Язык программирования Java

Мультимедийный курс

автор: Васильев А.Н.

www.vasilev.kiev.ua

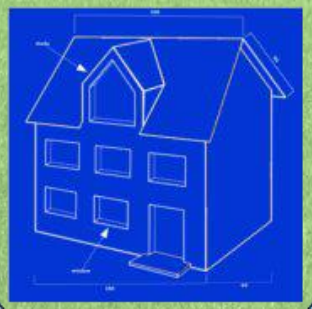
Киев 2017

Реализация ООП в Java

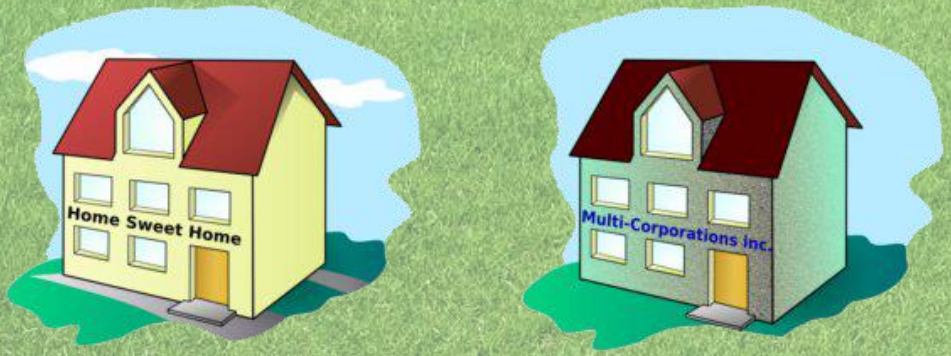
- Классы и объекты в Java
- Поля и методы
- Создание объектов
- Перегрузка методов
- Спецификаторы доступа
- Статические члены класса
- Конструкторы
- Использование объектов

Классы и объекты

- В ООП программа - набор взаимодействующих объектов.
"Место действия" - главный метод программы
- Объект создается на основе класса



Класс



Объекты

Класс с объектом соотносится так, как план здания соотносится с реальным зданием, построенным в соответствии чертежом

Описание класса

- Описание класса начинается с ключевого слова `class`
- После ключевого слова `class` указывают имя класса
- Тело класса описывается в фигурных скобках
- Тело класса содержит описание полей и методов. Поля и методы класса называются членами класса

Схема описания класса:

```
class Имя_класса {
    // Поля и методы класса
}
```

Поля

Поле - переменная, связанная с классом

- При создании объекта класса он получает переменную, соответствующую полю, описанном у в классе
- При описании поля указывается тип поля и его имя
- Несколько полей одного типа могут указываться через запятую

Например:

```
class MyClass{
    int number;
    char A,B;
}
```

Класс `MyClass` с тремя полями: поле `number` типа `int` и поля `A` и `B` типа `char`

Методы

- Метод - именованный блок кода, который можно выполнить, вызвав метод
- Метод может возвращать результат
- При описании метода указывают идентификатор типа результата, имя метода, список аргументов и код метода (в фигурных скобках)
- Если метод не возвращает результат, используют идентификатор типа результата `void`
- Инструкция `return` завершает выполнение метода. Если после инструкции указано значение, оно возвращается как результат метода

Примеры описания методов:

```
void show(){
   System.out.println("Hello!");
}
```

```
int getCode(char s){
   return (int)s;
}
```

