

2.1. Классы



IT ШКОЛА SAMSUNG

Первые программы не превышали нескольких сотен строк.

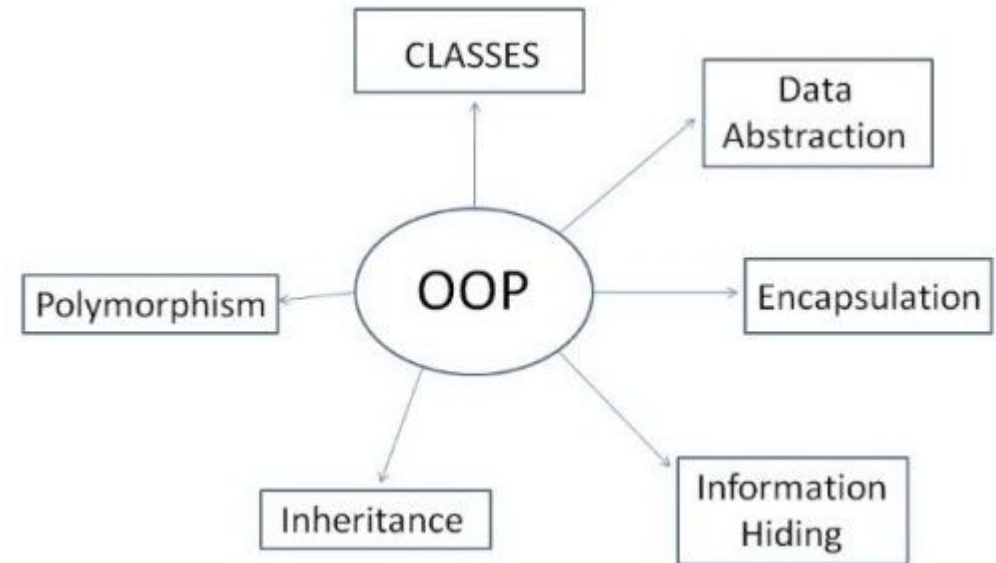
Увеличение программного кода до нескольких тысяч строк привело к внедрению приемов структурного программирования (появились функции), позволяющего создавать и сопровождать программы размером до сто тысяч строк.



Стремительное развитие программного обеспечения потребовало создания и сопровождения еще большего объема кода. Ответом на это было создание объектно-ориентированной технологии.

С помощью данной технологии возможно создавать большие по объему приложения.

Она позволяет программисту при создании кода мысленно отталкиваться не от архитектуры компьютера, а оперировать понятиями, обозначающими объекты реального мира.

