



ОСНОВЫ синтаксиса языка Java



Евгений Беркунский, НУК

eugeny.berkunsky@gmail.com

<http://www.berkut.mk.ua>



Зарезервированные слова

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* Не используются

** Добавлено в версии 1.2

*** Добавлено в версии 1.4

**** Добавлено в версии 5.0

добавлено в версии 9.0: _

ОСНОВЫ: программа DragonWorld

```
package heroes;
```

```
public class HelloDragonWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello DragonWorld!");  
    }  
}
```



Типы данных

Примитивные и ссылочные

Примитивные типы

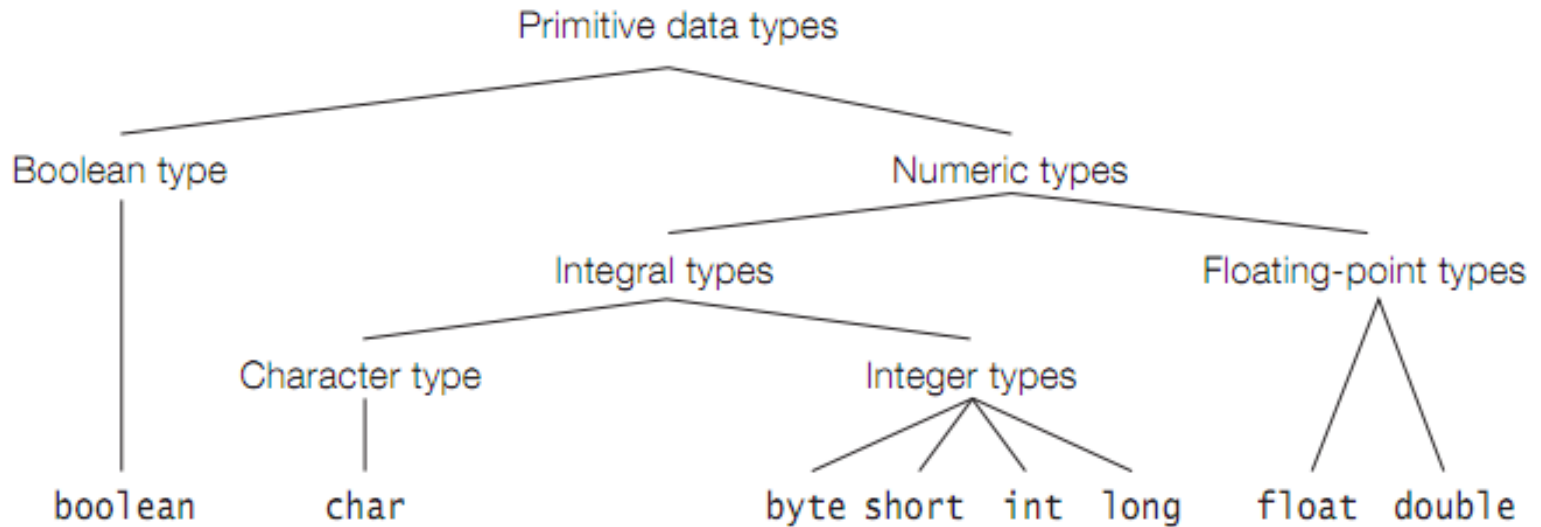
Тип	Размер	Диапазон	Пример
byte	1 байт	от -128 до 127	125
short	2 байта	от -32768 до 32767	-23
int	4 байта	от -2147483648 до 2147483647	2002300
long	8 байт	от -922372036854775808 до 922372036854775807	1243565L
float	4 байта	3.4e-038 ... 3.4e+ 038	1.2f
double	8 байт	1.7e-308 ... 1.7e+ 308	123.4
boolean		false, true	true
char	2 байта	Все символы стандарта Unicode	'z'

Ссылочные типы

Все остальные: строки, массивы, даты и т.п. (в том числе и описанные в программе)



Примитивные типы данных



Примитивные типы данных

```
static short bitCount(short s) {  
    short bits = 0;  
    while (s != 0) {  
        bits += s & 1;  
        s >>>= 1;  
    }  
    return bits;  
}
```

bitCount((short)-1)

1. 16
2. 32
3. 1
4. Ничего

1. Ошибка компиляции
2. Зациклится
3. ArithmeticException
4. StackOverflowError

Еще одна загадка

```
System.out.println("Красивый спиннер\n" +  
    "с логотипом Joker Conf\n"  
    + "Цена: всего " +  
    + '2' + " EUR");
```

1. Берём 100 штук, продадим участникам конфы по пять евро, выручку пропьём
2. Возьмём два - тебе и мне, 140 рублей с носа не такие большие деньги
3. Возьмём один на двоих, скинемся по 70, будем крутить по очереди
4. Вы что там с ума посходили с такими ценами???



Основы: пакет

Пакет – это совокупность классов и подпакетов, объединенных общим именем

```
package mydragons;  
public class Dragon{//реализация }
```

```
//использование  
import mydragons.Dragon;  
Dragon red = new Dragon(Dragon.RED);  
Dragon black = new mydragons.Dragon(Dragon.BLACK);
```



ОСНОВЫ: класс

Класс – это базовая сущность ООП, обладающая определенными свойствами

Любая программа на языке Java представляет собой класс

```
package animals.slowanimals;  
  
public class Reptile {  
    public void eat(Bird b){  
        b.wasEaten = true;  
    }  
}
```

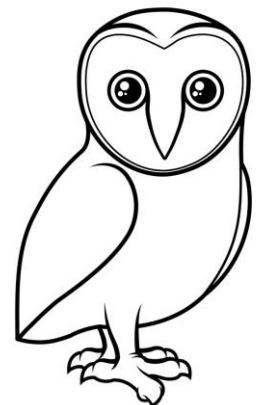


ОСНОВЫ: поле

Поле – это именованное свойство класса или объекта
Поле может относиться как к каждому объекту, так и к классу
в целом



```
package animals.slowanimals;  
  
public class Reptile {  
    private int length; // поле  
  
    public void eat(Bird b){  
        b.wasEaten = true;  
    }  
}
```



ОСНОВЫ: ОБЪЕКТ

Объект – это переменная, типом которой является соответствующий класс

Объект также называют экземпляром класса

```
package animals.slowanimals;  
//класс:  
public class Reptile {  
    private int length;  
    ...  
}  
...  
//объект:  
Reptile gecko = new Reptile();
```



