Экранирование символов

Статья из группы Java Developer

21958 участников

Привет!

В прошлых лекциях мы уже успели познакомиться со строками, которые в Java представлены классом String.

Как ты, наверное, помнишь, строка – это последовательность символов.

Символы могут быть любыми – буквы, цифры, знаки препинания и так далее. Главное, чтобы при создании строки вся последовательность заключалась в кавычки:

```
public class Main {

   public static void main(String[] args) {

       String sasha = new String ("Меня зовут Саша, мне 20 лет!");

   }

}
```

Но что будет, если нам нужно создать строку, внутри которой тоже есть

кавычки? Например, мы хотим рассказать миру о своей любимой книге:

```
public class Main {

   public static void main(String[] args) {

       String myFavoriteBook = new String ("Моя любимая книга - "Сумерки" Стефани Майер");

   }

}
```

Кажется, компилятор чем-то недоволен!

Как думаешь, в чем может быть причина ошибки, и почему она возникла именно с кавычками?

Дело в том, что компилятор воспринимает кавычки строго определенным образом, а именно – оборачивает в них строку. И каждый раз, когда он видит символ ", он ожидает, что для него далее будет следовать такой же символ, а между ними будет находиться текст строки, которую он, компилятор, должен создать. В нашем же случае кавычки вокруг слова "Сумерки" находятся внутри других кавычек. И когда компилятор доходит до этого

куска текста, он просто не понимает, что от него хотят. Вроде как стоит кавычка, а значит, он должен создать строку. Но ведь он уже это делает!

Именно в этом кроется причина. Говоря по-простому, в этом месте компилятор неправильно понимает, что от него хотят.

"Еще одна кавычка? Это какая-то ошибка? Я ведь уже создаю строку! Или я должен создать еще одну? Эээээ...:/"

Нам нужно объяснить компилятору, когда кавычка является для него командой ("создай строку!"), а когда она является простым символом ("выведи на экран слово "Сумерки" вместе с кавычками!").

Для этого в Java используется экранирование символов.

Оно осуществляется с помощью специального символа. Вот такого: \. В обычной жизни он называется "обратный

слэш", но в Java он (в сочетании с символом, который нужно экранировать) называется управляющей последовательностью.

Например, \" – вот она – управляющая последовательность для вывода на экран кавычки.

Встретив такую конструкцию внутри твоего кода компилятор поймёт, что это просто символ "кавычка", который нужно вывести на экран.

Попробуем изменить наш код с книгой:

```
public static void main(String[] args) {

	String myFavoriteBook = new String ("Моя любимая книга - \"Сумерки\" Стефани Майер");

	System.out.println(myFavoriteBook);

   }

}
```

Мы экранировали две "внутренние" кавычки с помощью символа \.

Попробуем запустить метод main()...

Вывод в консоль:

Моя любимая книга - "Сумерки" Стефани Майер

Отлично, код отработал именно так, как было нужно!

Кавычки – далеко не единственный случай, когда нам может понадобиться экранирование символов. Например, мы захотели рассказать кому-то о своей работе:

```
public class Main {

   public static void main(String[] args) {

       String workFiles= new String ("Мои рабочие файлы лежат в папке D:\Work Projects\java");

       System.out.println(workFiles);

   }

}
```

И снова ошибка! Уже догадываешься, в чем причина?

Компилятор снова не понимает, что ему делать. Ведь символ \ для него – ни что иное, как управляющая последовательность! Он ожидает, что после слэша должен следовать какой-то символ, который он должен будет как-то

по-особому интерпретировать (например, кавычка).

Однако здесь после \ следуют обычные буквы. Поэтому компилятор снова сбит с толку.

Что делать? Ровно то же самое, что и в прошлый раз: всего лишь добавить к нашему \ еще один \!

```
public class Main {


   public static void main(String[] args) {


       String workFiles= new String ("Мои рабочие файлы лежат в папке D:\\Work Projects\\java");
       System.out.println(workFiles);


   }
}
```

Посмотрим, что из этого выйдет:

Вывод в консоль:

```
Мои рабочие файлы лежат в папке D:\Work Projects\java
```

Супер! Компилятор моментально определил, что \ – обычные символы, которые нужно вывести в консоль вместе с остальными.