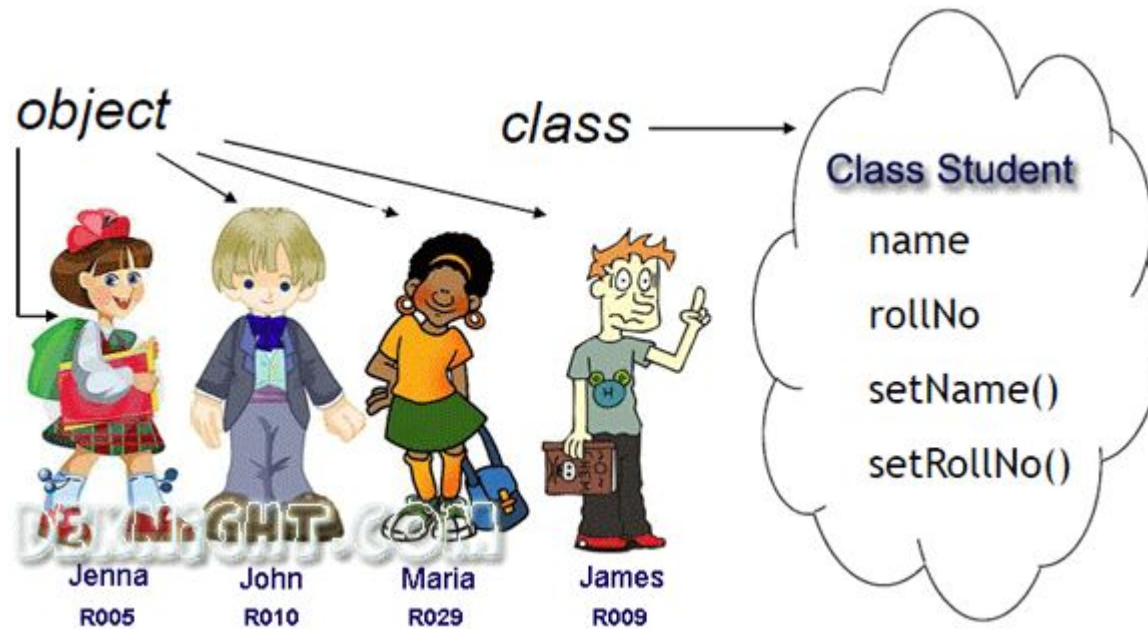


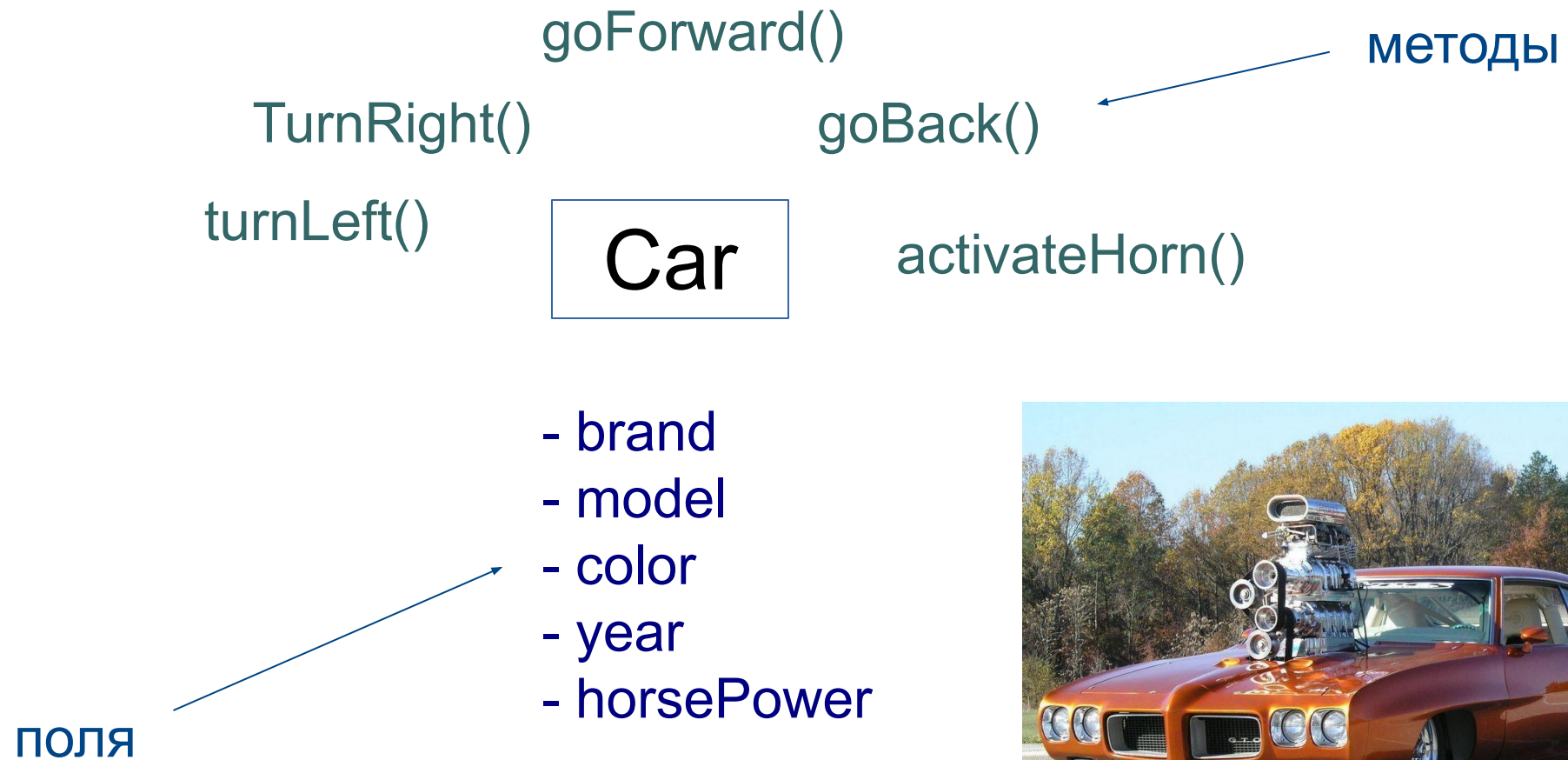
Класс — это абстрактный тип данных, описывающий реальную сущность и состоящий из полей и методов.

Поля класса - переменные для хранения данных, описывающих класс (те свойства/параметры/характеристики, которые **описывают состояние объекта**). Обычно это имена существительные.

Методы класса - функции для работы с полями класса. Это **действия, которые можно производить с этой сущностью** либо **действия, которые может совершать объект**. Обычно это глаголы.

С помощью класса описывается некоторая сущность (ее характеристики и возможные действия). Например, класс может описывать студента, автомобиль и т.д. Описав класс, мы можем создать его экземпляр – объект.
Объект – это уже конкретный представитель класса.



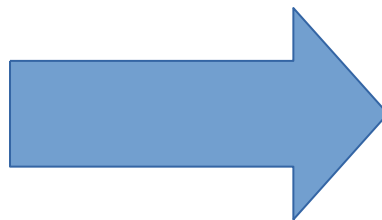




3 кита ООП:

- Инкапсуляция
- Наследовани
е
- Полиморфиз
м

Инкапсуляция - это сокрытие реализации класса и отделение его внутреннего представления от внешнего интерфейса.

