



Lesson 5

27.10.2022


```
public class Ex1 {  
    static int a = 1111;  
  
    static {  
        System.out.println("static");  
        a = a-- - --a;  
    }  
  
    {  
        System.out.println("non static");  
        a = a++ + ++a;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(a);  
    }  
}
```

```
public class Ex2 {  
    public static void main(String[] args) {  
        Integer i1 = 128;  
        Integer i2 = 128;  
        System.out.println(i1 == i2);  
  
        Integer i3 = 127;  
        Integer i4 = 127;  
        System.out.println(i3 == i4);  
    }  
}
```



```
public class Ex3 {  
    public static void show() {  
        System.out.println("Static method called");  
    }  
  
    public static void main(String[] args) {  
        Ex3 obj = null;  
        obj.show();  
    }  
}
```

```
public class Ex4 {  
    static int method1(int i) {  
        return method2(i *= 11);  
    }  
    static int method2(int i) {  
        return method3(i /= 11);  
    }  
    static int method3(int i) {  
        return method4(i -= 11);  
    }  
    static int method4(int i) {  
        return i += 11;  
    }  
    public static void main(String[] args) {  
        System.out.println(method1(11));  
    }  
}
```



Парадигма программирования – это совокупность принципов, методов и понятий, определяющих способ конструирования программ.

- 1 - Императивное программирование
- 2 - Структурное программирование
- 3 - Объектно-ориентированное программирование
- 4 - Функциональное программирование
- 5 - Логическое программирование



Объектно-ориентированное программирование

В данной парадигме программирования программа разбивается на **объекты** – структуры данных, состоящие из полей, описывающих *состояние*, и **методов** – подпрограмм, применяемых к объектам для изменения или запроса их состояния. В большей части объектно ориентированных парадигмах для описания объектов используются классы, объекты более высокого порядка, описывающие структуру и операции, связанные с объектами.

Основные механизмы управления/абстракции:

Объект


Класс

Иерархии классов/объектов

Полиморфизм

Инкапсуляция

Наследование



```
class Плита {  
    Горит Ли Конфорка? (конфорка)  
    Зажечь Конфорку (конфорка);  
    Потушить Конфорку (конфорка);  
    Установить Уровень Нагрева (конфорка, уровень);  
}  
class Чайник {  
    Пустой ли Чайник();  
    В Процессе Нагрева();  
    Поставить На Плиту(плита, конфорка);  
}
```




Свойства объекта

- Объект является экземпляром класса
- Объект имеет внутреннее состояние
- Объект может принимать сообщения
(в большинстве языков сообщение = вызов метода)