Статические и нестатические члены класса и уровни доступа

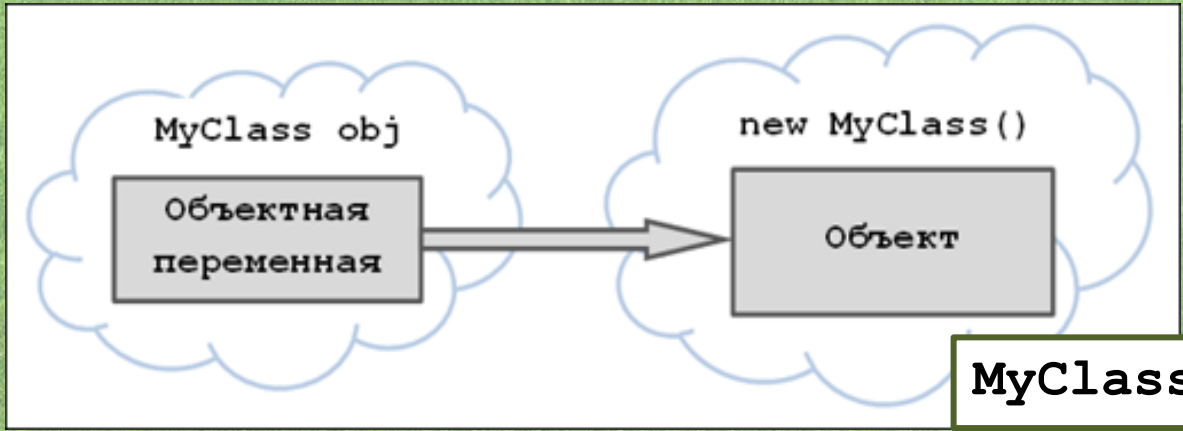
- Члены класса могут быть статическими и обычными (не статическими). Статические члены класса описываются с ключевым словом `static`
- Нестатические члены класса "привязаны" к объекту. Нет смысла говорить о нестатическом поле или методе без указания объекта
- Статические члены являются общими для всех объектов класса. Статические поля и методы существуют вне зависимости от наличия/отсутствия объектов класса

- Члены класса могут открытыми, закрытыми и защищенными
- По умолчанию члены класса открытые. Они доступны в программном коде как внутри класса, так и вне тела класса. Открытые члены могут описываться с ключевым словом `public`
- Закрытые члены класса описываются с ключевым словом `private`. Закрытые члены класса доступны в программном коде внутри класса

Создание объекта

Процесс создания объекта можно условно разделить на два этапа:

- Объявление объектной переменной, через которую будет осуществляться доступ к объекту
- Собственно создание объекта и "связывание" объекта с объектной переменной



Пример:

```
MyClass obj;
obj=new MyClass();
```

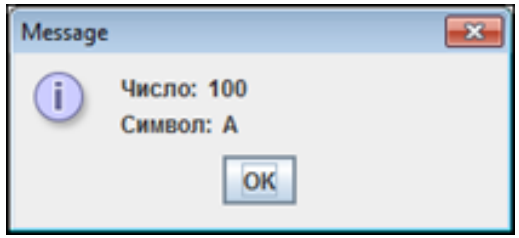
```
MyClass obj=new MyClass();
```

Значение объектной переменной - ссылка на объект (адрес объекта). При объявлении объектной переменной под нее выделяется место в памяти. Но туда, в переменную, следует еще записать ссылку. Ссылку получаем, когда создаем объект.

Использование объектов

Класс с полями

```
import javax.swing.JOptionPane;
// Описание класса:
class MyClass{
    // Поля класса:
    int number;
    char symbol;
}
// Описание класса с главным методом программы:
class UsingObjectDemo{
    // Главный метод программы:
    public static void main(String[] args){
        // Создание объекта:
        MyClass obj=new MyClass();
        // Присваивание значений полям объекта:
        obj.number=100;
        obj.symbol='A';
        // Текст для отображения в диалоговом окне:
        String text="Число:  "+obj.number+"\n";
        text+="Символ:  "+obj.symbol;
        // Отображение диалогового окна:
        JOptionPane.showMessageDialog(null,text);
    }
}
```



Использование объектов

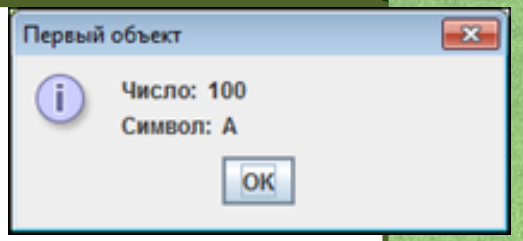
Класс с методами

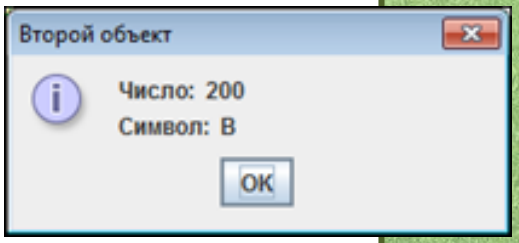
```
import javax.swing.JOptionPane;
// Описание класса:
class MyClass{
   // Поля класса:
   int number;
   char symbol;
   // Метод для присваивания значений полям:
   void set(int n,char s){
      number=n;
      symbol=s;
   }
   // Методом возвращается текстовая строка
   // с описанием объекта:
   String getInfo(){
      // Текст, который возвращается
      // результатом метода:
      String text="Число:  "+number+"\n";
      text+="Символ:  "+symbol;
      // Результат метода:
      return text;
   }
}
// Продолжение на следующем слайде!!!
```