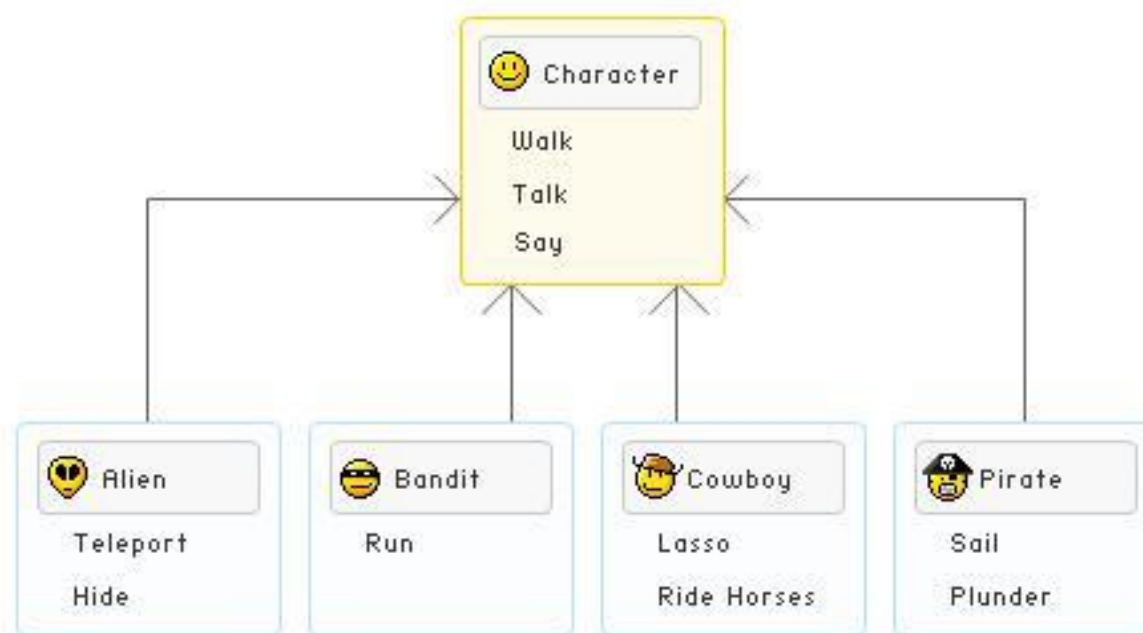


Для инкапсуляции применяются модификаторы доступа:

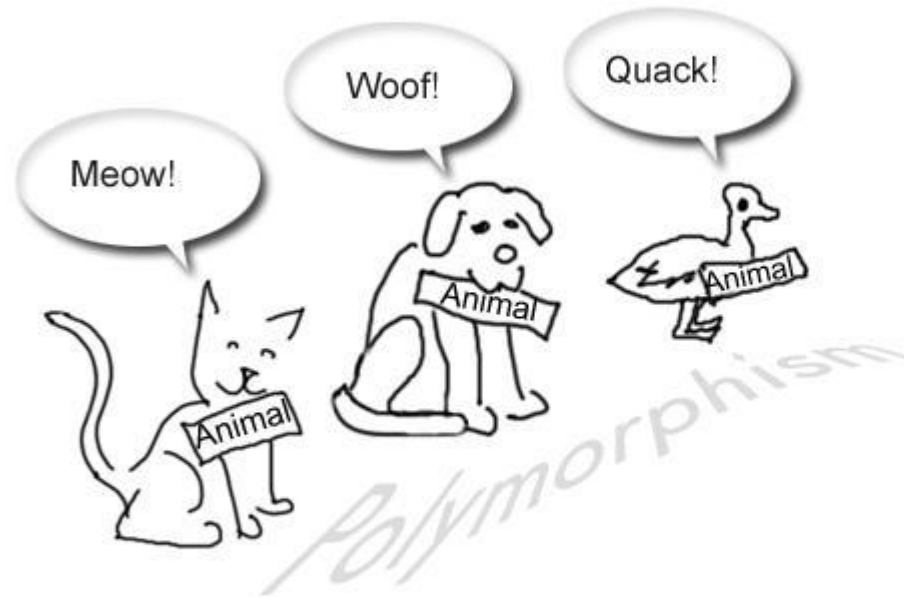
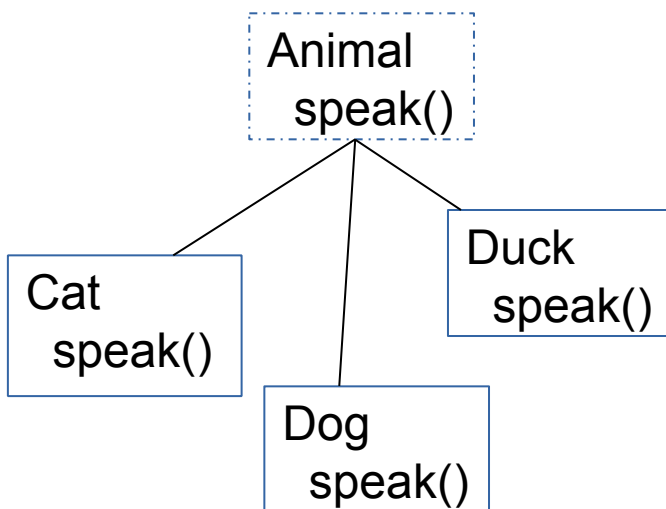
- public
- protected
- default
- private

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
no modifier	Y	Y	X	X
private	Y	X	X	X

Наследование - это отношение между классами, при котором класс использует структуру или поведение другого класса (одиночное *наследование*), или других (множественное *наследование*) классов.