ОСНОВЫ: МЕТОД

Метод – это программная функция, относящаяся к определенному объекту или классу
Области, откуда метод может быть доступен, определяются модификаторами метода

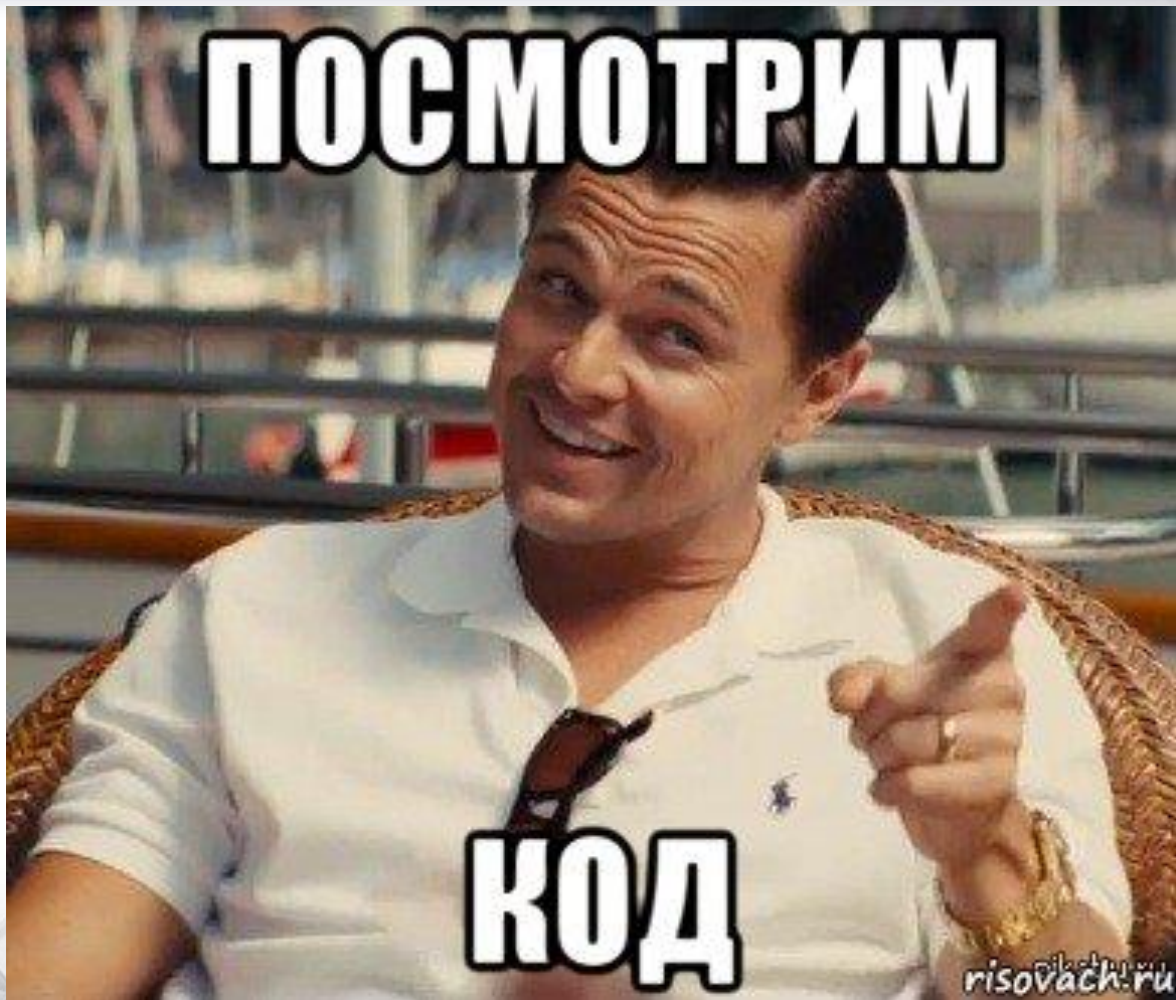
```
package animals.slowanimals;
```

```
public class Reptile {  
    private int length;  
    public void eat(Bird b){  
        b.wasEaten = true;  
        length++;  
    }  
}
```





Демонстрація



ОСНОВЫ: НАСЛЕДОВАНИЕ

Класс может заимствовать методы другого класса.
Язык Java поддерживает операцию наследования:

```
// наследование производится с помощью  
// ключевого слова extends  
public class Dragon extends Reptile {  
    //внутреннее поле класса  
    private String magic = "fire";  
    public String getMagic(){  
        //возврат результата  
        return magic; // fire  
    }  
}
```



Модификаторы

- Модификаторы доступа являются реализацией принципа инкапсуляции в языке Java
- Изменяя модификаторы, можно контролировать область видимости:
 - полей
 - методов
 - классов



Модификаторы полей (1/2)

Модификатор	Область видимости
private	Поле доступно только в данном классе или его объектах
Отсутствие модификатора	Поле доступно в классах и объектах того же пакета, где описан данный класс
protected	Поле доступно в классах и объектах того же пакета, где описан данный класс и в наследниках данного класса
public	Поле доступно отовсюду



Модификаторы полей (2/2)

Модификатор	Комментарий
<code>volatile</code>	Значение этого поля будет обновляться каждый раз при обращении к нему. Обычно используется при параллельном исполнении программы.
<code>static</code>	Поле принадлежит структуре класса. Одно значение присуще всем экземплярам.
<code>final</code>	Поле не может быть изменено
<code>transient</code>	Поле не участвует в процессе сериализации (сохранение состояния объекта во внешнюю память) по умолчанию.