Использование объектов

Класс с методами (продолжение)

```
class UsingObjectWithMethodsDemo{
    // Главный метод программы:
    public static void main(String[] args){
        // Создание первого объекта:
        MyClass objA=new MyClass();
        // Создание второго объекта:
        MyClass objB=new MyClass();
        // Присваивание значений полям первого объекта:
        objA.set(100,'A');
        // Присваивание значений полям второго объекта:
        objB.set(200,'B');
        // Отображение первого диалогового окна:
        JOptionPane.showMessageDialog(null,
            objA.getInfo(),  // Отображаемый текст
            "Первый объект", // Заголовок окна
            JOptionPane.INFORMATION_MESSAGE); // Тип окна
        // Отображение второго диалогового окна:
        JOptionPane.showMessageDialog(null,
            objB.getInfo(),  // Отображаемый текст
            "Второй объект", // Заголовок окна
            JOptionPane.INFORMATION_MESSAGE); // Тип окна
    }
}
```





Домашнее задание

- Напишите программу с классом, в котором есть два поля (текстовое и символьное), а также методы для присваивания значений полям и метод для отображения значений полей в консольном окне

Перегрузка методов

- В Java в одном и том же классе можно описывать несколько методов с одинаковыми названиями. Различаются такие методы типом аргументов и/или их количеством. Такой механизм называется перегрузкой методов

- Перегрузка методов - очень мощный инструмент, позволяющий создавать эффективные и гибкие программные коды

- Создавая методы с одинаковыми названиями, мы создаем реально разные методы

- При вызове метода с данным названием решение о том, какой из методов на самом деле вызывается, принимается исходя из контекста команды вызова метода например, по количеству переданных методу аргументов или их типу. Вместе с тем, с формальной точки зрения, при вызове таких методов создается иллюзия, что вызывается один и тот же метод, но с разными аргументами. В этом смысле можно говорить о разных версиях одного метода.

Перегрузка методов

Использование перегрузки методов

```
class MyClass{
  int number;
  char symbol;
  // Метод с одним аргументом для присваивания значения полю number:
  void set(int n){
    number=n;
  }
  // Метод с одним аргументом для присваивания значения полю symbol:
  void set(char s){
    symbol=s;
  }
  // Метод с двумя аргументами для присваивания значений полям
  // number и symbol:
  void set(int n,char s){
    set(n); // Присваивание значения полю number
    set(s); // Присваивание значения полю symbol
  }
  // Метод без аргументов для присваивания значений обеим полям:
  void set(){
    // Присваивание значения 0 полю number
    // и значения 'Z' полю symbol:
    set(0,'Z');
  } // Продолжение на следующем слайде!!!
```

Перегрузка методов

Использование перегрузки методов (продолжение)

```
  // Метод без аргументов для отображения
  // значений полей объекта:
  void show(){
    System.out.println("Значения полей "+number+" и "+symbol);
  }
  // Метод с одним аргументом для отображения
  // значений полей объекта:
  void show(String txt){
    System.out.println(txt+": значения полей "+number+" и "+symbol);
  }
  // Метод с двумя аргументами для отображения
  // значений полей объекта:
  void show(String txt1,String txt2){
    System.out.println(txt1+": "+number);
    System.out.println(txt2+": "+symbol);
  }
} // Завершение описания класса MyClass!!!


// Продолжение на следующем слайде!!!
```

Перегрузка методов

Использование перегрузки методов (продолжение)

```
class MethodOverloadingDemo{
  public static void main(String[] args){
    // Объявление объектных переменных и создание объектов:
    MyClass objA,objB;
    objA=new MyClass();
    objB=new MyClass();
    // Присваивание значений полям первого объекта:
    objA.set(100);
    objA.set('A');
    // Отображение значений полей первого объекта:
    System.out.println("Объект objA:");
    objA.show();
    // Присваивание значений полям второго объекта:
    objB.set();
    // Отображение значений полей второго объекта:
    objB.show("Объект objB");
    // Изменение значения полей второго объекта:
    objB.set(200,'B');
    // Проверка значений полей второго объекта:
    System.out.println("Объект objB после изменения:");
    objB.show("Число","Символ");
  }
}
```

Перегрузка методов

Использование перегрузки методов (результат)

```
Объект objA:
Значения полей 100 и A
Объект objB: значения полей 0 и Z
Объект objB после изменения:
Число: 200
Символ: B
```