Полиморфизм в ООП - это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта

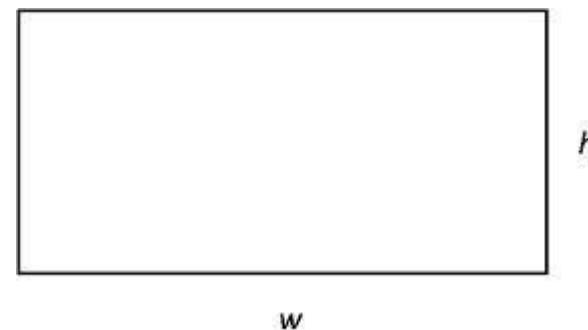


Общая форма оформления класса:

```
class Имя_класса {  
    тип_поля1 имя_поля1; // имя_переменной1  
    тип_поля2 имя_поля2; // имя_переменной2  
  
    тип_результата1 имя_метода1 (параметры_метода1) {  
        тело_метода1  
    }  
  
    тип_результата2 имя_метода2 (параметры_метода2) {  
        тело_метода2  
    }  
}
```

Класс Прямоугольник

```
class Rect {  
    double width;  
    double height;  
  
    double area() {  
        return width * height;  
    }  
}
```



Создание объекта

Для объявления объекта используется тот же синтаксис, как и для обычных переменных:

```
Имя_класса имя_объекта;
```

В нашем примере:

```
Rect rect; // объявили переменную класса Rect под именем rect
```

Мы только объявили переменную, т.е. сообщили компилятору, что в переменной `rect` мы собираемся хранить ссылку на объект - экземпляр класса `Rect`. Самого объекта в памяти еще нет.

Мы даже можем явно указать, что объектная переменная пока ни на что не ссылается, используя значение `null`:

```
rect = null;
```

Создание объекта

Чтобы создать сам объект, необходимо использовать операцию `new` (мы уже встречались с этой операцией, когда создавали массивы):

```
rect = new Rect();
```

Можно объединить описание и создание объекта:

```
Имя_класса имя_объекта = new Имя_класса();
```

В нашем случае:

```
Rect rect = new Rect();
```

Создание объекта

```
Rect rect = new Rect ();
```

Rect() - вызов метода класса без параметров и его имя совпадает с названием класса. Такой метод называется конструктором класса.

Конструктор определяет действия, выполняемые при создании объекта.

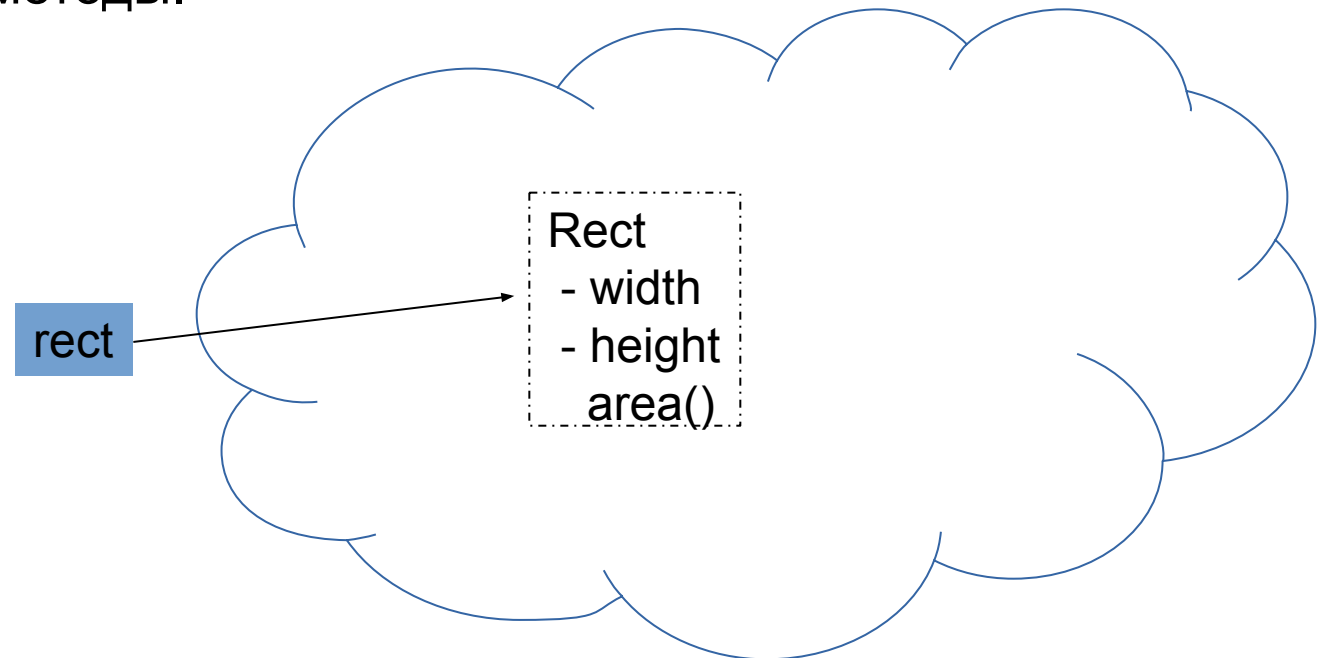
Если в классе программистом не был задан конструктор, то Java автоматически создаст "конструктор по умолчанию", который проинициализирует все поля создаваемого объекта значениями по умолчанию, принятыми для каждого типа Java.

