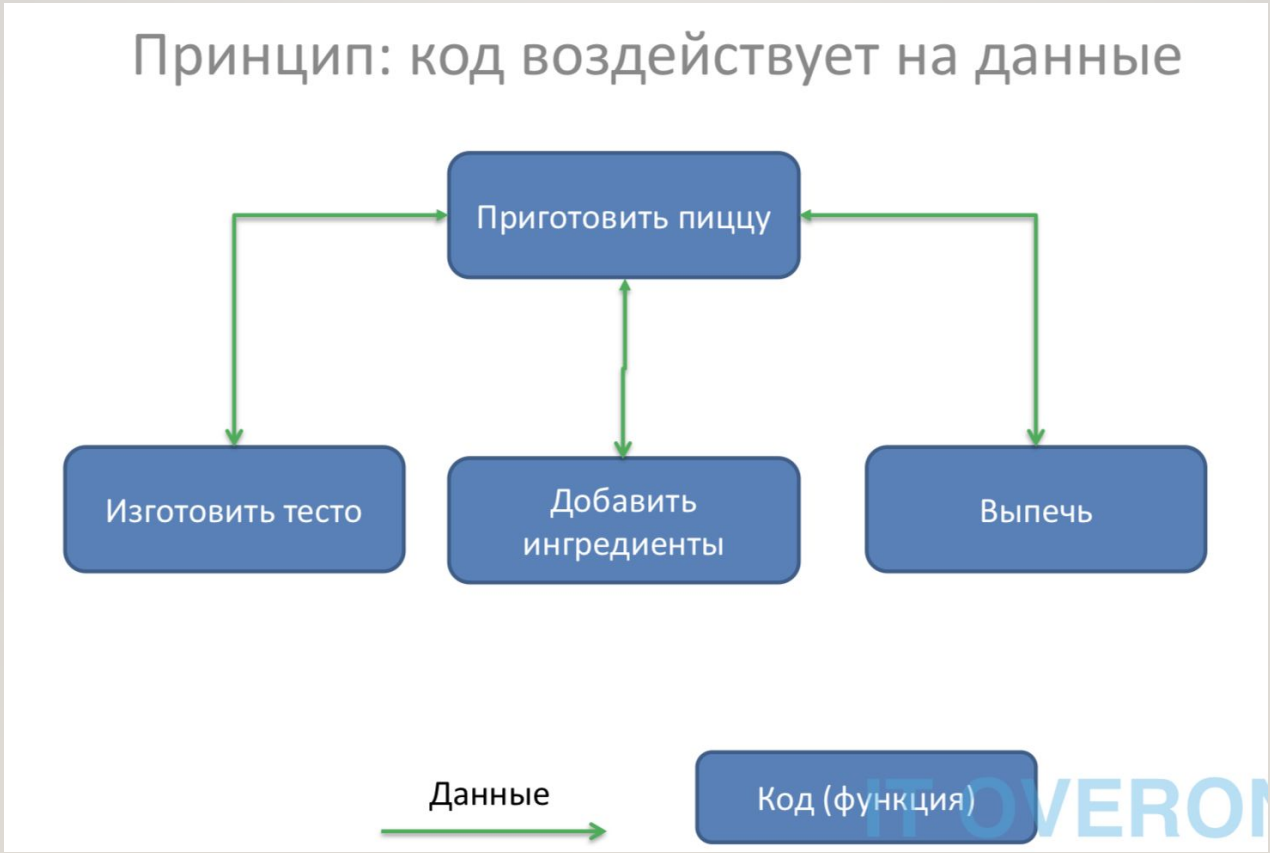