Модификатор	Область видимости
private	Метод доступен только в данном классе или его объектах
Отсутствие модификатора	Метод доступен в классах и объектах того же пакета, где описан данный класс
protected	Метод доступен в классах и объектах того же пакета, где описан данный класс и в наследниках данного класса
public	Метод доступен отовсюду



Модификатор	Комментарий
final	Метод не может быть переопределен в наследнике
static	Метод принадлежит классу
abstract	Метод не имеет реализации
synchronized	Запрещено одновременное выполнение метода на разных потоках
native	Метод имеет реализацию на языке C или C++



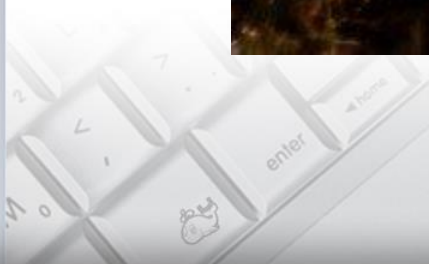
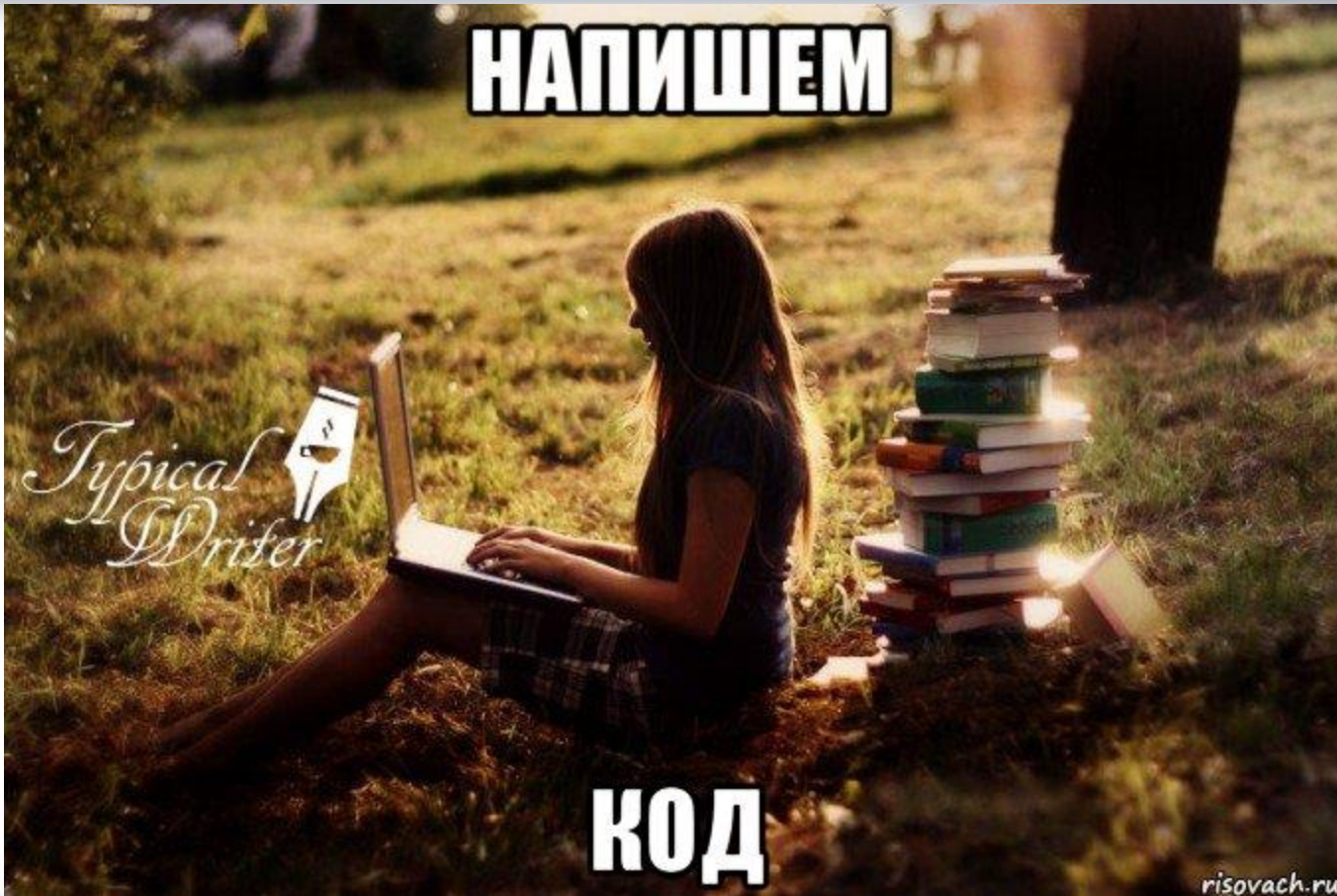
Модификаторы классов

Модификатор	Комментарий
отсутствие модификатора	Класс доступен только в текущем пакете
public	Класс доступен из любого пакета (публичный API)
final	Класс не может расширяться с помощью наследования
abstract	Класс является абстрактным, нельзя создать объект этого класса
static	Допустимо только для вложенных классов. Внутренний класс является статическим членом внешнего класса



НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ
КОРАБЛЕБУДУВАННЯ
ІМЕНІ АДМІРАЛА МАКАРОВА

Демонстрація



Конструктор

Конструктор – это метод, создающий экземпляр класса

- Не имеет заданного возвращаемого значения
- Имеет то же имя, что и класс

```
public class Dragon{  
    private String color = gold;  
    public Dragon(String newColor){  
        color = newColor;  
    }  
}
```

```
::  
{
```



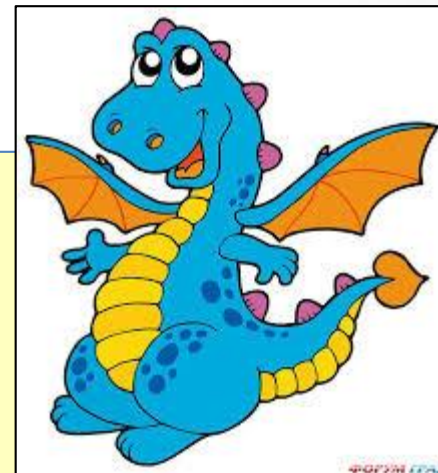
Конструктор по умовчанию

В классе всегда присутствует конструктор по умолчанию, если явно конструктор не задан

```
public class Dragon {  
    private String color;  
    public String getColor() {  
        return color;  
    }  
}
```

```
::  
}
```

```
//вызывается конструктор по умолчанию  
Dragon dragon = new Dragon();
```