Тип	Значение по умолчанию
byte, short, int, long, float, double	0
char	'\u0000'
boolean	false
объекты и массивы	null

Создание объекта

В переменных объектного типа хранятся ссылки на динамическую область памяти, где операцией `new` было выделено место под хранение объекта со всеми его полями и ссылками на методы.

```
Rect rect = new Rect();
```



Класс Прямоугольник

```
Rect rect1 = new Rect();  
rect1.height = 10;  
rect1.width = 20;
```

```
Rect rect2 = new Rect();  
rect2.height = 10;  
rect2.width = 20;
```

```
System.out.println(rect1 == rect2); // ?
```

```
rect2 = rect1;  
System.out.println(rect1 == rect2); // ?
```

false

true

Операция “==”, примененная к объектам, сравнивает не содержимое объектов, а ссылки на объекты в памяти.

rect1 и rect2 сначала хранят ссылки на разные ячейки памяти.

Далее, после операции rect2 = rect1; переменной rect2 мы присвоили адрес объекта, на который ссылается rect1. Таким образом, обе переменные стали ссылаться на один и тот же объект и во втором случае результат сравнения - true.

Вызов методов

Для вызова метода, описанного в классе, нужно обязательно указать объект, к которому применяется этот метод (исключение — статические методы, которые будут рассмотрены позже).

```
double area = rect1.area();
```

Вызов метода класса полезно рассматривать как посылку сообщения объекту. Указанный (перед точкой) объект рассматривается как адресат, которому посылают сообщение — приказ что-то выполнить. Если не указать конкретный объект — адресата нет и неясно, кому адресовать сообщение (метод).

Строки в Java

Для создания строки можно использовать следующие конструкции:

```
String имя_строки = "содержимое строки"; //рекомендуемая  
форма
```

```
String имя_строки = new String("содержимое строки");
```

Например:

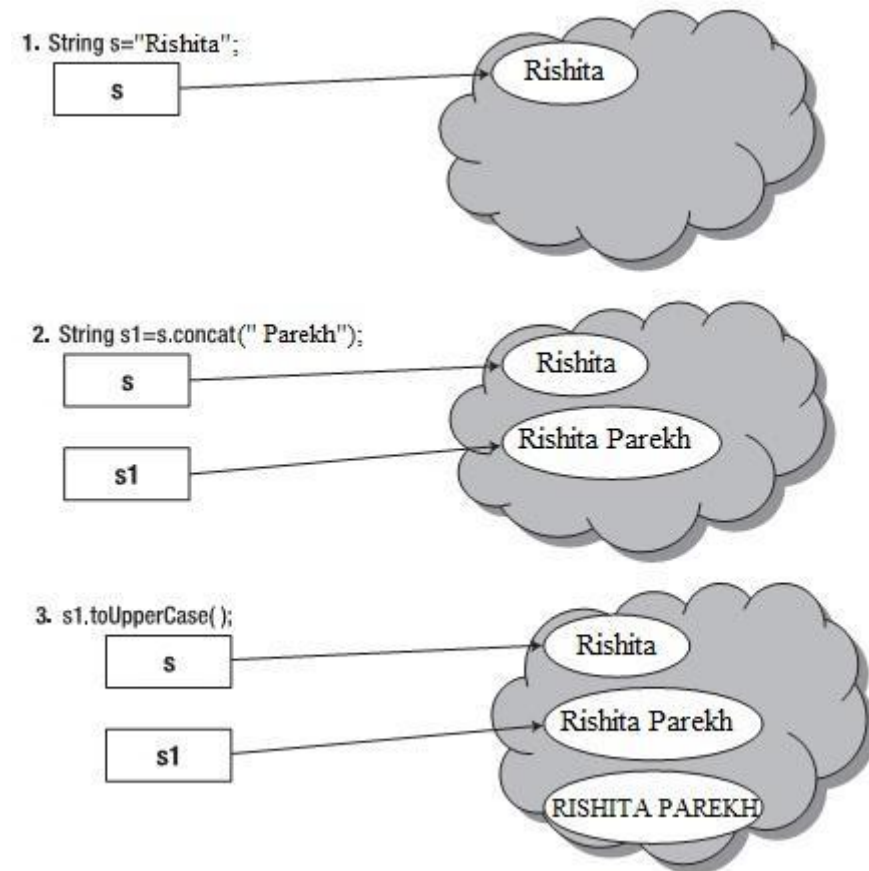
```
String str = "hello";
```

```
String str2 = new String("hello");
```

В результате выполнения этой строчки кода в памяти создается объект класса `String`, хранящий символы строки "hello", а ссылка на него сохраняется в переменной `str/str2`.

