Программирование на языке Java

Тема 32. Основы объектно-ориентированного программирования

ООП

Объектно-ориентированное программирование (ООП) – парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

Основные принципы ООП

- Все является объектом.
- Вычисления осуществляются путем взаимодействия (обмена данными) между объектами, при этом один объект требует от другого выполнить какое-либо действие.
- Каждый объект имеет независимую память, которая состоит из других объектов.
- Каждый объект является экземпляром класса, который выражает общие свойства объектов.
- В классе задается функциональность объекта.
- Классы организованы в единую древовидную структуру с общим корнем.

Объекты

Объект обладает состоянием, поведением и идентичностью.

Структура и поведение схожих объектов определяет общий для них класс.

Термины «экземпляр класса» и «объект» взаимозаменяемы.

Пример. Точка на плоскости с координатами (1,3) – объект или экземпляр класса «Точка».

Состояние и поведение объекта

Состояние объекта характеризуется перечнем всех свойств данного объекта и текущими значениями каждого из этих свойств.

Пример. Свойства человека: фамилия, имя, отчество, пол, дата рождения, номер паспорта и т.д.

Пример. Свойства точки: координата по оси абсцисс, координата по оси ординат, цвет точки.

Поведение это то, как объект действует и реагирует.

Пример. Поведение человека: сидеть, ходить, спать, учиться и т.д.

Пример. Поведение точки: перемещаться по оси абсцисс, перемещаться по оси ординат, менять цвет.

Программирование на языке Java

Тема 33. Знакомство с классами

Класс

Класс – центральный компонент Java.

Класс образует основу ООП в среде Java.

Классы, которые создавались нами до этого, служили только в качестве контейнеров для метода **`main()`**.

Класс определяет новый тип данных. После того, как он определен, этот новый тип можно применять для создания объектов данного типа.

Класс – шаблон объекта.

Объект – экземпляр класса.

Термины объект и экземпляр являются синонимами.

Класс

Класс – некое множество объектов, имеющих общую структуру и общее поведение.

Свойства (поля) – состояния, которые может принимать объект.

Методы – виды поведения, которые может осуществлять объект.

Пример. Класс Человек, Класс Точка.

Использование Java-классов

Класс – программная единица, которая представляет:

- программу / модуль – программа, которую можно выполнить, содержащая метод **`main`**;
- **шаблон для создания новых объектов** – «схема» для нового типа данных.

Аналогия со схемой

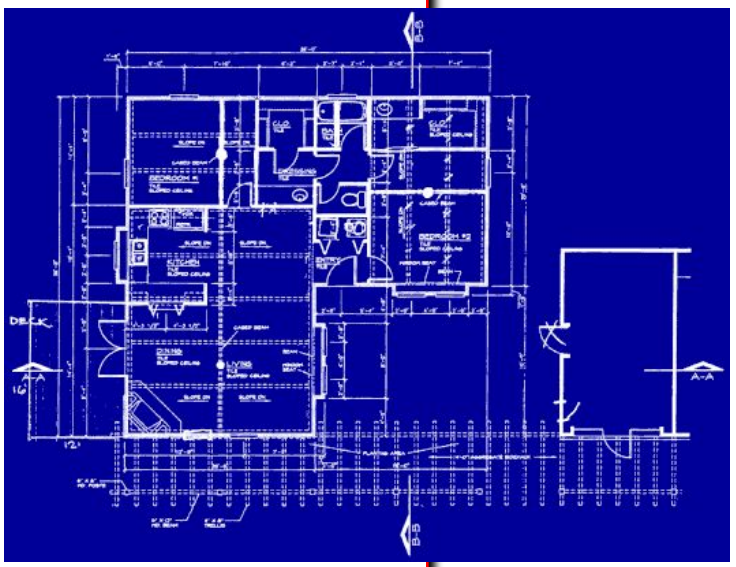
Схема iPod

Состояние:

текущаяПесня
громкость
уровеньЗаряда

Поведение:

Включить/Выключить
СменитьПесню
ИзменитьГромкость
ВыбратьСлучайнуюПесню



iPod#1

Состояние:

текущаяПесня=“Help”
Громкость = 29
уровеньЗаряда = 34

Поведение:

Включить/Выключить
…

iPod#2

Состояние:

текущаяПесня=“Girl”
Громкость = 17
уровеньЗаряда = 95

Поведение:

Включить/Выключить
…

Абстракция

Абстракция – разделение между идеей и реализацией. В ООП можно использовать объекты, не зная как они работают

Абстракция с iPod:

- Вы понимаете внешнее поведение плеера (кнопки, экран);
- Вы не знаете его внутреннее устройство

Но это и не нужно, чтобы пользоваться плеером!

