Урок 3

Пакеты. Работа со строками

Пакеты. Символьные строки.

Пакеты

Большая часть кода, которую мы писали до этого, работала в одном пространстве имен (**namespace**). Чтобы избежать конфликтов, приходилось выдумывать уникальные наименования.
Пакеты — это механизм для работы с пространством имен и ограничения видимости. У каждого файла в Java четыре характерных внутренних части, из которых мы использовали преимущественно одну:

- одиночный оператор **package**;
- множественный оператор **import**;
- одиночное объявление открытого (**public**) класса;
- множество закрытых (**private**) классов пакета.

Первая строка в файле Java (она необязательна, но при указании должна идти первой) — это оператор **package**. Он сообщает компилятору, в каком пакете должны лоцироваться классы, содержащиеся в данном файле. Если **package** не указан, классы создаются в пространстве имен без названия. Наименование пакета должно соответствовать его расположению в директориях. Если есть пакет с именем **package java.awt.image**, код этого класса должен лежать в **java/awt/image**.

При рассмотрении пути компилятору можно изменить корневую директорию через настройку **CLASSPATH**. С ее помощью можно указать несколько корневых каталогов для иерархии пакетов (через «**;**», как в обычном **PATH**).

После оператора **package** — но до начала описания классов в коде — может присутствовать список операторов **import**.

```
import package1[.package2].(имякласса|*);
```

Здесь **package1** — это имя пакета верхнего уровня, **package2** — необязательное имя вложенного пакета (таких может быть сколько угодно). В конце пишется либо имя конкретного подключаемого класса, либо звездочка, которая обозначает возможность подключения любого класса из указанного пакета. Постоянно использовать **import** со звездочкой нежелательно — это может значительно увеличить время компиляции кода.

Обратите внимание, что встроенные классы, входящие в JDK, располагаются в пакете **java**. Базовые функции размещены в **java.lang**. Весь этот пакет автоматически подключается компилятором во все программы, что эквивалентно размещению в каждой программе строчки **import java.lang.*;**.

Если в двух пакетах, которые подключены при помощи **import** со звездочкой, есть классы с одинаковыми именами — при попытке использования одного из них сразу придет сообщение об ошибке. В таком случае нужно переписать операторы **import** с явным указанием пакета, который содержит нужный класс.

Java предоставляет несколько уровней области видимости. Из-за пакетов Java должна уметь работать еще с четырьмя категориями видимости между элементами классов:

- Подклассы в том же пакете;

- Классы, которые не являются подклассами, но входят в тот же пакет;
- Подклассы в различных пакетах;
- Классы, которые не являются подклассами и не входят в тот же пакет.

В Java есть три уровня доступа, определяемых ключевыми словами **private** (закрытый), **public** (открытый) и **protected** (защищенный), которые употребляются в различных комбинациях. В таблице показано, как определить доступность переменной, исходя из комбинации модификаторов (столбец) в указанном месте (строка).

	private	**модификатор отсутствует**	**private protected**	**protected**	**public**
тот же класс	да	да	да	да	да
подкласс в том же пакете	нет	да	да	да	да
независимый класс в том же пакете	нет	да	нет	да	да
подкласс в другом пакете	нет	нет	да	да	да
независимый класс в другом пакете	нет	нет	нет	нет	да

Символьные строки

Во многих языках программирования строки — это, по сути, последовательность символов. Но в Java они также являются объектами класса **String**. Особенность работы строк в этом языке заключается в том, что в памяти они неизменны. Даже если поменять один символ в существующей строке, все равно неявно создается новая в памяти.

Экземпляр строки добавляется в Java множеством способов.