Строки в Java

В Java String относятся к неизменяемым объектам (англ. *immutable*) - объектам, состояние которых не может быть изменено после создания. При любой попытке изменения строки - в памяти создается новый объект.



Строки в Java

```
String s1 = new String("Ваня учится в IT ШКОЛЕ SAMSUNG");  
String s2 = new String("Ваня учится в IT ШКОЛЕ SAMSUNG");  
System.out.println(s1.equals(s2));  
System.out.println(s1 == s2);
```

На экран будет выведено:

```
true  
false
```

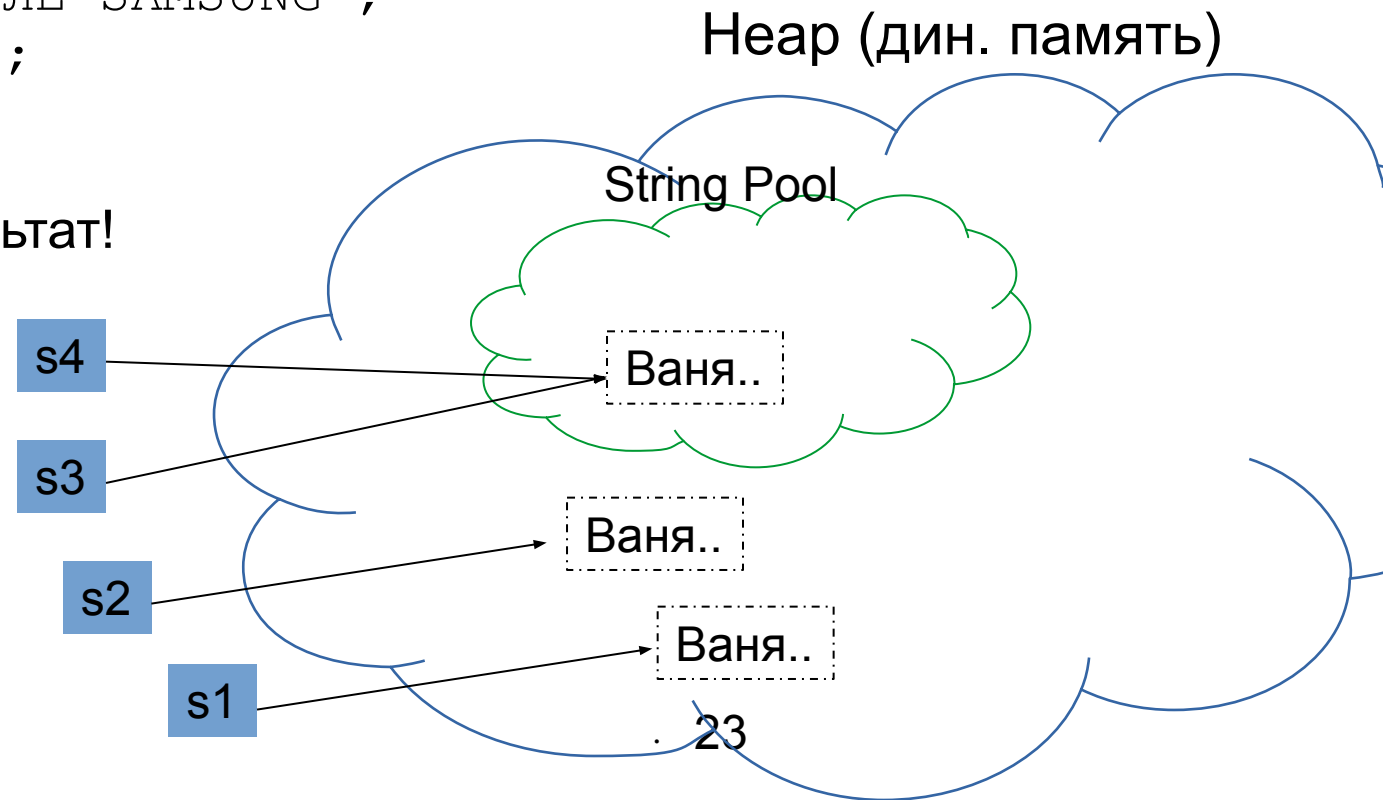
Метод `equals` позволяет сравнивать содержимое строк, что не позволяет сделать операция `==`.

Строки в Java

```
String s3 = "Ваня учится в IT ШКОЛЕ SAMSUNG";  
String s4 = "Ваня учится в IT ШКОЛЕ SAMSUNG";  
System.out.println(s3.equals(s4));  
System.out.println(s3 == s4);
```

Удивительно, но мы получим другой результат!