Домашнее задание

- Напишите программу с классом, в котором есть символьное поле. Класс должен содержать описание перегруженного метода для присваивания значения полю: без аргументов, с символьным аргументом, и с текстовым аргументом. В первом случае полю присваивается значением символ 'A', во втором - значение, переданное аргументом, в третьем - первая буква в тексте, переданном аргументом методу

Конструктор

- Конструктором называется метод, вызываемый автоматически при создании объекта класса. В конструкторе определяются дополнительные (помимо выделения памяти под объект) действия, которые следует выполнить при создании объекта

- Если конструктор в классе явно не описан, то используется так называемый конструктор по умолчанию. Действие конструктора по умолчанию состоит в том, что никаких дополнительных действий не выполняется

Правила описания конструктора:

- Имя конструктора совпадает с именем класса

- Конструктор не возвращает результат, но при этом ключевое слово `void` в сигнатуре конструктора не указывается

- У конструктора могут быть аргументы и конструктор можно перегружать другими словами, в классе может быть описано несколько конструкторов.

Конструктор

Если конструктор описан с аргументами, то при создании объекта их нужно передать конструктору:

в инструкции создания объекта после ключевого слова `new` и имени класса в круглых скобках указываются аргументы конструктора

Создание объекта с передачей аргументов конструктору:

```
Имя_класса переменная=new Конструктор(аргументы);
```

Конструктор

Использование конструкторов

```
// Описание класса:
class MyClass{
    // Поля класса:
    int number;
    char symbol;
    // Конструктор класса без аргументов:
    MyClass(){
        // Присваивание значений полям:
        number=100;
        symbol='A';
    }
    // Конструктор класса с двумя аргументами:
    MyClass(int n,char s){
        // Присваивание значений полям:
        number=n;
        symbol=s;
    }
    // Метод для отображения значений полей объекта:
    void show(){
        System.out.println("Значения полей "+number+" и "+symbol);
    }
}
// Продолжение на следующем слайде!!!
```

Конструктор

Использование конструкторов (продолжение)

```
// Описание класса с главным методом программы:
class UsingConstructorDemo{
   // Главный метод программы:
   public static void main(String[] args){
      // Создание первого объекта
      // (вызывается конструктор без аргументов):
      MyClass objA=new MyClass();
      // Создание второго объекта
      // (вызывается конструктор с двумя аргументами):
      MyClass objB=new MyClass(200,'B');
      // Отображение значений полей первого объекта:
      System.out.println("Объект objA:");
      objA.show();
      // Отображение значений полей второго объекта:
      System.out.println("Объект objB:");
      objB.show();
   }
}
```

Конструктор

Использование конструкторов (результат)

```
Объект objA:
Значения полей 100 и A
Объект objB:
Значения полей 200 и B
```

Домашнее задание

- Напишите программу с классом, в котором есть два символьных поля, а также конструкторы, позволяющие создавать объекты на основе таких значений:

а) один символ (значения полей);
б) два символа (значения полей);
в) текст (значения полей - первая и последняя буква в тексте)

Статические поля и методы

- Статический член класса "общий" для всех объектов этого класса и существует независимо от наличия или отсутствия в программе объектов класса
- Обращаться к статическому члену можно не только через объект класса, но и непосредственно через класс (последний способ является предпочтительным)
- При описании статических членов (полей и методов) используют ключевое слово `static`
- В описании класса статические поля можно инициализировать - присвоить значение непосредственно в классе
- При обращении к статическому полю указывают имя класса и через точку - имя статического поля
- При вызове статического метода указываем имя класса и, через точку, имя метода с аргументами (или без) в круглых скобках
- Статические поля играют роль глобальных переменных
- Статические методы служат "заменителем" глобальных функций

Статические поля и методы

Статические поля и методы

```
// Класс со статическими членами:
class MyClass{
   // Статическое поле:
   static int count=0;
   // Конструктор без аргументов:
   MyClass(){
      // Увеличение значения статического поля:
      count++;
      // Отображение сообщения:
      System.out.println("Создан объект номер "+count);
   }
   // Статический метод:
   static void show(){
      System.out.println("Количество объектов: "+count);
   }
}
// Продолжение на следующем слайде!!!
```