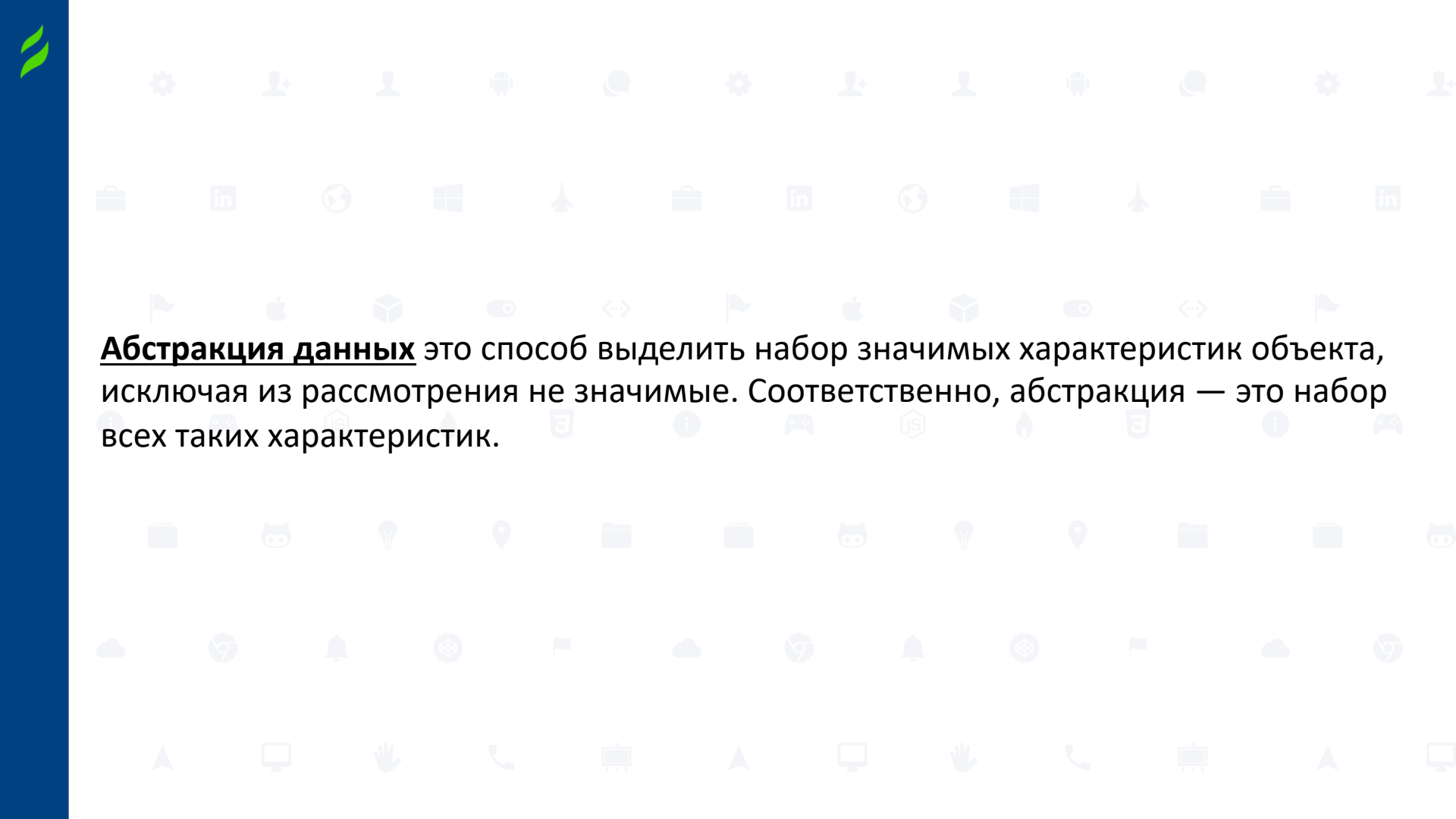


Три принципа ООП

Инкапсуляция это свойство системы, позволяющее объединить данные и методы в классе, и скрыть детали реализации от пользователя.

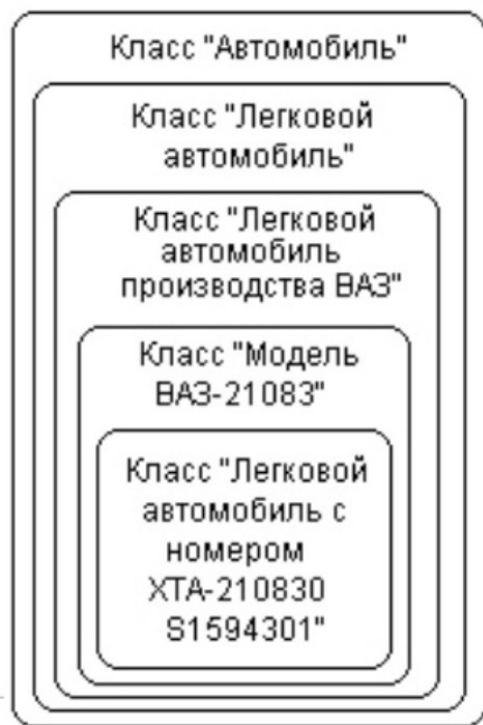
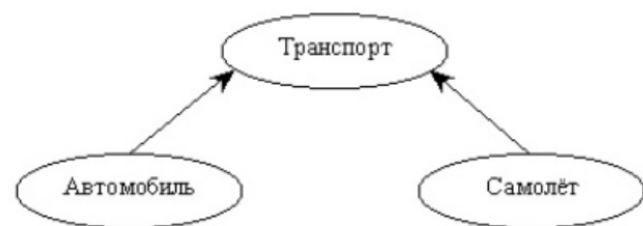
Наследование это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью

Полиморфизм “один интерфейс, множество методов”. Реализации полиморфизма в языке Java - это перегрузка и переопределение методов, интерфейсы.