Общая форма класса

имя класса

```
class ИмяКласса {
тип свойство1;
тип свойство2;
...
тип свойствоn;

тип имяМетода1(список параметров) {
// тело метода
}
...
тип имяМетодаM(список параметров) {
// тело метода
}
```

Набор данных

Поведение

Члены класса

Данные, или переменные, определенные внутри класса, называются **свойствами** или **полями**.

Код класса находится внутри **методов**.

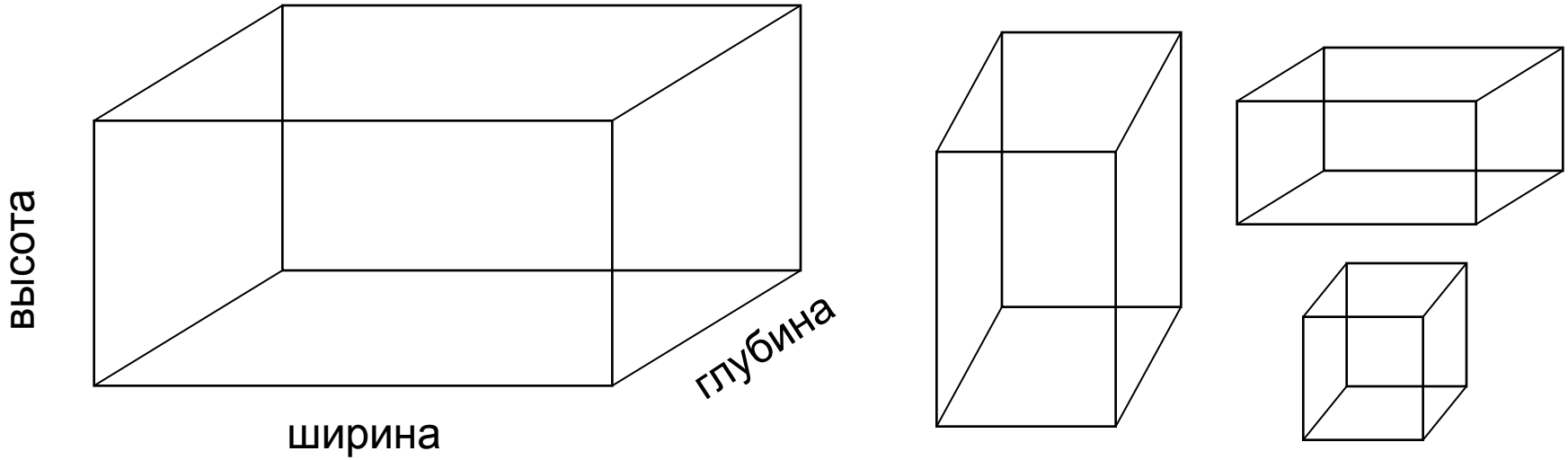
Членами класса – называются свойства и методы, определенные внутри класса.

Внимание! Общая форма класса не содержит метода **`main()`**, он не является обязательным для класса.

Метод **`main()`** служит начальной точкой программы.

Простой класс – 1

Код класса **Box** (прямоугольный параллелепипед):




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

Свойства: ширина, высота, глубина

Простой класс – 2

Класс определяет новый тип данный, в данном случае тип данных **Box**.

Объявление **class** создает действительный объект.

Создание объекта **myBox** типа **Box**

Чтобы создать объект **Box** нужно использовать оператор вида:

```
Box myBox = new Box();
```

MyBox – экземпляр класса **Box**.

При каждом создании экземпляра класса создается новый объект, который содержит собственную копию каждого свойства класса, определенной классом.

Простой класс – 3

Присвоение переменной **`width`** объекта **`myBox`** значения 10

Доступ к переменным экзем
применением операции точка (`.`).

```
Box myBox = new Box();
myBox.width = 10;
```

Операцию точка используют для доступа к переменным экземпляра класса и методам внутри объекта.

