нельзя обратиться как к просто lang.

Областью видимости импортированного типа являются все объявления верхнего уровня в этом модуле компиляции.

Областью видимости типа (класса или интерфейса) верхнего уровня является пакет, в котором он объявлен. Из других пакетов доступ возможен либо по составному имени, либо с помощью импортирующего выражения, которое помогает компилятору воссоздать составное имя.

Область видимости элементов классов или интерфейсов – это все тело типа, в котором они объявлены. Если обращение к этим элементам происходит из другого типа, необходимо воспользоваться составным именем. Имя может быть составлено из простого или составного имени типа, имени объектной переменной или ключевых слов super или this, после чего через точку указывается простое имя элемента.

Аргументы метода, конструктора или обработчика ошибок видны только внутри этих конструкций и не могут быть доступны извне.

Область видимости локальных переменных начинается с момента инициализации и до конца блока, в котором они объявлены. В отличие от полей типов, локальные переменные не имеют значений по умолчанию и должны инициализироваться явно.

```
int x;
for (int i=0; i<10; i++) {
   int t=5+i;
}// здесь переменная t уже недоступна, так как блок, в котором она была
// объявлена, уже завершен, а переменная x еще недоступна, так как пока не была
// инициализирована
```

"Затеняющее" объявление

Перейдем к проблеме перекрытия имен полей класса и локальных переменных. Пример:

```
class Human {
    int age;   // возраст
    int getAge() {
       return age;
    }
    void setAge(int age) {
       age=age; // ???
    } }
```

Во-первых, рассмотрим, из-за чего возникла конфликтная ситуация. Есть два элемента языка – аргумент метода и поле класса, области видимости которых пересеклись. Область видимости поля класса больше, она охватывает все тело класса, в то время как область видимости аргумента метода включает только сам метод. В таком случае внутри области пересечения по простому имени доступен именно аргумент метода, а поле класса "затеняется" (shadowing) объявлением параметра метода.

Остается вопрос, как в такой ситуации все же обратиться к полю класса. Если доступ по простому имени невозможен, надо воспользоваться составным. Здесь удобнее всего применить специальное ключевое слово this. Если вызвать метод setAge() у объекта класса Human и использовать в этом методе слово this, то его значение будет ссылкой на данный объект:

```
class Human {
   int age;   // возраст
   void setAge(int age) {
      this.age=age;   // верное присвоение!
   }}
```

"Заслоняющее" объявление

Может возникнуть ситуация, когда простое имя может быть одновременно рассмотрено как имя переменной, типа или пакета:

```
import java.awt.*;
public class Obscuring {
   static Point Test = new Point(3,2);
   public static void main (String s[]) {
      print(Test.x);
   }}
class Test {
   static int x = -5;
}
```

В методе main() простое имя Test одновременно обозначает имя поля класса Obscuring и имя другого типа, находящегося в том же пакете,– Test. С помощью этого имени происходит обращение к полю x, которое определено и в классе java.awt.Point и Test.

Результатом этого примера станет 3, то есть переменная имеет более высокий приоритет. В свою очередь, тип имеет более высокий приоритет, чем пакет. Таким образом, обращение к доступному в обычных условиях типу или пакету может оказаться невозможным, если есть объявление одноименной переменной или типа, имеющее более высокий приоритет. Такое объявление называется "заслоняющим" (obscuring).

Эта проблема скорее всего не возникнет, если следовать соглашениям по именованию элементов языка Java.

Соглашения по именованию

Соглашения регулируют именование следующих конструкций:

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

Имя каждого пакета начинается с маленькой буквы и представляет собой 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 для стандартных пакетов Java.

Имена типов начинаются с большой буквы и могут состоять из нескольких слов, каждое следующее слово также начинается с большой буквы. Конечно, надо стремиться к тому, чтобы имена были описательными, "говорящими".

Имена классов, как правило, являются существительными:

Human

HighGreenOak

ArrayIndexOutOfBoundsException

Аналогично задаются имена интерфейсов, хотя они не обязательно должны быть существительными. Часто используется английский суффикс "able":

Runnable

Serializable

Cloneable

Проблема "заслоняющего" объявления (obscuring) для типов встречается редко, так как имена пакетов и локальных переменных (параметров) начинаются с маленькой буквы, а типов – с большой.

Соглашения по именованию

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 F, называется toF(), например, метод toString(), который приводит любой объект к строке.

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, поле name в классе Human, или size в классе Planet. Поля могут быть константами, если в их объявлении стоит ключевое слово final. Их имена 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:

COLOR_RED
COLOR_GREEN
COLOR_BLUE

Соглашения по именованию

Имена локальных переменных и параметров методов, конструкторов и обработчиков ошибок, как правило, довольно короткие, но, тем не менее, должны быть осмыслены. Например, можно использовать аббревиатуру (имя cp для ссылки на экземпляр класса ColorPoint) или сокращение (buf для buffer).

Распространенные однобуквенные сокращения:

```
byte b;
char c;
int i,j,k;
long l;
float f;
double d;
Object o;
String s;
Exception e;   // объект, представляющий  ошибку в Java
```

Двух- и трехбуквенные имена не должны совпадать с принятыми доменными именами первого уровня Internet-сайтов.