Абстракция данных это способ выделить набор значимых характеристик объекта, исключая из рассмотрения не значимые. Соответственно, абстракция — это набор всех таких характеристик.





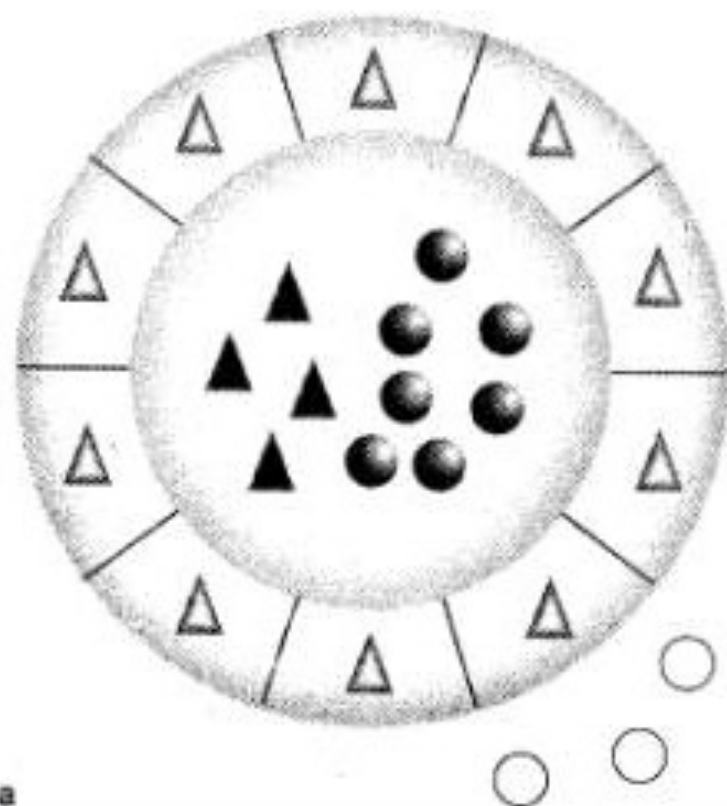
Inheritance



you can create new type of animal
changing or adding properties



Класс

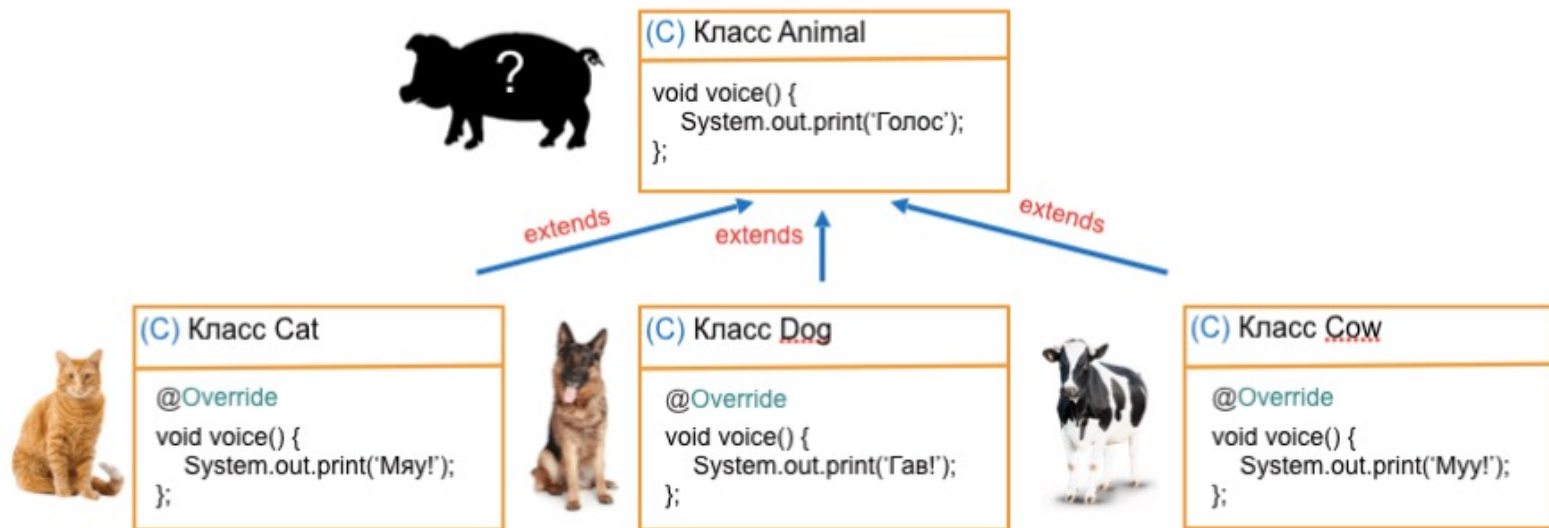


Incapsulation