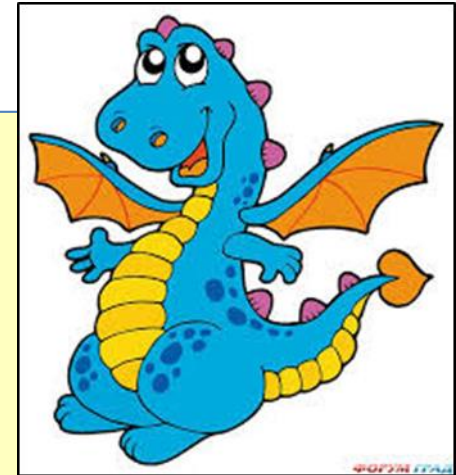


Вызов метода

Вызов метода – это обращение к члену класса по его имени
Результат вызова метода – выполненные операторы и возвращаемое значение (если указано)

```
public class Dragon{  
    private String name = "Kesha";  
    public String getName(){  
        return "I am " + name;  
    }  
}
```

```
Dragon dragon = new Dragon(); //конструктор  
System.out.println(dragon.getName()); //метод
```



Возвращаемое значение

Метод может возвращать одно значение (может быть простой тип, ссылочный тип, массив) в точку вызова

Если метод не возвращает никакого значения, то его возвращаемый тип – **void**

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

```
public void dumpValue() {  
    System.out.println("value = " + value);  
}
```

Виртуальный метод

Виртуальным называется метод, который замещает собой соответствующий метод предка, если метод вызывается для потомка

- Метод класса может быть переопределен в наследнике
- Конкретная реализация метода для вызова будет определяться во время исполнения
- Процесс по определению того метода, который следует вызывать называется диспатчем (dispatch)

Виртуальность – связывание класса с его методами на этапе создания объекта



Демонстрація



Повторное использование имен (переопределение)

Методы предка и наследника могут быть одноименными

```
class Reptile{  
    public void move() { /*ползти*/ }  
}
```

```
class Dragon extends Reptile{  
    public void move() { /*лететь*/ }  
}
```

```
Dragon d = new Dragon();  
d.move(); //обращение к методу экземпляра Dragon
```

```
Reptile r = new Reptile();  
r.move(); //обращение к методу экземпляра Reptile
```



Повторное использование имен (сокрытие)

Статические методы принадлежат классу

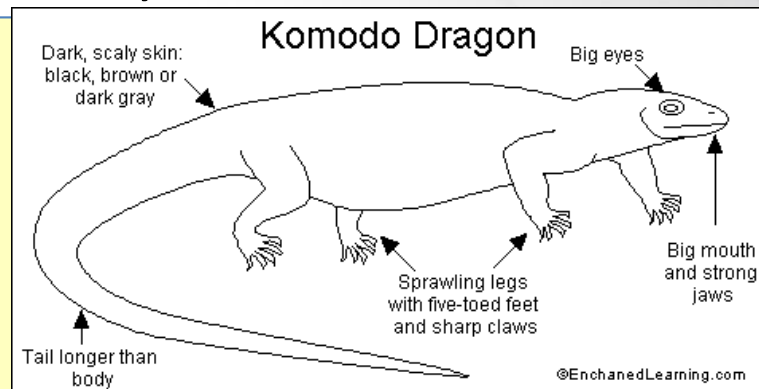
```
class Reptile{  
    public static void move() {}  
}  
class Dragon extends Reptile{  
    public static void move() {}  
}
```

```
Dragon d = new Dragon();  
Reptile r = new Reptile();  
d.move(); //обращение к методу Dragon  
r.move(); //обращение к методу Reptile
```

//Рекомендуется использовать вызовы класса:

```
Reptile.move();  
Dragon.move();
```

```
Reptile r1 = new Dragon(); //как работает r1.move();?
```



Повторное использование имен (перегрузка)

Методы выполняют схожую функцию над разными типами данных

```
class HungryDragon {  
    public void eat(int foodWeight){...}  
    public void eat(String foodWeight){  
        //разбор строки на значимое целое число  
        int i = Integer.parseInt(foodWeight);  
        ... //что бы вы поставили сюда?  
    }  
}  
  
...  
HungryDragon hd = new HungryDragon();  
hd.eat(10);  
hd.eat("10");
```

Повторное использование имен (затенение)

Локальная переменная делает одноименную глобальную переменную невидимой в локальной области

Так делать не рекомендуется

```
public class Dragon {  
    static String type = "Just Dragon";  
    public static void main(String[] s){  
        String type = "Black Dragon";  
        // выведет "Black Dragon"  
        System.out.println(type);  
    }  
}
```



Повторное использование имен (перекрытие)

Использование имен существующих методов и полей вносит в программу путаницу

Использование существующих имен классов недопустимо

```
public class BadExample {  
    static String System;  
    public static void main(String [] s){  
        System.out.println("A string");  
    }  
}
```

Результат работы программы:

```
BadExample.java:4: cannot resolve symbol symbol : variable out  
location: class java.lang.String  
    System.out.println("A string");
```