Статические поля и методы

Статические поля и методы (продолжение)

```
// Класс с главным методом программы:
class UsingStaticMembersDemo{
   // Главный метод программы:
   public static void main(String[] args){
      // Вызов статического метода:
      MyClass.show();
      // Создание объектов:
      MyClass objA=new MyClass();
      MyClass objB=new MyClass();
      MyClass objC=new MyClass();
      // Вызов статического метода через ссылку на класс:
      MyClass.show();
      // Вызов статического метода через ссылку на объект:
      objC.show();
      objB.show();
   }
}
```

Статические поля и методы

Статические поля и методы (результат)

```
Количество объектов: 0
Создан объект номер 1
Создан объект номер 2
Создан объект номер 3
Количество объектов: 3
Количество объектов: 3
Количество объектов: 3
```

Закрытые и открытые члены класса

- Открытые члены класса могут описываться без указания идентификатора уровня доступа или с указанием идентификатора уровня доступа `public`. И в том, и в другом случае, член класса будет открытым и доступным вне программного кода класса
- Если член класса описан без идентификатора уровня доступа, то область его доступности ограничивается текущим пакетом (пакетом, в котором описан класс)
- Если член класса описан с ключевым словом `public`, то член класса доступен и в других пакетах

- Закрытые члены класса описываются с ключевым словом `private`
- Закрытые члены класса доступны только в программном коде в теле класса, и к ним нет доступа вне тела класса

Закрытые члены класса

Закрытые члены класса

```
// Класс с закрытыми членами:
class MyClass{
   // Закрытое статическое поле:
   private static int count=0;
   // Закрытые нестатические поля:
   private int number;
   private String name;
   // Конструктор без аргументов:
   MyClass(String n){
      // Увеличение значения статического поля:
      count++;
      // Присваивание значений нестатическим полям:
      name=n;
      number=count;
      // Отображение сообщения:
      System.out.println("Создан объект с именем "+name);
   }
   // Метод для отображения сообщения:
   public void show(){
      System.out.println("Название объекта: "+name);
      System.out.println("Номер объекта: "+number);
      System.out.println("Количество объектов: "+count);
   }// Продолжение на следующем слайде!!!
```

Закрытые члены класса

Закрытые члены класса (продолжение)

```
    // Метод для присваивания значения закрытому
    // текстовому полю:
    public void set(String n){
        name=n;
    }
} // Класс с главным методом программы:
class UsingPrivateMembersDemo{
    // Главный метод программы:
    public static void main(String[] args){
        // Создание объектов:
        MyClass objA=new MyClass("Alpha");
        MyClass objB=new MyClass("Bravo");
        MyClass objC=new MyClass("Charlie");
        // Вызов метода для каждого из объектов:
        objA.show();
        objB.show();
        objC.show();
        // Изменение поля второго объекта:
        objB.set("Второй Объект");
        // Отображение значений полей объекта:
        objB.show();
    }
}
```

Закрытые члены класса

Закрытые члены класса (результат)

```
Создан объект с именем Alpha
Создан объект с именем Bravo
Создан объект с именем Charlie
Название объекта: Alpha
Номер объекта: 1
Количество объектов: 3
Название объекта: Bravo
Номер объекта: 2
Количество объектов: 3
Название объекта: Charlie
Номер объекта: 3
Количество объектов: 3
Название объекта: Второй Объект
Номер объекта: 2
Количество объектов: 3
```

Передача аргументов методам

При передаче аргументов методам на самом деле передается техническая, автоматически создаваемая копия аргументов

Передача аргументов методам

```
class MethodArgumentsDemo{
  // Статический метод с двумя целочисленными аргументами,
  // которые "обмениваются" значениями:
  static void swap(int a,int b){
    System.out.println("Выполняется метод swap()");
    // Значения аргументов метода до
    // изменения значений:
    System.out.println("Аргументы до изменения значений: "+a+" и "+b);
    // Аргументы "обмениваются" значениями:
    int x=b;
    b=a;
    a=x;
    // Значения аргументов метода после
    // изменения значений:
    System.out.println("Аргументы после изменения значений: "+a+" и
"+b);
    System.out.println("Завершено выполнение метода swap()");
  }
  // Продолжение на следующем слайде!!!
```

Передача аргументов методам

Передача аргументов методам

```
  // Главный метод программы:
  public static void main(String[] args){
    // Целочисленные переменные:
    int m=100,n=200;
    // Значения переменных до вызова метода swap():
    System.out.println("Переменные до вызова метода swap(): "+m+" и
"+n);
    // Вызов метода swap():
    swap(m,n);
    // Значения переменных после вызова метода swap():
    System.out.println("Переменные после вызова метода swap(): "+m+"
и "+n);
  }
}
```