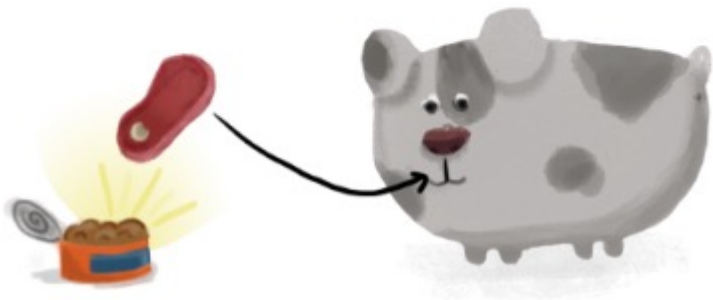


every animal eats
and then poop

Полиморфизм



Polymorphism



each animal can eat
its own type of food



Class vs. Object

Car class



Car
Objects



Green
Ford
Mustang
Gasoline



Red
Toyota
Prius
Electricity



Blue
Volkswagon
Golf
Deisel

Box myBox;



null

myBox = new Box();



Box

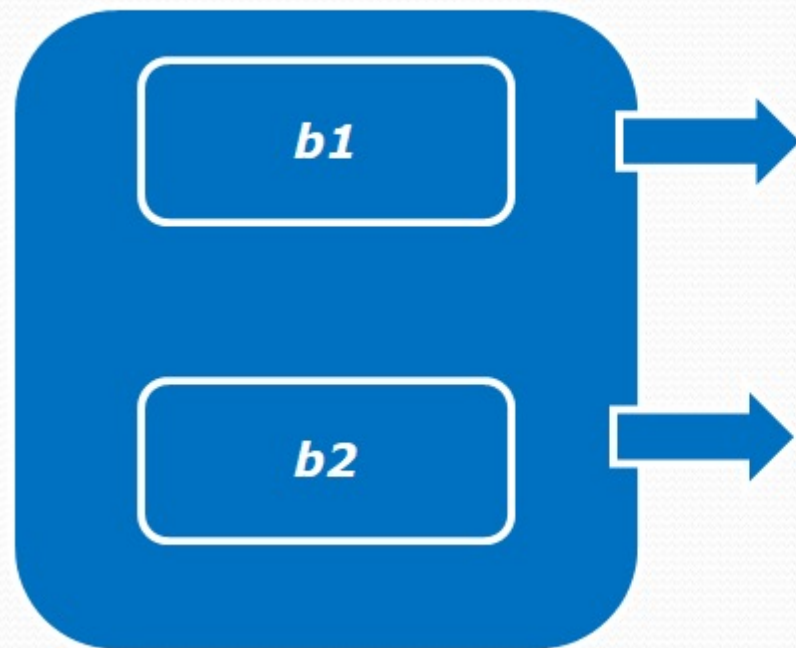
double height;

double depth;