В Java существует достаточно много управляющих последовательностей. Вот их полный перечень:

- \t символ табуляции.
- \b символ возврата в тексте на один шаг назад или удаление одного символа в строке (backspace).
- \n символ перехода на новую строку.
- \r символ возврата каретки. ()
- \f прогон страницы.
- \' символ одинарной кавычки.
- \" символ двойной кавычки.
- \\ символ обратной косой черты (\).

Таким образом, если компилятор встретит в тексте символ \n, он поймёт, что это не просто символ и буква, которые нужно вывести в консоль, а специальная команда для него – "сделай перенос строки!".

Например, это может нам пригодиться, если мы хотим вывести в консоль кусок стихотворения:

```
public class Main {

   public static void main(String[] args) {

       String borodino = new String ("Скажи-ка, дядя, \nВедь не даром \nМосква, спаленная пожаром, \nФранцузу отдана?");

       System.out.println(borodino);

   }

}
```

И вот что мы получили:

Вывод в консоль:

Скажи-ка, дядя,

Ведь не даром

Москва, спаленная пожаром,

Французу отдана?

То, что нужно! Компилятор распознал управляющую последовательность и вывел кусочек стиха в 4 строки.

Юникод

Еще одна важная тема, о которой тебе нужно знать в связи с экранированием символов – Unicode(Юникод).

Юникод – это стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира.

Иными словами, это список специальных кодов, в котором найдется код почти для любого символа из любого языка! Естественно, что список этот очень большой и наизусть его никто не учит :)

Если тебе интересно, откуда он появился и зачем стал нужен, почитай познавательную статью на Хабрахабре.

Все коды символов в Юникоде имеют вид “буква u + шестнадцатеричная цифра”.

Например, знаменитый знак копирайта обозначается кодом u00A9.

Так вот, если при работе с текстом в Java тебе понадобится его использовать этот знак, ты можешь экранировать его в своем тексте!

Например, мы хотим сообщить всем, что авторские права на эту лекцию принадлежат JavaRush:

```
public class Main {
```

```
   public static void main(String[] args) {

       System.out.println("Лекция \"Экранирование символов\", \u00A9 2018 Javarush");

   }

}
```

Вывод в консоль:

```
Лекция "Экранирование символов", © 2018 Javarush
```

Отлично, все получилось!

Но специальные символы – это еще не все!

С помощью Юникода и экранирования символов ты можешь закодировать текст, написанный одновременно на разных языках. И даже на нескольких разных диалектах одного языка!

```
public class Main {

   public static void main(String[] args) {


       System.out.println("\u041c\u0430\u0301\u043e " +

               "\u0426\u0437\u044d\u0434\u0443\u0301\u043d " +

               "\u0028\u043a\u0438\u0442\u002e \u0442\u0440\u0430\u0434\u002e " +
```

```
                "\u6bdb\u6fa4\u6771\u002c
\u0443\u043f\u0440\u002e " +

                "\u6bdb\u6cfd\u4e1c\u002c
\u043f\u0438\u043d\u044c\u0438\u043d\u044c\u003a " +

                "\u004d\u00e1\u006f
\u005a\u00e9\u0064\u014d\u006e\u0067\u0029 " +

                "\u2014
\u043a\u0438\u0442\u0430\u0439\u0441\u043a\u0438\u0439 " +


"\u0433\u043e\u0441\u0443\u0434\u0430\u0440\u0441\u0442\u0432\u0
435\u043d\u043d\u044b\u0439 " +

                "\u0438
\u043f\u043e\u043b\u0438\u0442\u0438\u0447\u0435\u0441\u043a\u04
38\u0439 " +

                "\u0434\u0435\u044f\u0442\u0435\u043b\u044c
\u0058\u0058 \u0432\u0435\u043a\u0430\u002c " +

                "\u0433\u043b\u0430\u0432\u043d\u044b\u0439
\u0442\u0435\u043e\u0440\u0435\u0442\u0438\u043a " +


"\u043c\u0430\u043e\u0438\u0437\u043c\u0430\u002e");

   }

}
```

Вывод в консоль:

Ма́о Цзэду́н (кит. трад. 毛澤東, упр. 毛泽东, пиньинь: Máo Zédōng) — китайский государственный и политический деятель XX века, главный теоретик маоизма.

В этом примере мы, зная коды символов, написали строку, состоящую из кириллицы, и трех (!) разных типов записи китайских иероглифов – классического, упрощенного и латинского (пиньинь).

Вот, в общем-то, и все! Теперь ты знаешь об экранировании символов вполне достаточно, чтобы использовать этот инструмент в работе:)