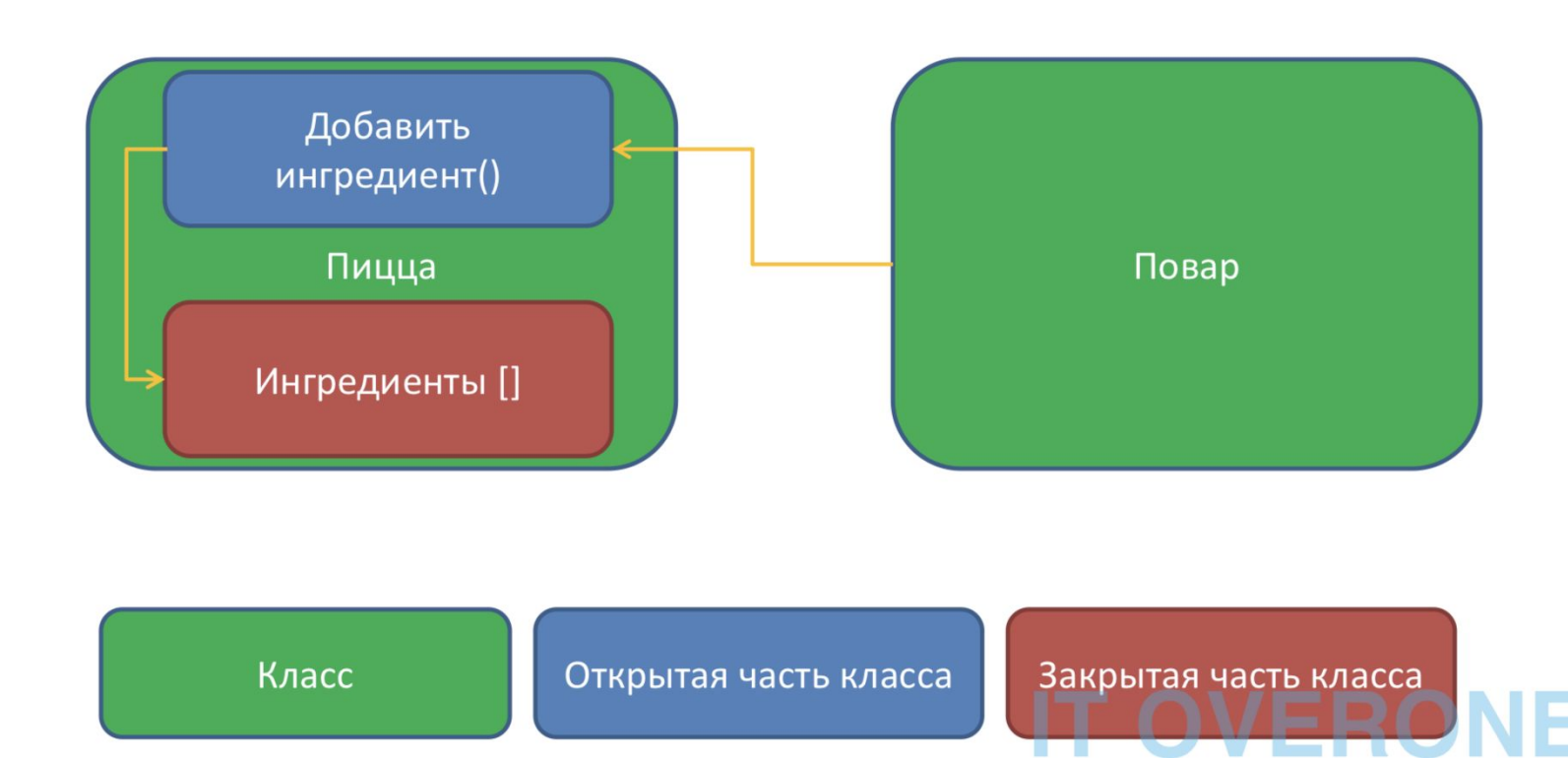
ЗАНЯТИЕ 16

СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ



ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ



КЛАССЫ И ОБЪЕКТЫ

- Java является объектно-ориентированным языком, поэтому такие понятия как "класс" и "объект" играют в нем ключевую роль. Любую программу на Java можно представить как набор взаимодействующих между собой объектов.
- Шаблоном или описанием объекта является **класс**, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке - наличие двух рук, двух ног, головы, туловища и т.д. Есть некоторый шаблон - этот шаблон можно назвать классом. Реально же существующий человек (фактически экземпляр данного класса) является объектом этого класса.
- Person.java, Program.java

КОНСТРУКТОРЫ. БЕЗ ПАРАМЕТРОВ

- Кроме обычных методов классы могут определять специальные методы, которые называются **конструкторами**. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.
- !!! Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор без параметров.
- Person.java
- Конструктор без параметров: Для создания объекта Person используется выражение new Person().
  Оператор **new** выделяет память для объекта Person. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта Person. А переменная »person » получит ссылку на созданный объект.
- Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных числовых типов это число 0, а для типа string и классов - это значение null (то есть фактически отсутствие значения).

КОНСТРУКТОРЫ. С ПАРАМЕТРАМИ

- Если необходимо, чтобы при создании объекта производилась какая-то логика, например, чтобы поля класса получали какие-то определенные значения, то можно определить в классе свои конструкторы с параметрами
- **Ключевое слово this**
- Ключевое слово **this** представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы

КОНСТРУКТОРЫ

Конструктор

- Конструктор и метод внешне похожи
- Конструктор имеет имя как у класса
- Если конструктор не указан – компилятор создаст конструктор по умолчанию
- В конструкторе не должно быть лишней логики
- Конструктор имеет модификатор доступа
- Если создали свой конструктор – конструктор по умолчанию не создаётся!

ИНИЦИАЛИЗАТОРЫ

- Кроме конструктора начальную инициализацию объекта вполне можно было проводить с помощью инициализатора объекта. Инициализатор выполняется до любого конструктора. То есть в инициализатор мы можем поместить код, общий для всех конструкторов:

МОДИФИКАТОРЫ ДОСТУПА

- Все члены класса в языке Java - поля и методы - имеют модификаторы доступа. В прошлых темах мы уже сталкивались с модификатором public. Модификаторы доступа позволяют задать допустимую область видимости для членов класса, то есть контекст, в котором можно употреблять данную переменную или метод.

В Java используются следующие модификаторы доступа:

- **public**: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов.
- **private**: закрытый класс или член класса, противоположность модификатору public. Закрытый класс или член класса доступен только из кода в том же классе.
- **protected**: такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
- **Модификатор по умолчанию**. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.

ИТОГО

- ООП – объектно-ориентированное программирование – методология программирования с помощью объектов, которые являются экземплярами конкретного класса.
- класс — это описание того, какими свойствами и поведением будет обладать объект. А объект — это экземпляр с собственным состоянием этих свойств
- «свойства» — это такие же обычные переменные, просто они являются атрибутами какого-то объекта (их называют полями объекта). Аналогично «поведение» — это функции объекта (их называют методами), которые тоже являются атрибутами объекта.

Нужно ли всегда создавать объекты?

- Даже если программа простейшая – всегда нужно создавать объекты и писать код в стиле ООП!
- Это должно быть привычкой
- В программе не должно быть лишних объектов
- Никогда не давайте объекту чужие понятия и действия

ПРИНЦИПЫ ООП

- Концепции ООП являются основополагающими элементами и составляют основу языка программирования Java. В рамках данного подхода выделяют следующие термины: **абстракция**, **инкапсуляция**, **наследование** и **полиморфизм**. Понимание данных принципов служит ключом к построению целостной картины того, как работают программы, написанные на Java. По большому счету, объектно-ориентированный подход позволяет нам описывать классы, определять методы и переменные таким образом, чтобы затем использовать их вновь, частично либо полностью, без нарушения безопасности.

АБСТРАКЦИЯ

- Абстракция означает использование простых вещей для описания чего-то сложного.
- Гораздо проще сказать «на столе стоит кружка» нежели
- В этом и состоит суть абстракции – в абстрагировании длинного описания в 1.

«На столе стоит предмет цилиндрической формы, из керамики, с ручкой, предназначенный для чая или кофе.»

«Абстракция является основой объектно-ориентированного программирования и позволяет работать с объектами, не вдаваясь в особенности их реализации.»

- Абстракция позволяет представить любой объект из реального мира в программе и выделить необходимые его свойства/поведение
- Animal.java

ИНКАПСУЛЯЦИЯ

- Под инкапсуляцией подразумевается сокрытие полей внутри объекта с целью защиты данных от внешнего, бесконтрольного изменения со стороны других объектов. Доступ к данным (полям) предоставляется посредством публичных методов (геттеров/сеттеров). Это защитный барьер позволяет хранить информацию в безопасности внутри объекта.
- Ответ для собеса: когда атрибуты и поведение объекта объединяются в одном классе, внутренняя реализация скрывается, а пользователю предоставляется функционал для работы

НАСЛЕДОВАНИЕ

- позволяет описывать новые классы на основе уже существующих. При этом поля и методы класса-предка становятся доступны и классам-наследникам. Это делает классы более чистыми и понятным за счет устранения дублирования программного кода.
- Класс, который наследуется называется дочерним (или подклассом). Класс, от которого наследуется новый класс — называется родительским, предком и т. д. В языке программирования Java используется ключевое слово **extends** для того, чтобы указать на класс-предок.
- Cat.java
- Dog.java