Простой класс – 4

Объявление класса **Box**

Объявление класса **BoxDemo** и метода **main**

Создание экземпляра класса **Box**

Присвоение переменным

Вычисление объема

```
class Box {
   double width;
   double height;
   double depth;
}
class BoxDemo {
 public static void mai
   Box myBox = new Box(
   myBox.width = 10;
   myBox.height = 20;
   myBox.depth = 15;
   double vol = myBox.width * myBox.height *
myBox.depth;
   printf("V = %f", vol);
}}
```

Вывод:

```
V = 3000.0
```

Простой класс – 5

Каждый объект содержит собственные копии переменных экземпляра.

Т.о. при наличии двух объектов Box каждый будет содержать собственные копии переменных width, height, depth.

Изменения переменных одного экземпляра класса не влияют на переменные другого экземпляра этого же класса.

Простой класс – 6

```
class BoxDemo {
 public static void main(String[] args) {
   Box myBox1 = new Box();
   Box myBox2 = new Box();
   myBox1.width = 10;
   myBox1.height = 20;
   myBox1.depth = 15;
   myBox2.width = 3;
   myBox2.height = 6;
   myBox2.depth = 9;
   double vol = myBox1.width * myBox1.height *
myBox1.depth;
   printf("V = %f\n", vol);
   vol = myBox2.width * myBox2.height *
myBox2.depth;
   printf("V = %f", vol);
}}
```

Вывод:

```
V = 3000.0
V = 162.0
```

Объявление объектов – 1

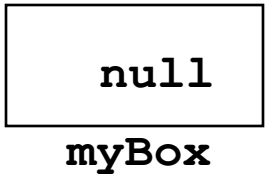
Создание объектов класса – двухступенчатый процесс:

1. Создание переменной типа класса (переменная не определяет объект, она может только ссылаться на объект).
2. Получение действительной, физической копии объекта и присвоение ее переменной – выполняется с помощью операции **`new`**.

   Операция **`new`** динамически (во время выполнения программы) распределяет память под объект и возвращает ссылку на него. Ссылка сохраняется в переменной.

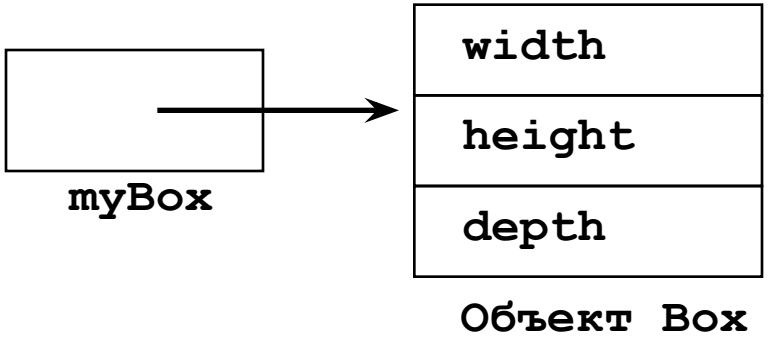
Объявление объектов – 2

```
Box myBox;
```



1. Объявление ссылки на объект. Переменная **myBox** содержит значение **null** (пустая ссылка). Попытка использовать переменную **myBox** приведет к возникновению ошибки.

```
myBox = new Box();
```



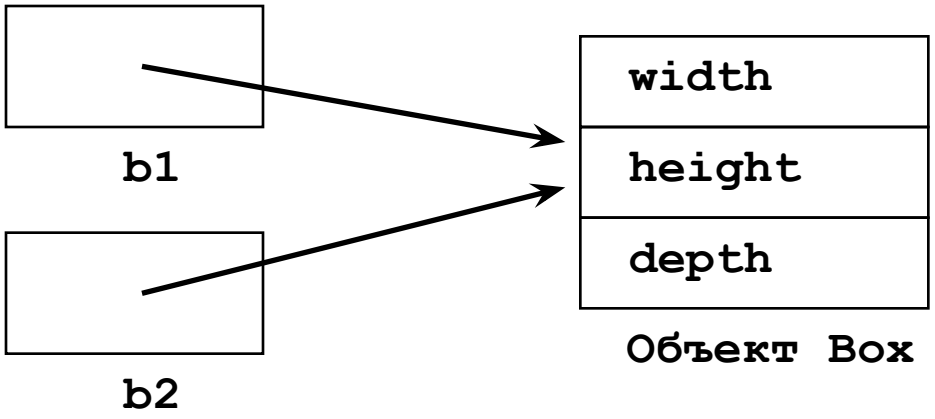
2. Распределение памяти под реальный объект **Box**. Присваивание переменной **myBox** ссылки на этот объект. Переменную **myBox** можно использовать как ссылку на объект **Box**.