Передача аргументов методам (результат)

```
Переменные до вызова метода swap(): 100 и 200
Выполняется метод swap()
Аргументы до изменения значений: 100 и 200
Аргументы после изменения значений: 200 и 100
Завершено выполнение метода swap()
Переменные после вызова метода swap(): 100 и 200
```

Объект как аргумент метода

При передаче объекта аргументом методу фактически аргументом указывается объектная переменная

Передача аргументом объекта

```
class MyClass{                  // Класс для создания объектов
   int number;                  // Целочисленное поле
   MyClass(int n){              // Конструктор
      number=n;
   }
} // Класс со статическим методом swap() и главным методом программы:
class SwapFieldsDemo{
   // Статический метод с двумя аргументами, являющимися
   // объектами класса MyClass:
   static void swap(MyClass A,MyClass B){
      System.out.println("Выполняется метод swap()");
      // Значения поля number объектов, переданных
      // аргументами методу:
      System.out.println("Объект A: "+A.number);
      System.out.println("Объект B: "+B.number);
      // Объекты "обмениваются" значениями полей:
      int number=B.number;
      B.number=A.number;
      A.number=number;
      // Продолжение на следующем слайде!!!
```

Объект как аргумент метода

Передача аргументом объекта (продолжение)

```
        // Значения поля number объектов, переданных
        // аргументами методу swap()
        // (после "обмена" значениями полей):
        System.out.println("Значения полей изменены");
        System.out.println("Объект A: "+A.number);
        System.out.println("Объект B: "+B.number);
        System.out.println("Завершено выполнение метода swap()");
    } // Главный метод программы:
    public static void main(String[] args){
        // Создание объектов:
        MyClass A=new MyClass(100);
        MyClass B=new MyClass(200);
        // Значения поля number объектов A и B до
        // вызова метода swap():
        System.out.println("До вызова метода swap():
A.number="+A.number+" и B.number="+B.number);
        // Вызов метода swap():
        swap(A,B);
        // Значения поля number объектов A и B после вызова swap():
        System.out.println("После вызова метода swap():
A.number="+A.number+" и B.number="+B.number);
    }
}
```

Объект как аргумент метода

Передача аргументом объекта (результат)

```
До вызова метода swap(): A.number=100 и B.number=200
Выполняется метод swap()
Объект A: 100
Объект B: 200
Значения полей изменены
Объект A: 200
Объект B: 100
Завершено выполнение метода swap()
После вызова метода swap(): A.number=200 и B.number=100
```

Объект как аргумент метода

Передача аргументом объекта - 2

```
class MyClass{
   int number;
   MyClass(int n){
      number=n;
   }
}
class SwapObjectsDemo{
   static void swap(MyClass A,MyClass B){
      System.out.println("Выполняется метод swap()");
      System.out.println("Объект A: "+A.number);
      System.out.println("Объект B: "+B.number);
      // Аргументы "обмениваются" значениями:
      MyClass X=B;
      B=A;
      A=X;
      System.out.println("Значения аргументов изменены");
      System.out.println("Объект A: "+A.number);
      System.out.println("Объект B: "+B.number);
      System.out.println("Завершено выполнение метода swap()");
   }
   public static void main(String[] args){
      MyClass A=new MyClass(100);
      MyClass B=new MyClass(200);
      System.out.println("До вызова метода swap(): A.number="+A.number+" и
B.number="+B.number);
      swap(A,B);
      System.out.println("После вызова метода swap(): A.number="+A.number+" и
B.number="+B.number);
   }
}
```

Объект как аргумент метода

Передача аргументом объекта - 2 (результат)

```
До вызова метода swap(): A.number=100 и B.number=200
Выполняется метод swap()
Объект A: 100
Объект B: 200
Значения аргументов изменены
Объект A: 200
Объект B: 100
Завершено выполнение метода swap()
После вызова метода swap(): A.number=100 и B.number=200
```

Домашнее задание

- Напишите программу с классом, в котором есть два поля (символьное и целочисленное) и метод, аргументом которому передается объект того же класса. При вызове метода объекту, из которого вызываются метод, присваиваются значения полей объекта, переданного аргументом методу. У объекта, переданного аргументом методу, значения полей увеличиваются на единицу

Домашнее задание

- Напишите программу с двумя классами. У первого класса есть целочисленное поле. У второго класса есть символьное поле.

В первом классе есть метод, аргументом которому передается объект второго класса. При вызове этого метода полю объекта, из которого вызывается метод, значением присваивается код символа, являющегося значением поля объекта, переданного аргументом методу.