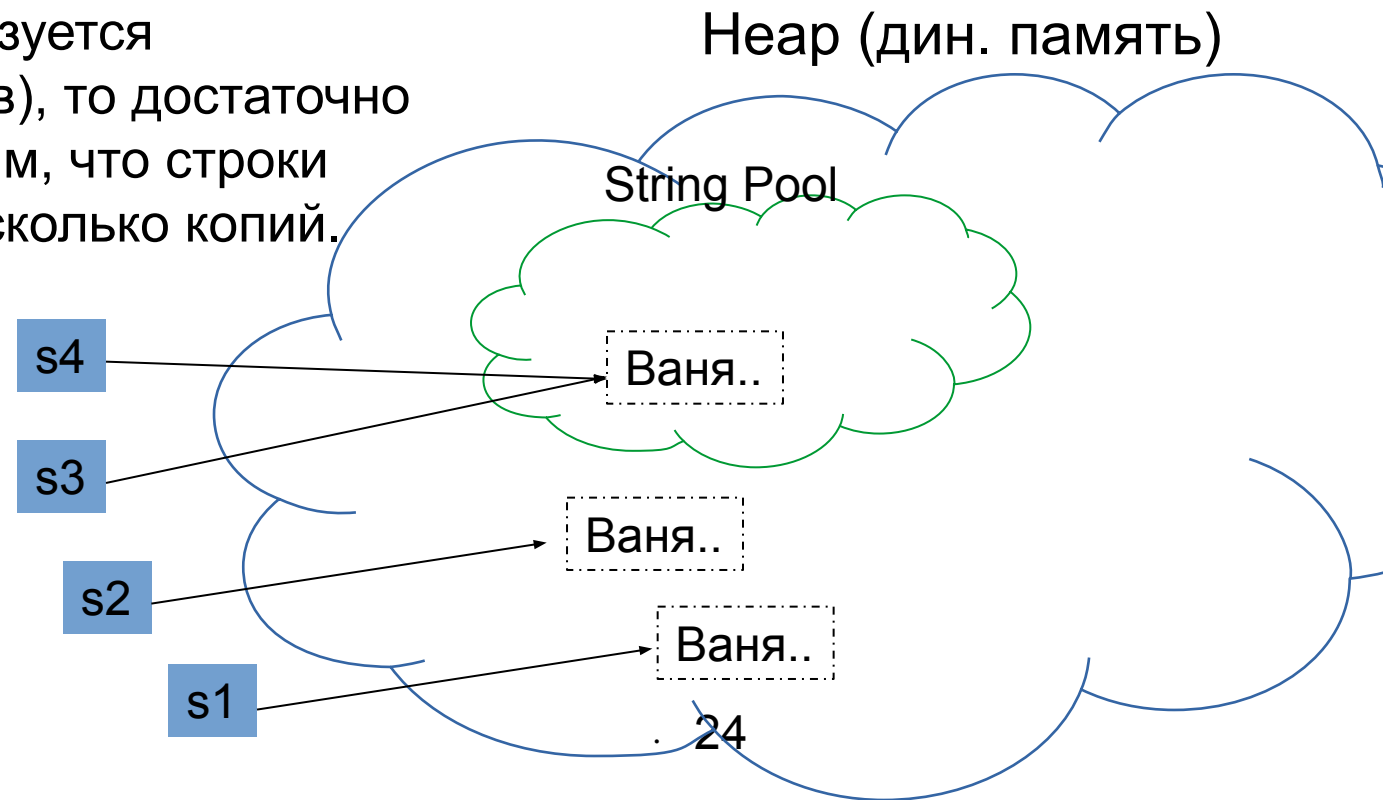
```
true  
true
```



Строки в Java

Тип String – один из самых часто используемых в Java и довольно дорогой для хранения, поэтому в памяти виртуальной машины было решено использовать подход Flyweight (приспособленец): если используется множество одинаковых данных (объектов), то достаточно создать только один экземпляр (напомним, что строки Java являются неизменяемыми), чем несколько копий.

Поэтому всякий раз, когда создаются строковые литералы, в Строковом пуле проверяется, существует ли такой же строковый литерал. Если он существует, то новый образец для этой строки в Строковом Пуле не создается и переменная получает ссылку на уже существующий строковый литерал.