ПОЛИМОРФИЗМ

- Данный принцип позволяет программистам использовать одни и те же «термины» для описания различного поведения, зависящего от контекста.
- Когда у 1 метода есть множество реализаций
- Ответ для собеса: когда программа может использовать объекты с одинаковым интерфейсом (предком) без информации о внутреннем устройстве

ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ

- **Переопределение метода** (англ. *Method overriding*) в объектно-ориентированном программировании — одна из возможностей языка программирования, позволяющая подклассу или дочернему классу обеспечивать специфическую реализацию метода, уже реализованного в одном из суперклассов или родительских классов.
  Переопределение позволяет взять какой-то метод родительского класса и написать в каждом классе-наследнике свою реализацию этого метода. Новая реализация «заменит» родительскую в дочернем классе.

ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ. ОГРАНИЧЕНИЯ

- **У переопределенного метода должны быть те же аргументы, что и у метода родителя.**

Если метод `voice` родительского класса принимает на вход `String`, переопределенный метод в классе-потомке тоже должен принимать на вход `String`, иначе компилятор выдаст ошибку:

```
1  public class Animal {
2
3     public void voice(String s) {
4
5         System.out.println("Голос! " + s);
6     }
7  }
8
9  public class Cat extends Animal {
10
11    @Override//ошибка!
12    public void voice() {
13        System.out.println("Мяу!");
14    }
15 }
```

ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ. ОГРАНИЧЕНИЯ

- **У переопределенного метода должен быть тот же тип возвращаемого значения, что и у метода ро**

В ином случае мы получим ошибку компиляции:

```
1  public class Animal {
2
3      public void voice() {
4
5          System.out.println("Голос!");
6      }
7  }
8
9
10 public class Cat extends Animal {
11
12     @Override
13     public String voice() {          //ошибка!
14         System.out.println("Мяу!");
15         return "Мяу!";
16     }
17 }
```

ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ. ОГРАНИЧЕНИЯ

- **Модификатор доступа у переопределенного метода также не может отличаться от «оригин**

```
1  public class Animal {
2
3      public void voice() {
4
5          System.out.println("Голос!");
6      }
7  }
8
9  public class Cat extends Animal {
10
11     @Override
12     private void voice() {       //ошибка!
13         System.out.println("Мяу!");
14     }
15 }
```

- Д/З: отличие перегрузки метода от переопределения
- https://www.turbopro.ru/index.php/yazyk-programmirovaniya-java/7705-25-peregruzka-i-pereopredelenie-metodov
- «статическое vs динамическое связывание (или раннее и позднее)в Java»
- https://javarush.ru/groups/posts/439-razlichija-mezhdu-rannim-i-pozdnim-svjazihvaniem-v-java

КЛЮЧЕВОЕ СЛОВО STATIC

- Для объявления статических переменных, констант, методов и инициализаторов перед их объявлением указывается ключевое слово **static**.
- **Статические поля**:
- При создании объектов класса для каждого объекта создается своя копия нестатических обычных полей. А статические поля являются общими для всего класса. Поэтому они могут использоваться без создания объектов класса.
- Point_3/Program-Person.java

КЛЮЧЕВОЕ СЛОВО STATIC

- **Статические константы:**
- Также статическими бывают константы, которые являются общими для всего класса.
- Point_3/Math.java
- Стоит отметить, что на протяжении всех предыдущих тем уже активно использовались статические константы. В частности, в выражении: System.out.println("hello");
- **out** как раз представляет статическую константу класса System. Поэтому обращение к ней идет без создания объекта класса System.

КЛЮЧЕВОЕ СЛОВО STATIC

- **Статические инициализаторы: ДЗ**
- **Статические методы**
- Статические методы также относятся ко всему классу в целом. Вызываем статический метод по имени класса. Нестатический – через объект.

КЛЮЧЕВОЕ СЛОВО STATIC

- Статичному элементу запрещено использовать нестатичные переменные и методы класса.
- Статичные элементы не манипулируют свойствами объекта и не привязаны к конкретному объекту.
- Статичные методы и свойства можно вызывать:

  1. Через имя класса

  2. Через ссылку на экземпляр класса
- Чаще используется первый вариант
- Статичный элемент связан не с объектом, а с классом, следовательно его нельзя переопределить