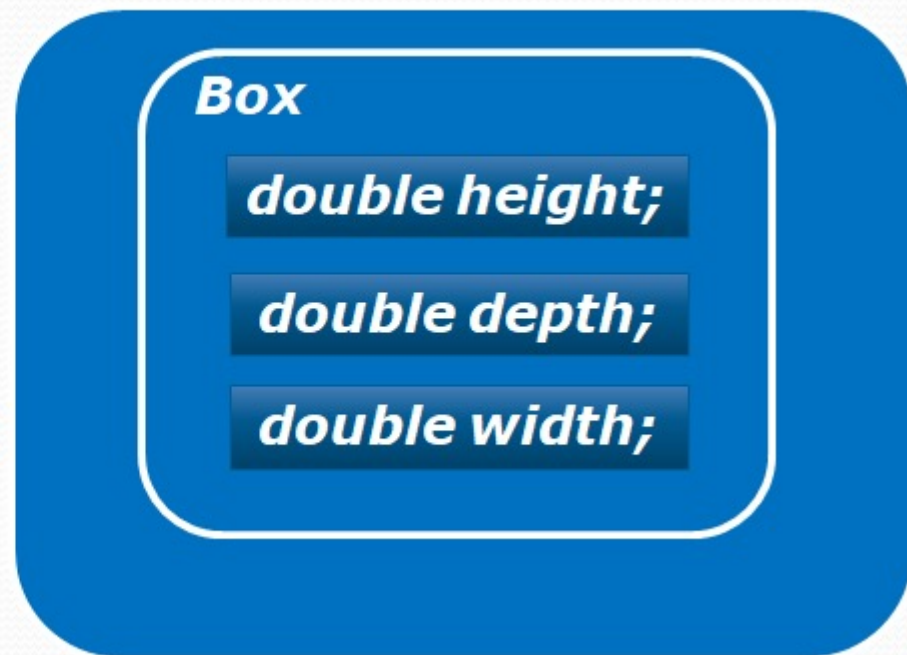
double width;


Heap

stack



Heap





В языке Java при проектировании классов принято ограничивать уровень доступа к переменным с помощью модификатора `private` и обращаться к ним через геттеры и сеттеры.

Существуют правила объявления таких методов, рассмотрим их:

- Если свойство НЕ типа `boolean`, префикс геттера должно быть `get`. Например: `getName()` это корректное имя геттера для переменной `name`.
- Если свойство типа `boolean`, префикс имени геттера может быть `get` или `is`. Например, `getPrinted()` или `isPrinted()` оба являются корректными именами для переменных типа `boolean`.
- Имя сеттера должно начинаться с префикса `set`. Например, `setName()` корректное имя для переменной `name`.
- Для создания имени геттера или сеттера, первая буква свойства должна быть изменена на большую и прибавлена к соответствующему префиксу (`set`, `get` или `is`).
- Сеттер должен быть `public`, возвращать `void` тип и иметь параметр соответствующий типу переменной.
- Геттер метод должен быть `public`, не иметь параметров метода, и возвращать значение соответствующее типу свойства.



Класс Object

Является суперклассом для **всех** классов (включая массивы)

Переменная этого типа может ссылаться на **любой** объект (но не на переменную примитивного типа)

Его методы наследуются **всеми** классами

Реализует базовые операции с объектами



Получение строкового представления объекта

`String toString()`

Получение ссылки на описание класса объекта

`final Class getClass()`

Клонирование объекта (получение копии)

`protected Object clone()`

Проверка равенства объектов

`boolean equals(Object obj)`

Получение хэш-кода объекта

`int hashCode()`

Метод завершения работы с объектом

`protected void finalize()`

Методы обслуживания блокировок в многопоточных приложениях

`void wait(...), void notify(), void notifyAll()`