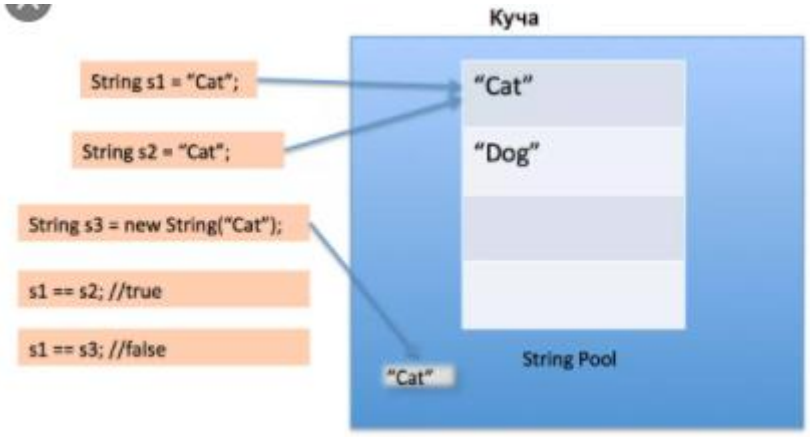
Strings

1. String s = " abc "; String y = new String("abc");
   Конструкторов у класса большое множество, такие как:

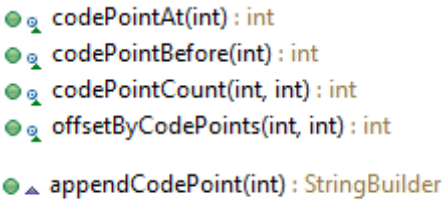
   a. создать объект, содержащий пустую строку
   b. создать копию строковой переменной
   c. создать строку на основе массива символов
   d. создать строку на основе массива байтов (с учетом кодировок) и тд. тп.

   Строковые литералы — это экземпляры класса String (например, "abc");
   Упрощенное создание объекта класса String – это присвоение ему значения строкового литерала.
2. Состояние объекта типа String изменить нельзя.
   При попытке изменения, вы меняете его ссылку на другую.
   От класса String наследоваться нельзя, так как он объявлен как final.
   Immutable гарантирует, что вы не сможете изменить состояние созданного вами объекта
3. Кодировка – преобразование символа определенным порядком цифр в памяти.
   Кодировки в java бывают: UTF-8, UTF-16, Windows-1251 и тд. тп.
   Charset koi8 = Charset.forName("KOI8-R");
   Charset windows1251 = Charset.forName("Windows-1251");
   byte[] buffer = new byte[1000];
   inputStream.read(buffer);
   String s = new String(buffer, koi8);
   buffer = s.getBytes(windows1251);
   outputStream.write(buffer);
4. Пул строк (String Pool) — это множество строк в кучи (Java Heap Memory)



Когда мы используем двойные кавычки, чтобы создать новую строку, то первым делом идет поиск строки с таким же значением в пуле строк. Если java такую строку нашла, то возвращает ссылку, в противном случае создается новая строка в пуле, а затем возвращается ссылка.

Однако использование оператора new заставляет класс String создать новый объект String. После этого можем использовать метод intern(), чтобы поместить этот объект в пул строк или обратиться к другому объекту из пула строк, который имеет такое же значение.
Jdk 7 - интернированные строки выделяются в основной части кучи
Jdk 6 - интернированные строки выделяются в постоянном поколении кучи Java;

5. Класс String представляет собой неизменяемые последовательности символов постоянной длины и частое использование объектов класса занимают много места в памяти. Класс StringBuffer представляет расширяемые и доступные для изменений последовательности символов, позволяя вставлять символы и подстроки в существующую строку и в любом месте. Данный класс гораздо экономичнее в плане потребления памяти и настоятельно рекомендуется к использованию.
   Класс StringBuilder идентичен классу StringBuffer и обладает большей производительностью. Однако он не синхронизирован, поэтому его не нужно использовать в тех случаях, когда к изменяемой строке обращаются несколько потоков.
   sb1.toString().equals(sb2.toString()), так как в этом случае будет сравниваться значение, а не ссылки на объекты.
6. Unicode — это международный стандарт кодировки символов
7. 