Присваивание переменным объектных ссылок

```
Box b1 = new Box();
Box b2 = b1;
```

Внимание! Обе переменные **b1** и **b2** будут ссылаться на один и тот же объект.

Любые изменения, выполненные в объекте через **b2**, окажут влияние на объект, на который ссылается переменная **b1**, т.к. это один и тот же объект.



Создается не копия объекта, а копия ссылки

Методы – 1

Общая форма объявления метода:

```
тип имяМетода(список параметров) {
// тело метода
}
```

тип – тип данных, возвращаемый методом, может быть:

- простым типом (**byte, short, int, long, float, double, char, boolean**);
- типом класса, созданным программистом;
- **void** (не возвращает значения).

Список параметров – последовательность пар «тип идентификатор», разделенных запятыми

Методы – 2

Если тип возвращаемого значения отличается от **`void`**, то метод должен возвращать значение вызывающему методу с помощью оператора **`return`**:

```
return значение;
```

Методы определяют интерфейсы классов.

Это позволяет программисту, который реализует класс, скрывать конкретную схему внутренних структур данных за более понятными абстракциями метода.

Кроме методов, обеспечивающих доступ к данным, можно определять методы, используемые только внутри самого класса.

Добавление метода классу Box

```
class Box {
  double width;
  double height;
  double depth;

  void volume() {
    printf("V=%f", width * height * depth);
  }
}
```

Метод **`volume()`** выводит на экран объем объекта

Вызов метода класса Box – 1

```
class BoxDemo {
 public static void main(String[] args) {
   Box myBox1 = new Box();
   Box myBox2 = new Box();

   myBox1.width = 10;
   myBox1.height = 20;
   myBox1.depth = 15;

   myBox2.width = 3;
   myBox2.height = 6;
   myBox2.depth = 9;

   myBox1.volume();
   myBox2.volume();
}}
```

Вывод:

```
V = 3000.0
V = 162.0
```

Вызов метода класса Box – 2

Обращение к объекту **myBox1**

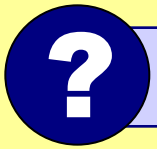
Вызов метода **volume()** по отношению к объекту **myBox1**

```
myBox1.volume();
```

Возвращение значения – 1

```
class Box {
   double width;
   double height;
   double depth;

   double volume() {
      return width * height * depth;
      }
}
```



Что плохо в методе **volume()**?

Возвращение значения – 2

```
class BoxDemo {
 public static void main(String[] args) {
   Box myBox1 = new Box();
   Box myBox2 = new Box();

   myBox1.width = 10;
   myBox1.height = 20;
   myBox1.depth = 15;

   myBox2.width = 3;
   myBox2.height = 6;
   myBox2.depth = 9;

   printf("V = %f", myBox1.volume());
   printf("V = %f", myBox2.volume());
}}
```

Вывод:

```
V = 3000.0
V = 162.0
```

Методы чтения (Accessors, Getter)

Метод чтения – метод, который предоставляет информацию о состоянии объекта.

```
class Box {
  double width;
  double height;
  double depth;

  double getWidth() {
    return width;
  }
}
```

Метод чтения свойства **width**

Методы-модификаторы (Mutators, Setter)

Метод-модификатор – метод, который изменяет внутреннее состояние объекта.

```
class Box {
  double width;
  double height;
  double depth;

  double setWidth(double value) {
    if (value > 0)
       width = value;
  }
}
```

Метод-модификатор свойства **width**

Метод с параметрами

```
class Box {
  double width;
  double height;
  double depth;

  double volume() {
    return width * hei
  }

  void setDim(double w, double h, double d) {
   width = w;
   height = h;
   depth = d;
  }
}
```

Метод **setDim()** устанавливает размеры параллелепипеда

Вызов метода setDim – 1

```
class BoxDemo {
  public static void main(String[] args) {
   Box myBox1 = new Box();
   Box myBox2 = new Box();

   myBox1.setDim(10, 20, 15);

   myBox2.setDim(3, 6, 9);

   printf("V = %f", myBox1.volume());
   printf("V = %f", myBox2.volume());
  }
}
```

? Что плохо?