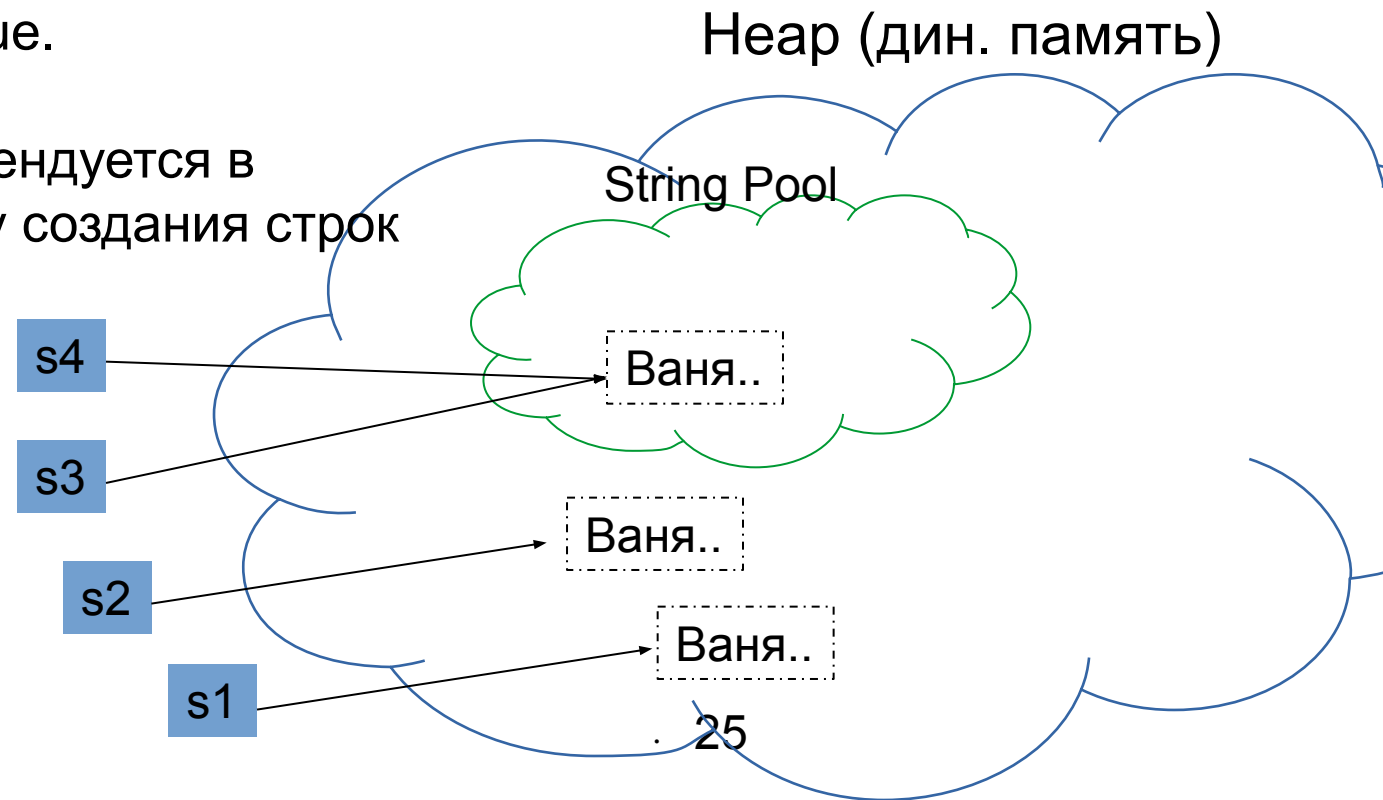


Строки в Java

В нашем примере переменным присвоены одинаковые строковые литералы и поэтому переменная `s2` получила ссылку на литерал, который был создан для `s1`. И в результате операция `s1 == s2` вернула `true`.

В связи со всем выше сказанным, рекомендуется в программах на Java использовать форму создания строк через строковые литералы.

```
String s = "Ваня..";
```



Упражнение

Спроектируйте и реализуйте простейший класс, описывающий рациональную дробь.



Классы-оболочки

Для того, чтобы с примитивными типами данных можно было работать так же, как с остальными-объектными, для каждого из них существует свой собственный класс-оболочка.

Они инкапсулируют в себе эти простые типы и предоставляют широкий набор методов для работы с ними.

Примитивный тип	Соответствующий класс-оболочка
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Классы-оболочки

Создание объектов соответствующих классов:

```
Integer i = Integer.valueOf(50);  
Double db = Double.valueOf(50.5);
```

Обратная операция:

```
int d = i.intValue(); //50
```

Преобразование в строку:

```
String s1 = i.toString(); //"50"  
String s2 = Double.toString(2.5); //"2.5"
```

Классы-оболочки

Переводит строковое представление числа в заданной системе счисления, в целое число:

```
int value = Integer.parseInt("110", 2);  
System.out.println(value); //выведет 6 (110 в 2-чной системе счисления)
```

Переводит число из 10-чной системы в 2-чную и возвращает в виде строки :

```
int x = 12;  
System.out.println(Integer.toBinaryString(x)); //1100
```

Аналогичны предыдущему методу, но переводят в 8-чную и 16-чную системы соответственно:

```
int x = 12;  
System.out.println(Integer.toOctalString(x)); // 14  
System.out.println(Integer.toHexString(x)); // c
```

Упражнение

Введите с клавиатуры целое число X ($|X| \leq 10^9$).

В первых трех строках выведите это число на экран в двоичной, восьмеричной, 16-ричной системах счисления.

В следующих двух строках выведите, поместится ли это число в ячейке типа `byte` и ячейке типа `short` ("YES"/"NO").

Запрещается использовать циклы и знания о том, сколько именно байт/бит памяти занимают переменные типа `int`, `byte`, `short`.

Пример ввода:
123

Пример вывода:
1111011
173
7B
YES
YES

Пример ввода:
40000

Пример вывода:
1001110001000000
116100
9C40
NO
NO