Передача параметров

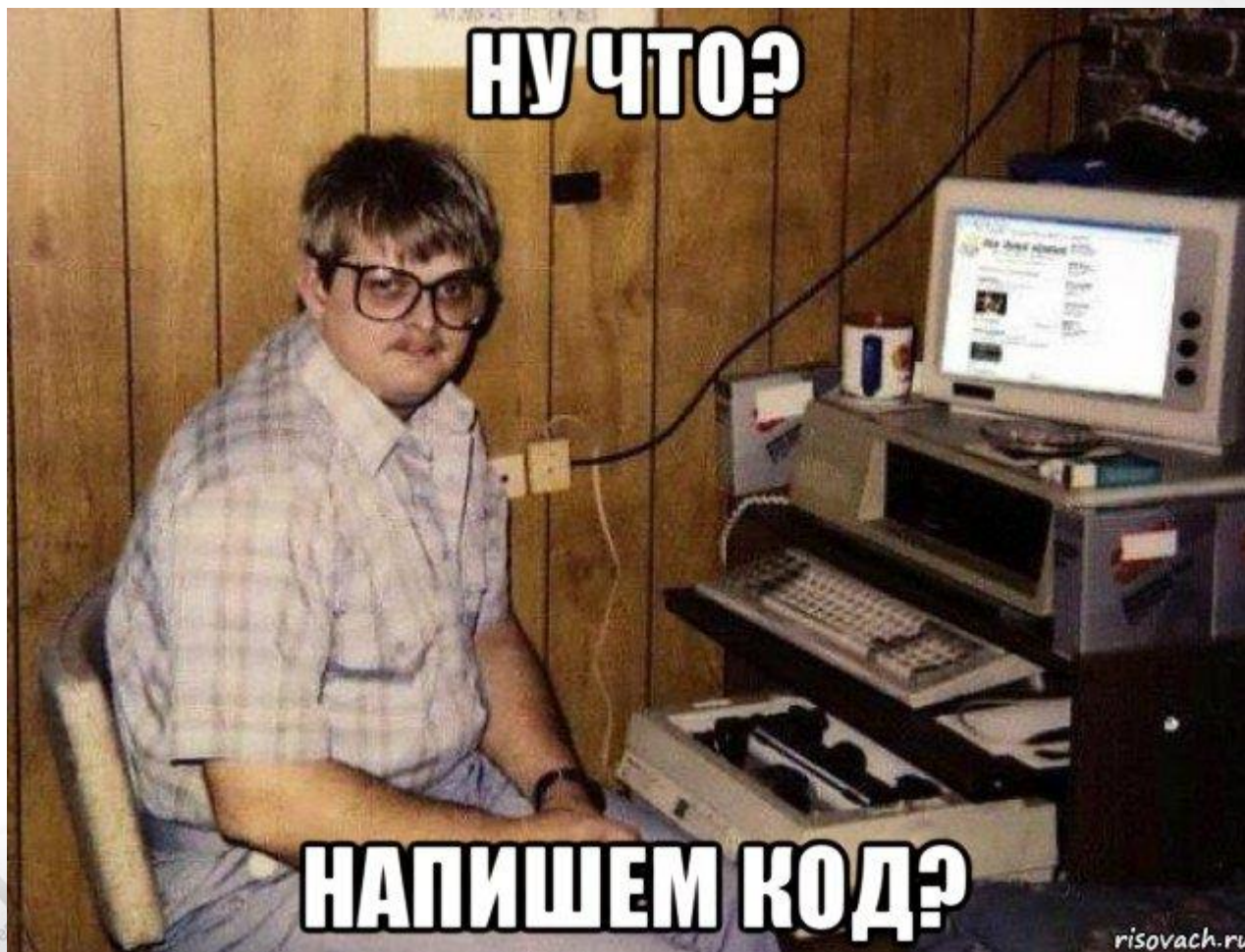
- Метод класу може отримувати до 255 параметрів
- Фактичний параметр вважається локальною змінною методу

```
class Dragon {  
    public void eat(Object obj){}  
    public void fly(String direction){}  
}
```

```
Dragon d = new Dragon();  
d.eat(new Girl());  
d.fly(Direction.WEST);
```



Демонстрація



Поле this

- Каждый объект имеет ссылку на самого себя
- Может использоваться для формирования ссылки на перегруженный конструктор и на поля объекта

```
class Dragon {  
    private int weight;  
    public Dragon(int weight){  
        //затенение  
        this.weight = weight;  
    }  
}
```



Поле super (1/2)

- Каждый объект имеет ссылку на объект-предок
- Позволяет организовать восходящие вызовы конструкторов

```
class NamedDragon extends Dragon {  
    private String name;  
    public NamedDragon(int weight, int name){  
//обращение к конструктору класса Dragon,  
// который умеет инициализировать объект  
// его весом  
        super(weight);  
        this.name = name;  
    }  
}
```

Поле super (2/2)

- С помощью данного поля можно вызывать методы предка

```
class Dragon extends Reptile{
    int flyingSpeed;
    public void attack(Object obj){
        //обращение к предку за выполнением
        //базовых действий
        super.attack(obj);
        burn(obj); //метод класса Dragon
    }
    // Dragon умеет атаковать, как Reptile,
    // а заодно сжигать жертву
    public void burn(Object obj){ //сжечь объект
    }
}
```

Статический блок

- Класс может иметь в себе участок кода, выполняющийся при инициализации класса

```
class Dragon {  
    // статический блок выполняется до того,  
    // как создан первый экземпляр класса  
    static {  
        System.out.println("Dragons are alive!");  
    }  
    //конструктор может не выполниться ни разу  
    //В то время как статический инициализатор  
    //выполнится при загрузке  
    public Dragon(){  
        System.out.println("New dragon was born");  
    }  
}
```

Порядок инициализации

- Инициализация членов класса и выполнение статического инициализатора происходит в порядке их описания в классе

```
class Dragon {  
    static int dragonCount = 10;  
    static {  
        //ошибка - переменная dragonEnemy еще не инициализирована  
        System.out.println(dragonEnemy);  
        //OK - переменная dragonCount уже проинициализирована  
        System.out.println(dragonCount);  
    }  
    static String dragonEnemy = "Phoenix";  
}
```

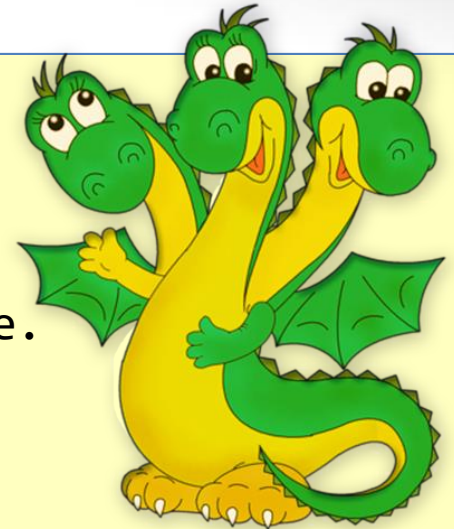
Константы

- Константа - это именованное значение, неизменяемое стандартными средствами языка Java

```
class Dragon {  
    final static int headCount = 1;  
}  
Dragon.headCount = 3;  
// ошибка - попытка присвоить значение константе.  
// Дракон - не Змей Горыныч!
```



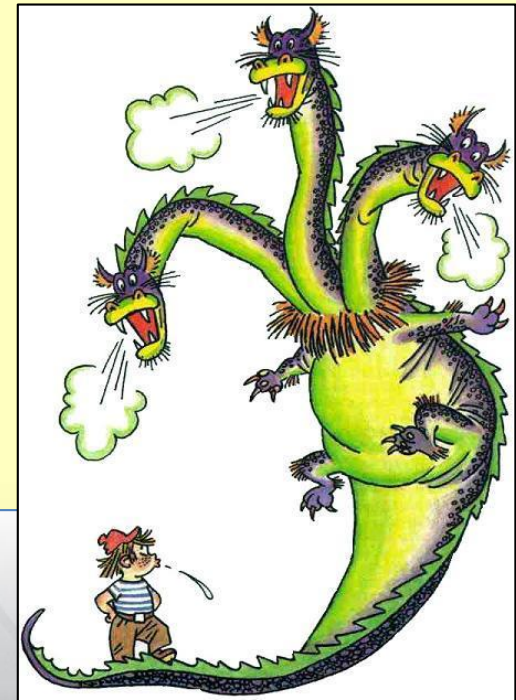
```
class Gorinich {  
    //MutableInt - это класс-хранилище целого числа,  
    // позволяющий изменять его  
    final static MutableInt headCount = new MutableInt(1);  
}  
Gorinich.headCount.setValue(3);  
// ОК, т.к. значение указателя не меняется,  
// меняется только содержимое
```



