ТЕХНОПОЛИС

ООП. Классы, объекты

Сергей Товмасян

одноклассники

ПОЛИТЕХ

@mail.ru group

Память

1. Регистры

Это самое быстрое хранилище, потому что данные хранятся прямо внутри процессора. Однако количество регистров жестко ограничено, поэтому регистры используются компилятором по мере необходимости. У вас нет прямого доступа к регистрам, вы не сможете найти и малейших следов их поддержки в языке. (С другой стороны, языки C и C++ позволяют порекомендовать компилятору хранить данные в регистрах.)

Память

2. Стек

Стек. Эта область хранения данных находится в общей оперативной памяти (RAM), но процессор предоставляет прямой доступ к ней с использованием указателя стека. Указатель стека перемещается вниз для выделения памяти или вверх для ее освобождения. Это чрезвычайно быстрый и эффективный способ размещения данных, по скорости уступающий только регистрам. Во время обработки программы компилятор Java должен знать жизненный цикл данных, размещаемых в стеке. Это ограничение уменьшает гибкость ваших программ, поэтому, хотя некоторые данные Java хранятся в стеке (особенно ссылки на объекты), сами объекты Java не помещаются в стек.

Память

3. Куча

Пул памяти общего назначения (находится также в RAM), в котором размещаются все объекты Java. Преимущество кучи состоит в том, что компилятору не обязательно знать, как долго просуществуют находящиеся там объекты. Таким образом, работа с кучей дает значительное преимущество в гибкости. Когда вам нужно создать объект, вы пишете код с использованием new, и память выделяется из кучи во время выполнения программы. Конечно, за гибкость приходится расплачиваться: выделение памяти из кучи занимает больше времени, чем в стеке (даже если бы вы могли явно создавать объекты в стеке, как в C++).

Память

4. Постоянная память

Значения констант часто встраиваются прямо в код программы, так как они неизменны. Иногда такие данные могут размещаться в постоянной памяти (ROM), если речь идет о «встроенных» системах.

5. Неоперативная память

Неоперативная память. Если данные располагаются вне программы, они могут существовать и тогда, когда она не выполняется. Особенностью такого вида хранения данных является возможность перевода объектов в нечто, что может быть сохранено на другом носителе информации, а потом восстановлено в виде обычного объекта, хранящегося в оперативной памяти.

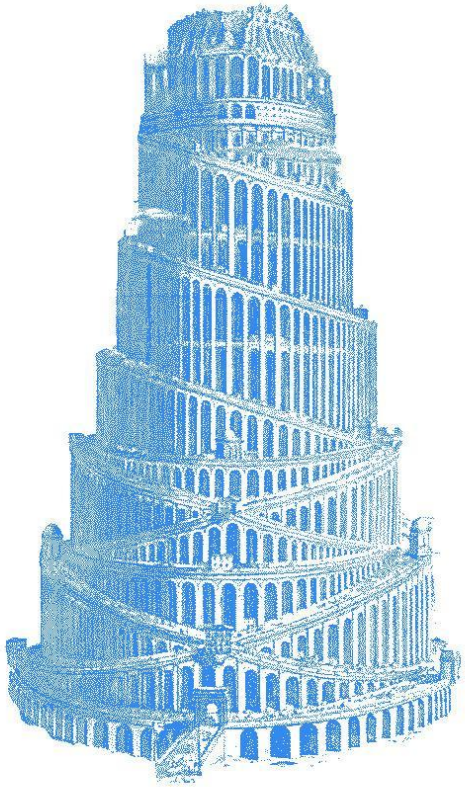
Примитивные типы

Одна из групп типов, часто применяемых при программировании, требует особого обращения. Причина для особого обращения состоит в том, что создание объекта недостаточно эффективно, так как new помещает объекты в кучу. В таких случаях Java следует примеру языков C и C++. То есть вместо создания переменной с помощью new создается «автоматическая» переменная, не являющаяся ссылкой. Переменная напрямую хранит значение и располагается в стеке, так что операции с ней гораздо производительнее.