У второго класса есть метод, аргументом которому передается объект первого класса. Значением полю объекта, из которого вызывается метод, присваивается символ с кодом, определяемым значением поля объекта, переданного аргументом методу

Объект как результат метода

Если метод возвращает результатом объект, то формально это объектная переменная со ссылкой на объект

Объект как результат метода

```
// Класс:
class MyClass{
   // Закрытые поля:
   private int code;
   private String name;
   // Конструктор:
   MyClass(int n,String s){
      System.out.println("Создание объекта:");
      // Присваивание значений полям и
      // отображение этих значений:
      set(n,s).show();
   }
   // Метод для присваивания значения целочисленному полю,
   // возвращающий результатом ссылку на объект:
   MyClass set(int n){
      // Присваивание значения целочисленному полю:
      code=n;
      // Результат метода:
      return this;
   } Продолжение на следующем слайде!!!
```

Объект как результат метода

Объект как результат метода (продолжение)

```
    // Метод для присваивания значения текстовому полю,
    // возвращающий результатом ссылку на объект:
    MyClass set(String s){
        // Присваивание значения текстовому полю:
        name=s;
        // Результат метода:
        return this;
    }
    // Метод для присваивания значений полям,
    // возвращающий результатом ссылку на объект:
    MyClass set(int n,String s){
        // Присваивание значений полям и результат метода:
        return set(n).set(s);
    }
    // Метод для отображения значений полей:
    void show(){
        System.out.println("Поле code="+code);
        System.out.println("Поле name="+name);
        System.out.println("-----------------");
    }
}
// Продолжение на следующем слайде!!!
```

Объект как результат метода

Объект как результат метода (продолжение)

```
class ObjectAsResultDemo{
   // Статический метод для создания объекта:
   static MyClass createObject(int n,String s){
      // Результат метода:
      return new MyClass(n,s);
   }
   // Главный метод программы:
   public static void main(String[] args){
      // Создание объекта:
      MyClass obj=createObject(100,"alpha");
      // Изменение значения целочисленного поля и
      // отображение значений полей объекта:
      obj.set(200).show();
      // Изменение значения текстового поля и
      // отображение значений полей объекта:
      obj.set("bravo").show();
      // Отображение значений полей:
      obj.show();
      // Создание объекта, изменение значений его полей
      // и вызов метода для отображения значений полей:
      createObject(300,"charlie").set(400,"delta").show();
   }
}
```

Объект как результат метода

Объект как результат метода (результат)

```
Создание объекта:
Поле code=100
Поле name=alpha
-----------------
Поле code=200
Поле name=alpha
-----------------
Поле code=200
Поле name=bravo
-----------------
Поле code=200
Поле name=bravo
-----------------
Создание объекта:
Поле code=300
Поле name=charlie
-----------------
Поле code=400
Поле name=delta
-----------------
```

Массив как аргумент и результат метода

- **При передаче массива аргументом методу в метод передается переменная массива**
- **При возвращении массива результатом метода из метода возвращается переменная массива со ссылкой на массив**

Детали

Матрица представляет собой таблицу из чисел. Каждый элемент матрицы определяется парой индексов - то есть все как в двумерном массиве. Массив представляет собой набор элементов, каждый из которых определяется индексом. Векторы мы будем отождествлять с одномерным массивом. Предположим, что задана некоторая матрица A с элементами a_{ij}, индексы которых $1 \le i \le m$ и $1 \le j \le n$. Пускай вектор B состоит из элементов b_j (индекс $j = 1,2,\dots,n$). Тогда результатом произведения матрицы A на вектор B является вектор $C = AB$, элементы c_i (индекс $i = 1,2,\dots,m$) которого вычисляется в виде суммы $c_i = \sum_{j=1}^{n} a_{ij} b_j$. Стоит заметить, что количество столбцов в матрице A должно совпадать с количеством элементов в векторе B.