Константы в статическом блоке

- Инициализацию константы можно отложить, но только до времени выполнения статического блока класса
- Допустимо произвести только одно присваивание константе

```
class Dragon {  
    //нет инициализации  
    final static int headCount;  
    static {  
        //OK - 1-я инициализация  
        headCount = 1;  
        //Ошибка - константа уже присвоена  
        headCount = 3;  
    }  
}
```



Абстрактный класс

- Класс является абстрактным, если имеет модификатор `abstract`
- Класс должен быть помечен этим модификатором, если у него хоть один абстрактный метод (помечен словом `abstract` и не имеет реализации)

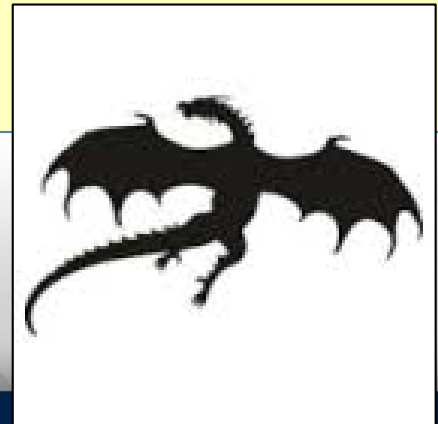
```
abstract class FlyingThing {  
    protected String name;  
    abstract public void fly();  
    public String getName(){  
        return name;  
    }  
}
```

//ошибка, абстрактный класс не может иметь реализаций
`FlyingThing aThing = new FlyingThing();`

Наследование от абстрактного класса

- Как правило, абстрактный класс служит для создания базы дерева наследования классов

```
class Dragon extends FlyingThing {  
    public void fly(){  
        flySomewhere(); //реализуем полет куда-нибудь  
    }  
}  
  
// OK - создавать экземпляры можно  
Dragon d = new Dragon();  
// OK - создание ссылки на абстрактный класс и инициализация  
// конкретным классом  
FlyingThing ft = new Dragon();
```



Реализация интерфейса

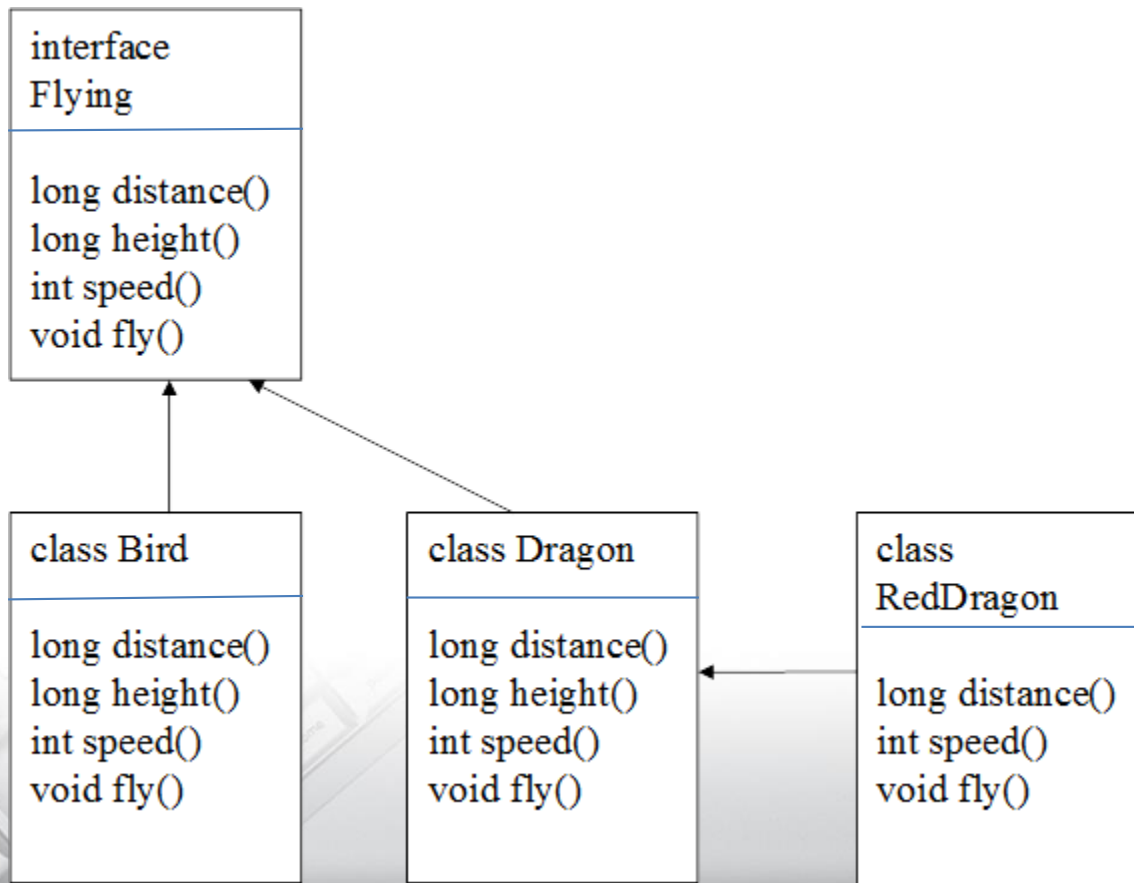
- Интерфейс – это сущность, предназначенная для формирования структуры реализующего его класса или для наследования другим интерфейсом



```
public interface Flying{  
    // класс, реализующий данный интерфейс,  
    // должен предоставить реализацию для этого метода  
    int speed();  
}
```

Реализация интерфейса

- Класс может реализовывать множество интерфейсов
- Реализующий класс должен реализовать все методы интерфейса
- Интерфейсы могут наследоваться друг от друга