   Кодовые точки обычно используются в суррогатных парах

Regular Expressions

1. Регулярное выражение (regular expression/regex/regexp) — это строка, которая является шаблоном (pattern), описывающим некий набор строк. Основой синтаксиса регулярных выражений является тот факт, что некоторые символы, встречающиеся в строке, рассматриваются не как обычные символы, а как имеющие специальное значение т.н. метасимволы.
   классы символов: диапазонные классы, инверсию, объединение, пересечение и вычитание классов.
   Метасимволы для обозначения количества символов – квантификаторы. Особенностью квантификаторов является возможность использования их в разных режимах: жадном, сверхжадном и ленивом

Метасимвол	Назначение
?	один или отсутствует
*	ноль или более раз
+	один или более раз
{n}	n раз
{n,}	n раз и более
{n,m}	не менее n раз и не более m раз

Таблица:

^	Совпадает начало строки
$	Совпадает конец строки
.	Совпадают все символы, кроме символа новой строки.

[…]	Совпадает любой отдельный символ в скобках.
[^…]	Совпадает любой отдельный символ не в скобках.
\A	Начало всей строки
\z	Конец всей строки
\Z	Конец всей строки, за исключением допустимого конечного конца строки.
re*	Совпадает 0 или более вхождений предыдущего выражения.
re+	Совпадает 1 или более из предыдущих вещей.
re?	Совпадает 0 или 1 вхождений предыдущего выражения.
re{ n}	Совпадает n вхождений предыдущего выражения.
re{ n,}	Совпадает n или более вхождений предыдущего выражения.
re{ n, m}	Совпадает от n до m вхождений предыдущего выражения.
a| b	Совпадает a или b.
(re)	Группирует регулярные выражения и запоминает совпадающий текст.
(?: re)	Группирует регулярные выражения без запоминания совпадающего текста.

(?> re)	Совпадает независимый шаблон без возвратов.
\w	Совпадает символ в слове.
\W	Совпадает символ не в слове.
\s	Совпадают пробелы.
\S	Совпадают не пробелы.
\d	Совпадают числа.
\D	Совпадают не числа.
\G	Соответствует точке, где заканчивается предыдущее совпадение.
\n	Обратная ссылка для захвата числа n.
\b	Совпадают границы слова за пределами скобок.
\B	Совпадают границы не слова.
\n, \t, etc.	Совпадают соответствующие символы (новая строка, возврат каретки и т.д.)
\Q	Цитаты, до символа \E
\E	Конец цитаты \Q

Под «нулевой шириной» мы подразумеваем, что при написании эти символы не отображаются. Они есть, но мы не можем их видеть. Эти две позиции в строке называются якорями, поскольку они позволяют нам привязать шаблон регулярного выражения к конкретной точке в строке.

2. Matcher этот класс интерпретирует шаблон и определяет совпадения в водимой строке.
   Класс Pattern предоставляет нам скомпилированный вариант регулярного выражения.
   PatternSyntaxException этот класс предоставляет нам непроверяемые исключения, которые указывают нам на синтаксическую ошибку в нашем регулярном выражении.
   В пакете java.util.regex

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class CapturingGroupDemo {
    public static void main(String[] args) {
        String inputString = "This is simple that contains phone number +380505055050 That's great.";
        String pattern = "(\\d+)";

        Pattern ptrn = Pattern.compile(pattern);
        Matcher matcher = ptrn.matcher(inputString);

        if(matcher.find()){
            System.out.println("Phone number: " + matcher.group(0));
        }else {
            System.out.println("PHONE NUMBER NOT FOUND");
        }
    }
}
```

```
/*Some system messages*/
Phone number: 380505055050
```

   Как мы видим, мы применяем регулярное выражение \d, которое выводит числа (от 0 до 9). В результате программа опускает все нечисловые символы в строке и возвращает нам только номер телефона.
3. Группа - это пронумерованная часть регулярного выражения.

присутствуют две группы. Группа 0 всегда относится ко всему выражению, а группа 1 - к подвыражению, начинающемуся с открывающей круглой скобки "(" и заканчивающемуся закрывающей круглой скобкой ")". Текст соответствующих групп сохраняется обнаружителем соответствий регулярного выражения и может быть извлечен в дальнейшем в регулярном выражении.

```
import java.util.regex.*;

public class GroupDemo1 {
  static final String stringlist[] = { "abc 123 def", "456 ghi", "jkl789mno" };

  public static void main(String args[]) {

    // компилировать шаблон регулярного выражения для
    // числа, состоящего из одного или более цифр

    Pattern patt = Pattern.compile("(\\d+)");

    for (int i = 0; i < stringlist.length; i++) {
      String currstr = stringlist[i];

      // определить, содержит ли соответствие текущая строка

      Matcher match = patt.matcher(currstr);

      // если соответствие найдено - отобразить текст строки
      // для подходящей группы (группа 1)

      if (match.find()) {
        System.out.println("For \"" + currstr + "\" match is: "
            + match.group(1));
      }
    }
  }
}
```

```
For "abc 123 def" match is: 123
For "456 ghi" match is: 456
For "jkl789mno" match is: 789
```