Вызов метода setDim – 2

Было бы проще и удобнее, если бы все действия по установке переменных выполнялись при создании объекта.

Такая автоматическая инициализация при создании осуществляется с помощью **конструктора**.

Программирование на языке Java

Тема 34. Конструкторы

Конструктор класса

Конструктор класса – специальный блок инструкций, вызываемый при создании объекта.

Конструктор класса – инициализирует объект непосредственно во время создания.

Если конструктор класса не определен явно, Java создает для этого класса **конструктор**, который будет использоваться **по умолчанию**.

Конструктор, используемый **по умолчанию**, инициализирует все переменные экземпляра нулевыми значениями.

Особенности конструктора класса

1. Имя конструктора совпадаем с именем класса, в котором он находится.
2. Конструкторы не имеют четко определенного типа возвращаемых данных, даже типа **`void`**.
3. Конструктор автоматически вызывается непосредственно после создания объекта, перед завершением выполнения операции **`new`**.
4. Конструкторы не могут напрямую вызываться (необходимо использовать ключевое слово **`new`**).

Пример конструктора класса Box – 1

```
class Box {
  double width;
  double height;
  double depth;

  double volume() {
    return width * height * depth;
  }
  Box() {
    println("Конструирование объекта Box");
    width = 10;
    height = 10;
    depth = 10;
  }
}
```

Конструктор класса **Box**

Внимание! Вывод текста только для демонстрации работы конструктора.

Пример конструктора класса Box – 2

```
class BoxDemo {
  public static void main(String[] args) {
    Box myBox1 = new Box();

    Box myBox2 = new Box();

    printf("V = %f", myBox1.volume());
    printf("V = %f", myBox2.volume());
  }
}
```

Вывод:

```
V = 1000.0
V = 1000.0
```

? Что плохо?

Конструкторы с параметрами

Конструктор с параметрами – способ конструирования объектов с различными значениями переменных экземпляра.

Пример конструктора с параметрами – 1

```
class Box {
  double width;
  double height;
  double depth;

  double volume() {
    return width * hei
  }

  Box(double w, double h, double d) {
      width = w;
      height = h;
      depth = d;
    }
}
```

Конструктор с параметрами класса **Box**

Пример конструктора с параметрами – 2

```
class BoxDemo {
  public static void main(String[] args) {
    Box myBox1 = new Box(10, 20,
    15);

    Box myBox2 = new Box(3, 6, 9);

    printf("V = %f", myBox1.volume());
    printf("V = %f", myBox2.volume());
  }
}
```

Вывод:

```
V = 3000.0
V = 162.0
```

Ключевое слово this

Ключевое слово this используется в методах класса для ссылки на вызвавший метод объект.

Ключевое слово this может использоваться внутри любого метода для ссылки на текущий объект.

Версия конструктора класса Box:

```
Box(double w, double h, double d) {
    wi this.width = w;
    he this.height = h;
       this.depth = d;
  }
```

Сокрытие переменной экземпляра

В Java не допускается объявле
переменных с одним и тем же

Ссылки на локальные переменные

Если имя локальной переменной совпадает с именем переменной экземпляра, то локальная переменная скрывает переменную экземпляра.

```
Box(double width, double height, double depth)
  {
  wi this.width = width;
  he this.height = height;
     this.depth = depth;
  }
```

Ссылки
не на пе
но
а

Ссылки на переменные экземпляра класса

Печать объекта

По-умолчанию Java не знает как напечатать объект.

```
class BoxDemo {
  public static void main(String[] args) {
   Box myBox1 = new Box(10, 20, 30);
   println(myBox1);
  }
}
```

```
Box@9e8c34
```

Можно самостоятельно вывести состояние объекта

```
printf("(%f, %f, %f)", myBox1.width,
               myBox1.height,
               myBox1.depth);
```

? Что плохо?

Метод `toString`

Метод **`toString`** предназначен для описания преобразования из объека в строку

Вызывается, когда происходит попытка печати объекта или конкатенация со строкой.

В каждом классе есть метод toString, даже если его нет в исходном коде класса. По умолчанию он возвращает имя класса и число в 16-ой с/с

Синтаксис `toString`

По-умолчанию Java не знает как напечатать объект.

```
public String toString() {
   // код, возвращающий строку
}
```