Реализация интерфейса

- Реализация позволяет снабдить класс дополнительными свойствами

```
public class Dragon implements Flying {  
    protected int speed;  
    public int speed(){  
        return speed;  
    }  
}  
public class RedDragon extends Dragon{  
    public int speed(){  
        return 2*speed;  
    }  
    public long distance(){...}  
    public long burn(Object obj){...}  
}
```

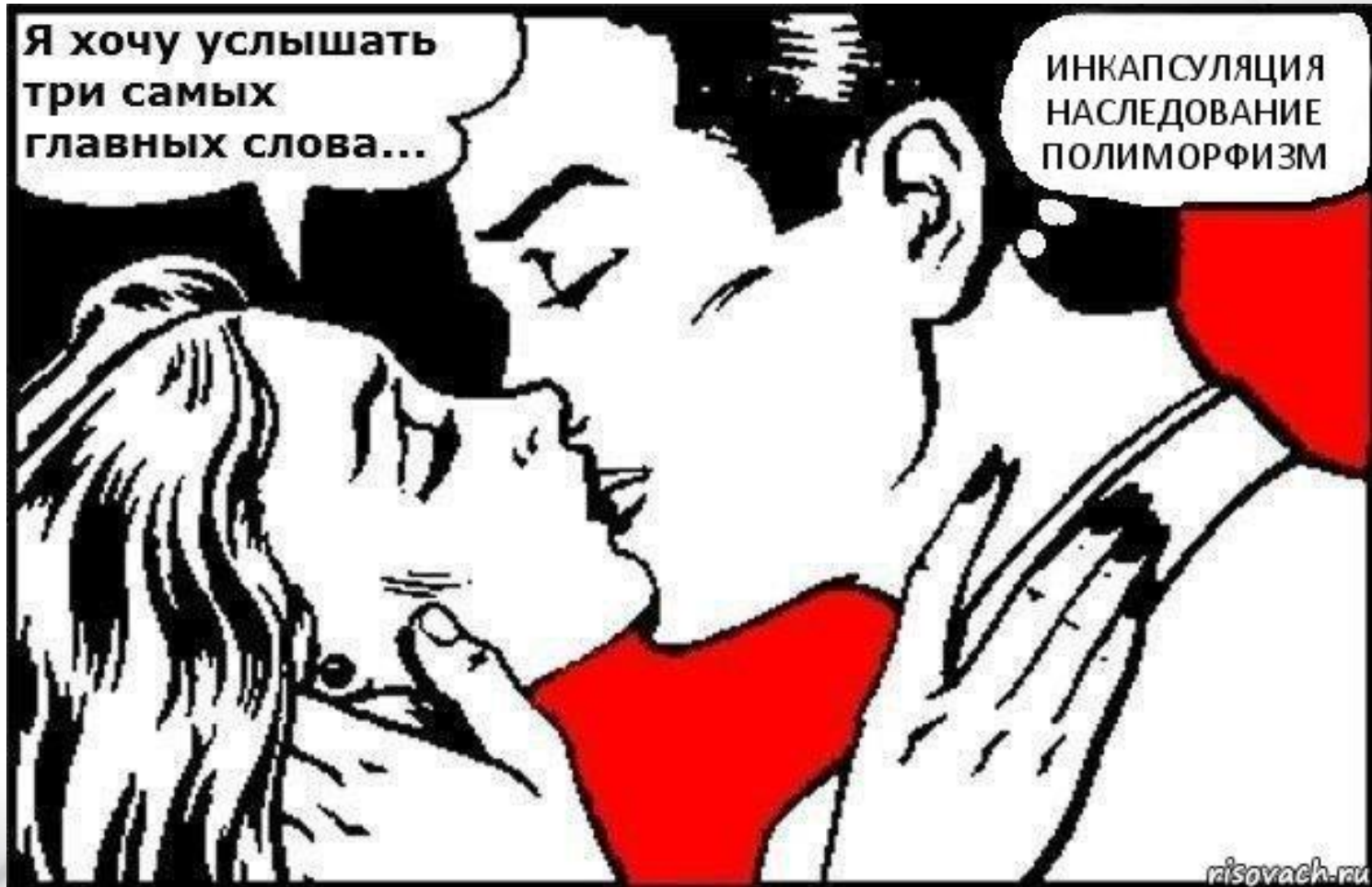




Демонстрація

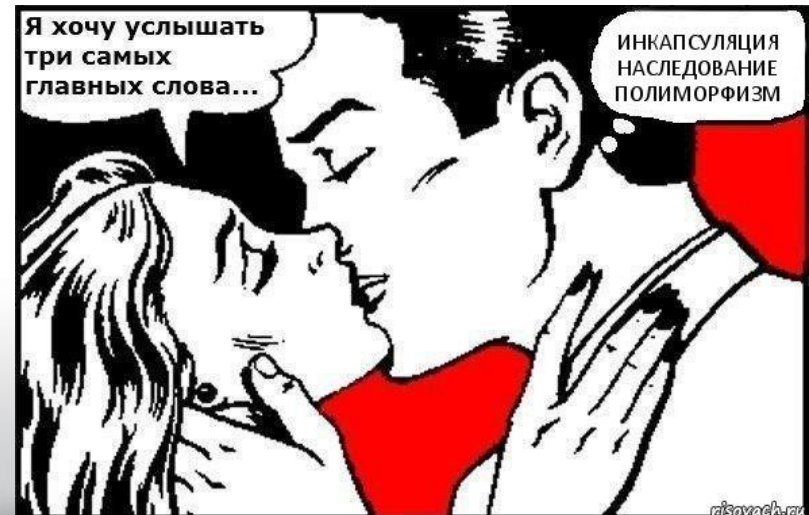


Основы ООП



Основы ООП

- *Инкапсуляция* (сокрытие данных) - объединение в одной сущности данных и методов работы с ними.
- *Наследование* – возможность класса-наследника приобретать признаки класса-предка.
- *Полиморфизм* – способность наследников по-другому реализовывать возможности предков. Объект способен проявлять признаки своего предка.