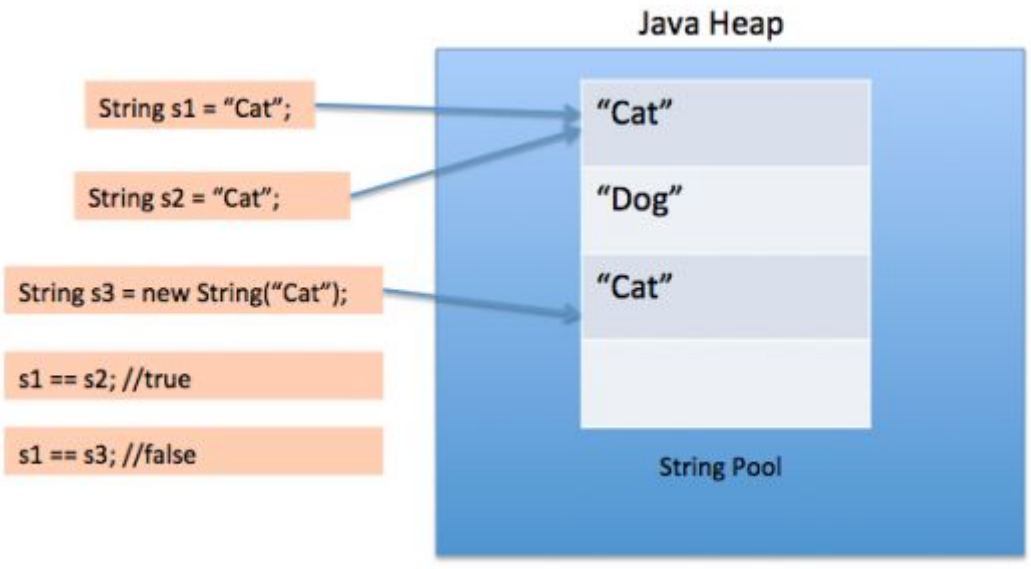
```
public class Main {
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = new String("Home");
        String s3 = new String(new char[]{'A', 'B', 'C'});
        String s4 = new String(s3);
        String s5 = new String(new byte[]{65, 66, 67});
        String s6 = new String(new byte[]{0, 65, 0, 66},
StandardCharsets.UTF_16);
        System.out.printf("s1 = %s, s2 = %s, s3 = %s, s4 = %s, s5 = %s, s6 =
%s", s1, s2, s3, s4, s5, s6);
    }
}
Результат:
s1 = Java, s2 = Home, s3 = ABC, s4 = ABC, s5 = ABC, s6 = AB
```

Строки можно наращивать или конкатенировать. Это, по сути, операция сложения строк.

```
public class Main {
    public static void main(String[] args) {
        String a1 = "Hello ";
        String a2 = "World";
        String a3 = a1 + a2;
        System.out.println(a3);
        String b1 = "Число 10: ";
        int b2 = 10;
        String b3 = b1 + b2;
        System.out.println(b3);
        String c1 = "Home";
        String c2 = "[" + c1 + "] = " + 1;
        System.out.println(c2);
    }
}
Результат:
Hello World
Число 10: 10
[Home] = 1
```

С ее помощью можно порождать новые объекты в памяти, сцепляя строки — не только с другими строками, но и с иными типами данных, у которых будет автоматически вызван метод преобразования в строку **toString()**. Здесь надо быть осторожным и не забывать о приоритетах операций.

В Java также есть пулы строк, располагающиеся в **Heap** (в куче). Пул строк связан с неизменностью строк в памяти. В определенных случаях ссылки на строковые объекты будут указывать на одну и ту же область в пуле строк.



Можно принудительно создать новую сущность в пуле строк при помощи оператора **new**.

Если часто модифицировать строки, память будет расходоваться на их создание. Для решения этой проблемы предназначены построители строк — **StringBuffer** и **StringBuilder**.

Класс **StringBuffer** позволяет модифицировать строку до формирования конкретного объекта **String**, который будет неизменным. Благодаря классу можно добавлять и удалять подстроки. **StringBuffer** потокобезопасен, поэтому его можно использовать в многопоточных приложениях. Но он работает гораздо медленнее, чем его аналог для однопоточных продуктов — **StringBuilder**.