Вопросы для самостоятельного изучения

32 vs 64

Дефрагментация памяти

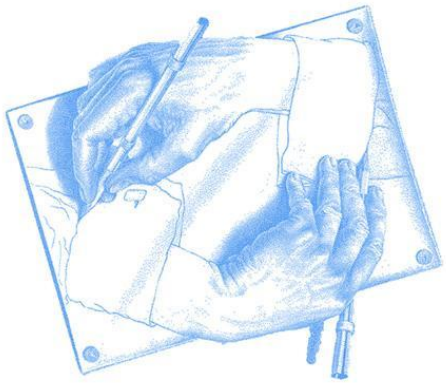


ООП

Методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию.

Основные понятия

1. **Абстракция данных**
2. **Инкапсуляция**
3. **Наследование**
4. **Полиморфизм**
5. **Класс**
6. **Объект**



Класс

Ключевое понятие ООП, под которое и заточена Java.

Примеры:

```
class Point {
   int x;
   int y;
}
```

```
class Box {
   int width;
   int height;
   int depth;
}
```

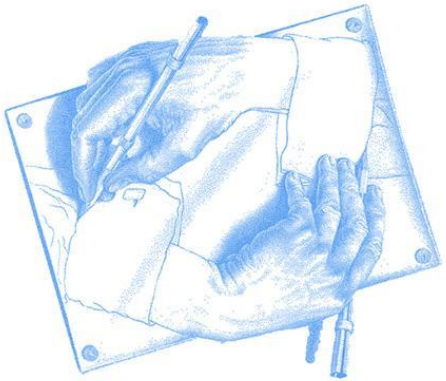
```
class Student {
   boolean male;
   int age;
   String name;
   String surname;
}
```

Представлять лучше всего как шаблон для создания объектов.

Конструкторы и создание объекта

```
class Point {
   int x;
   int y;

   Point() {
   }

   Point(int x) {
      this.x = x;
   }

   Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
}
```

```
Point point1 = new Point();
Point point2 = new Point(1);
Point point3 = new Point(3,5);
```

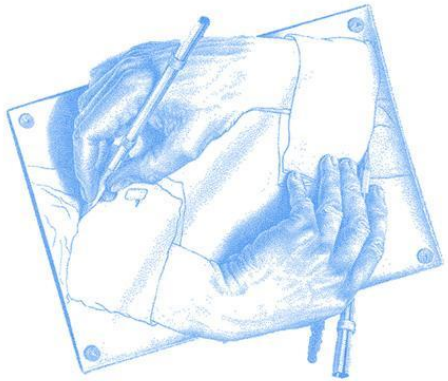


Базовый конструктор, - особенности

Наследование

```
class Box {
   int width;
   int height;
   int depth;
}
```

```
class HeavyBox extends Box {
   int weight;

   HeavyBox(int width, int height, int depth, int weight) {
      this.width = width;
      this.height = height;
      this.depth = depth;
      this.weight = weight;
   }
}
```



Добавим printVolume в HeavyBox.
Подумаем о плюсах и минусах наследования!!

“Всё является объектом” Брюс Эккель

В Java вы обращаетесь со всем, как с объектом, а идентификатор, которым Вы манипулируете представляет собой ссылку на объект.

Представьте себе телевизор (объект) с пультом дистанционного управления (ссылка). Во время владения этой ссылкой у вас имеется связь с телевизором, но при переключении канала или уменьшении громкости вы распоряжаетесь ссылкой, которая, в свою очередь, манипулирует объектом. А если вам захочется перейти в другое место комнаты, все еще управляя телевизором, вы берете с собой «ссылку», а не сам телевизор.

Класс java.lang.Object

String

Основные функции языка хранятся в пакете java.lang, который не надо импортировать, класс String – не исключение.

Важно! Строки – это константы. Их значения не могут быть изменены после создания. Выполнение того или иного метода над строками приводит к появлению нового объекта «строка», а не изменению старого.