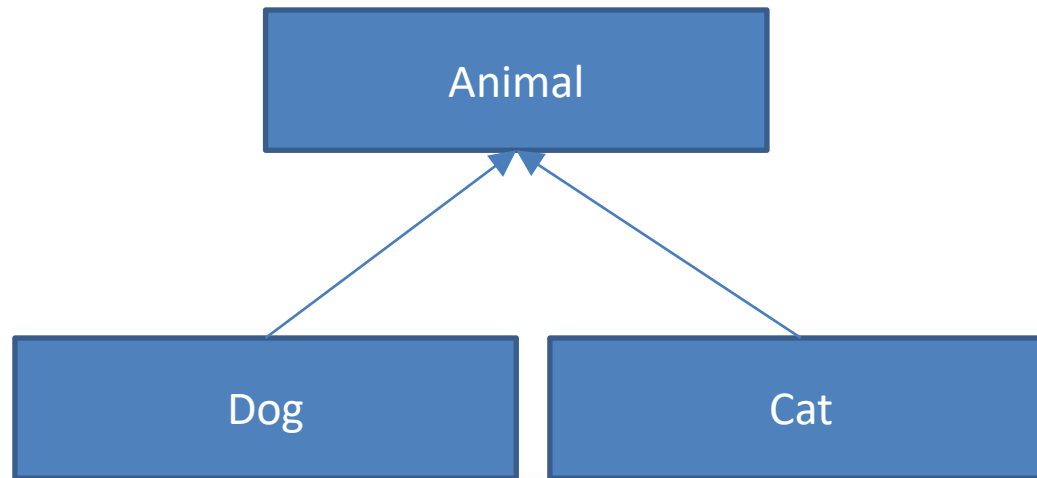
Инкапсуляция

- Состояние объекта определяется значениями его полей.
- Соккрытие данных осуществляется с помощью установки модификаторов, влияющих на видимость членов класса.
- В языке Java используется несколько уровней соккрытия.



Наследование

- Наследование - это такое отношение между классами, когда один класс повторяет структуру и поведение другого класса (одиночное наследование) или других классов (множественное наследование – отсутствует в Java).



Полиморфізм

- Екземпляр класу може виступати як екземпляр любого класу-предка данного класу.

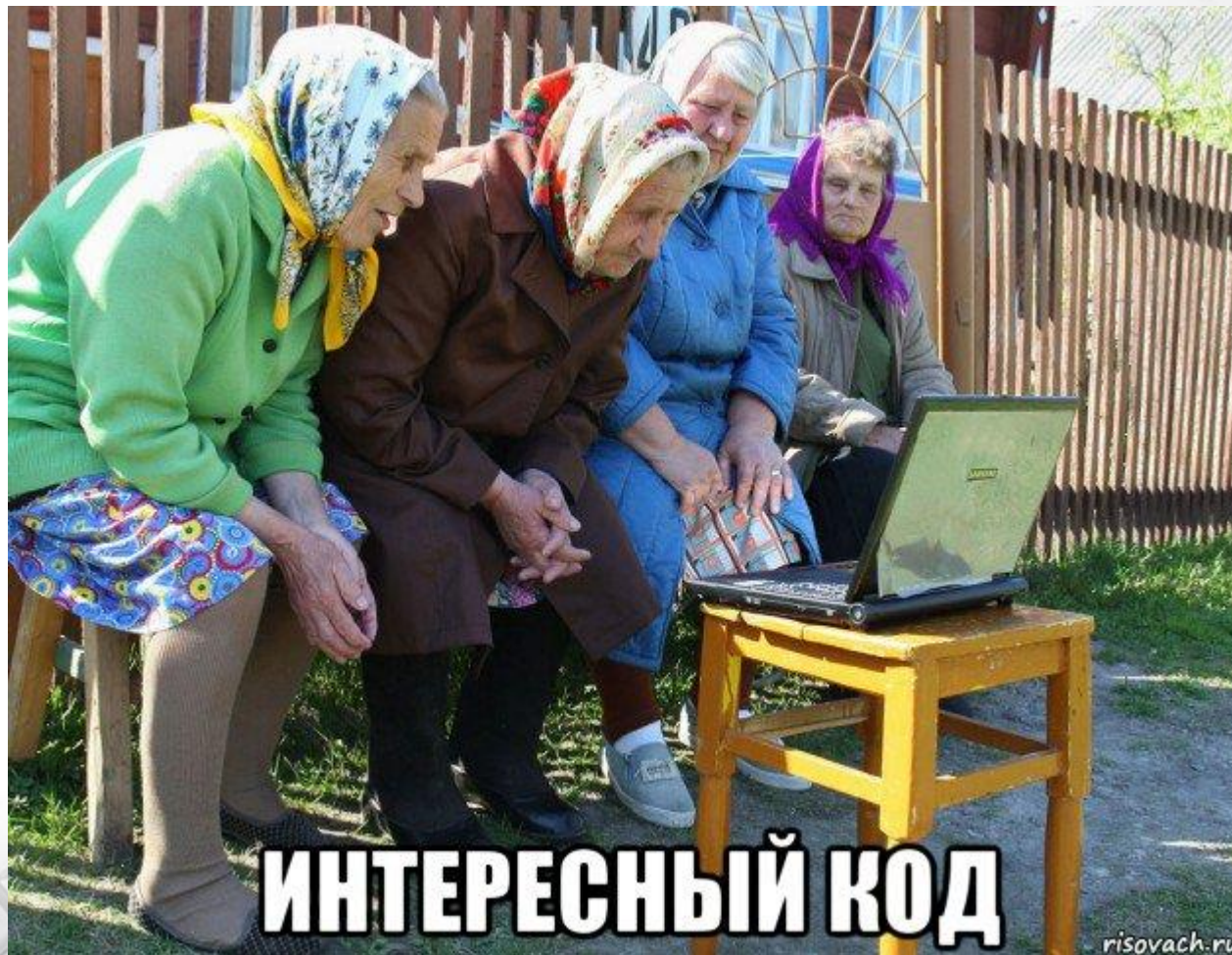
```
class Animal {  
    void talk(){...};  
}  
class Dog extends Animal {  
    void talk() { System.out.println("Woof!"); }  
}  
class Cat extends Animal {  
    void talk() { System.out.println("Meow!"); }  
}
```

Екземпляр класу Dog или Cat одночасно є екземпляром класу Animal:

```
Animal myDog = new Dog();  
Animal myCat = new Cat();
```



Демонстрація



ИНТЕРЕСНЫЙ КОД





НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ
КОРАБЛЕБУДУВАННЯ
ІМЕНІ АДМІРАЛА МАКАРОВА

QUESTIONS
& **A**NSWERS

ОСНОВЫ синтаксиса языка Java

Евгений Беркунский, НУК

eugeny.berkunsky@gmail.com

<http://www.berkut.mk.ua>