Несколько способов создать строку:

```
String s1 = new String(“abc”);
//и так, но есть разница
String s2 = “abc”;
//ещё так
char[] arr = {‘a’ , ‘b’ , ‘c’};
String s1 = new String(arr);
```

compareTo(String anotherString) - лексиграфическое сравнение строк;
compareToIgnoreCase(String str) - лексиграфическое сравнение строк без учета регистра символов;
regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len) - тест на идентичность участков строк, можно указать учет регистра символов;
regionMatches(int toffset, String other, int ooffset, int len) - тест на идентичность участков строк;
concat(String str) - возвращает соединение двух строк;
contains(CharSequence s) - проверяет, входит ли указанная последовательность символов в строку;
endsWith(String suffix) - проверяет завершается ли строка указанным суффиксом;
startsWith(String prefix) - проверяет, начинается ли строка с указанного префикса;
startsWith(String prefix, int toffset) - проверяет, начинается ли строка в указанной позиции с указанного префикса;
equals(Object anObject) - проверяет идентична ли строка указанному объекту;
getBytes() - возвращает байтовое представление строки;
getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) - возвращает символьное представление участка строки;
hashCode() - хеш код строки;
indexOf(int ch) - поиск первого вхождения символа в строке;
indexOf(int ch, int fromIndex) - поиск первого вхождения символа в строке с указанной позиции;
indexOf(String str) - поиск первого вхождения указанной подстроки;
indexOf(String str, int fromIndex) - поиск первого вхождения указанной подстроки с указанной позиции;

lastIndexOf(int ch) - поиск последнего входения символа;
lastIndexOf(int ch, int fromIndex) - поиск последнего входения символа с указанной позиции;
lastIndexOf(String str) - поиск последнего вхождения строки;
lastIndexOf(String str, int fromIndex) - поиск последнего вхождения строки с указанной позиции;
replace(char oldChar, char newChar) - замена в строке одного символа на другой;
replace(CharSequence target, CharSequence replacement) - замена одной подстроки другой;
substring(int beginIndex, int endIndex) - возвратить подстроку как строку;
toLowerCase() - преобразовать строку в нижний регистр;
toLowerCase(Locale locale) - преобразовать строку в нижний регистр, используя указанную локализацию;
toUpperCase() - преобразовать строку в верхний регистр;
toUpperCase(Locale locale) - преобразовать строку в верхний регистр, используя указанную локализацию;
trim() - отсечь на концах строки пустые символы;
valueOf(a) - статические методы преобразования различных типов в строку.
matches(String regex) - удовлетворяет ли строка указанному регулярному выражению;
replaceAll(String regex, String rplc) - заменяет все вхождения строк, удовлетворяющих регулярному выражению, указанной строкой;
replaceFirst(String regex, String rplc) - заменяет первое вхождение строки, удовлетворяющей регулярному выражению, указанной строкой;
split(String regex) - разбивает строку на части, границами разбиения являются вхождения строк, удовлетворяющих регулярному выражению;

Object

Методы:

hashCode()
Есть заблуждение широкое, что Object.hashCode возвращает целочисленное представление адреса объекта в памяти. На самом деле это не так, а используется генератор случайных чисел. Генерится для объекта при первом вызове метода hashCode() и сохраняется в заголовке объекта.

```
public boolean equals(Object obj) {
   return (this == obj);
}
```

Разница между == и equals. == сравнивает ссылки, equals сравнивает значения. Но для Object, судя по коду, это одно и то же.

Integer, Double etc…

Integer, как класс обёртка

Разница между == и equals. == сравнивает ссылки, equals сравнивает значения.
Пример:

```
Integer a1 = new Integer(7);
Integer a2 = new Integer(7);
System.out.println(a1 == a2);
System.out.println(a1.equals(a2));
```

Boxing, unboxing

Вопрос, какой будет результат…

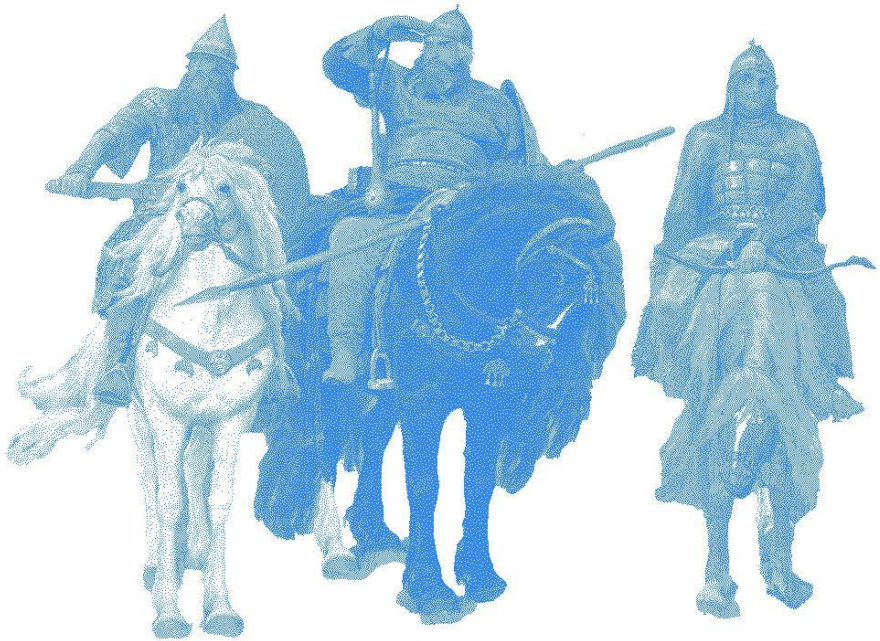
```
Integer a1 = 50;
Integer a2 = 50;
Integer a3 = 500;
Integer a4 = 500;

System.out.println(a1 == a2);
System.out.println(a3 == a4);
```

Таким образом, в большинстве случаев создаётся новый объект, а потому опасно

```
Integer a=0;
while(true) a++;
```

Integer, Double etc…

Изучаем полезные методы



Инкапсуляция

Это фундаментальная объектно-ориентированная концепция, позволяющая упаковывать данные и поведение в единый компонент с разделением его на обособленные части - интерфейс и реализацию. Последнее осуществляется благодаря принципу изоляции решений разработки в ПО, известному как сокрытие информации

- контроль доступа
- Контроль целостности/валидности данных
- Возможность изменения реализации

Инкапсуляция

Visibility	Public	Protected	*Default*	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, *through inheritance*	No	No
From any non-subclass class outside the package	Yes	No	No	No

Модификатор static

Мы уже знаем, что класс – это описание свойств и методов некоторого объекта. Объект – это экземпляр(инстанс) класса. Поля и методы объекта существуют только когда объект создан

static модификатор означает, что поле или метод принадлежит классу как таковому, а не конкретному объекту. Обращаться к такому полю/методу можно через имя класса

final и классы констант.

Плюсы и минусы static.

Абстрактный класс

Определяет каркас поведения.

Детали отданы дочерним классам на переопределение, а общее поведение вынесено в родительский абстрактный класс.

Создать экземпляр такого класса нельзя, так как его описание неполное.
Каждый конкретный дочерний класс должен дополнить описание.

```
abstract class GraphicObject {
   int x, y;

   void moveTo(int newX, int newY) {
   //code
   }
   abstract void draw();
   abstract void resize();
}

class Circle extends GraphicObject {
   void draw() {
   ...
   }
   void resize() {
   ...
   }
}
```

Немного про методы

Void, возвращаемые типы, модификаторы.
Overloading(compile time) – поиск подходящей сигнатуры в зависимости от списка параметров

Overriding(runtime) – поиск подходящей реализации (по реальному типу объекта)

```
public Integer sum(Integer a, Integer b) {
   return a+b;
}

public Integer sum(Float a, Integer b) {
   return a.intValue() + 2*b;
}
```

Overriding на примере equals и hashCode – генерация в Idea.

Интерфейсы

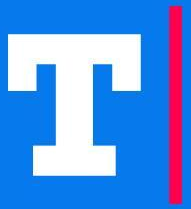
- Определяет, что можно сделать с классом
- Не определяет, как это сделать
- Класс может реализовывать несколько интерфейсов
- Чистая абстракция
- Позволяют строить гибкую архитектуру
- Интерфейс – это контракт

Маркерные интерфейсы

java.io.Serializable
java.rmi.Remote
java.lang.Cloneable

Полиморфизм

- Дочерний класс может быть использован везде, где используется родительский
- Если дочерний класс приведён к родительскому, то доступны только методы родительского класса
- Вызывается реализация из дочернего класса



Спасибо за внимание!