Домашнее задание

- Напишите программу с классом, в котором есть символьное поле. В главном классе опишите статический метод, который на основании символьного аргумента создает объект соответствующего класса

Домашнее задание

- Напишите программу с классом, в котором есть целочисленное поле. В классе опишите перегруженный нестатический метод, который позволяет создавать копию объекта, из которого метод вызывается (без аргументов). Если метод вызывается с целочисленным аргументом, то переданное аргументом значение присваивается полю объекта, из которого вызывается метод. Значение поля создаваемого объекта определяется "старым" значением поля объекта, из которого вызывается метод

Массив как аргумент и результат метода

```
class ArraysAndMethodsDemo{
  // Метод для отображения содержимого одномерного массива:
  static void show1D(int[] nums){
    // Оператор цикла по коллекции:
    for(int s: nums){
      // Форматированный вывод числового значения:
      System.out.printf("%4d",s);
    }
    // Переход к новой строке:
    System.out.println("");
  }
  // Метод для отображения содержимого двумерного массива:
  static void show2D(int[][] nums){
    // Внешний оператор цикла по коллекции:
    for(int[] p: nums){
      // Внутренний оператор цикла по коллекции:
      for(int s: p){
        // Форматированный вывод числового значения:
        System.out.printf("%4d",s);
       }
       System.out.println(""); // Переход к новой строке
     }
   } // Продолжение на следующем слайде!!!
```

Массив как аргумент и результат метода (продолжение)

```
// Метод для вычисления произведения матрицы и вектора.
// Аргументами методу передаются двумерный
// и одномерный массивы. Результатом возвращается
// одномерный массив:
static int[] prod(int[][] A,int[] B){
  // Создание одномерного массива:
  int[] C=new int[A.length];
  // Вычисление значений элементов массива - результата
  // произведения матрицы и вектора:
  for(int i=0;i<C.length;i++){
    // Начальное нулевое значение элемента:
    C[i]=0;
    for(int j=0;j<B.length;j++){
      // Добавление очередного слагаемого:
      C[i]+=A[i][j]*B[j];
    }
  }
  // Результат метода:
  return C;
}
// Продолжение на следующем слайде!!!
```

Массив как аргумент и результат метода

Массив как аргумент и результат метода (продолжение)

```
  // Главный метод программы:
  public static void main(String[] args){
    // Двумерный массив (матрица):
    int[][] A={{1,0,3,-1},{2,-1,-2,3},{-1,1,0,2}};
    // Одномерный массив (вектор):
    int[] B={1,-1,3,2};
    // Результат произведения матрицы на вектор:
    int[] C=prod(A,B);
    // Отображение содержимого матрицы:
    System.out.println("Матрица A:");
    show2D(A);
    // Отображение содержимого вектора:
    System.out.println("Вектор B:");
    show1D(B);
    // Отображение результат произведения
    // матрицы на вектор:
    System.out.println("Вектор C=AB:");
    show1D(C);
  }
}
```

Массив как аргумент и результат метода

Массив как аргумент и результат метода (результат)

```
Матрица A:
   1   0   3  -1
   2  -1  -2   3
  -1   1   0   2
Вектор B:
   1  -1   3   2
Вектор C=AB:
   8   3   2
```

Массив как поле класса

Полем класса является переменная массива. Сам массив обычно создается в конструкторе или при вызове методов

По определению биномиальные коэффициенты $C_n^k = \frac{n!}{k!(n-k)!}$. При вычислении коэффициентов мы исходим из того, что $C_n^0 = 1$ и для всех $k = 1,2,3,\dots,n$ имеет место соотношение $C_n^k = C_n^{k-1} \cdot \frac{n-k+1}{k}$.

Массив как поле класса

```
class Binomial{
   // Закрытое поле-массив:
   private int[] binoms;
   // Конструктор:
   Binomial(int n){
      // Создание массива:
      binoms=new int[n+1];
      // Значение начального элемента массива:
      binoms[0]=1;
      // Заполнение массива:
      for(int k=1;k<=n;k++){
         binoms[k]=binoms[k-1]*(n-k+1)/k;
      }
   } // Продолжение на следующем слайде!!!
```

Массив как поле класса

Массив как поле класса (продолжение)

```
    // Переопределение метода toString():
    public String toString(){
        // Текстовая переменная для формирования результата:
        String txt="| ";
        // Добавление к тексту значений элементов массива:
        for(int k=0;k<binoms.length;k++){
            txt+=binoms[k]+" | ";
        }
        // Результат метода:
        return txt;
    }
}
// Главный класс:
class ArrayAsFieldDemo{
    public static void main(String[] args){
        // Создание объектов:
        Binomial A=new Binomial(5);
        Binomial B=new Binomial(10);
        // Отображение биномиальных коэффициентов:
        System.out.println(A);
        System.out.println(B);
    }
}
```

Массив как поле класса

Массив как поле класса (результат)

```
  | 1 | 5 | 10 | 10 | 5 | 1 |

  | 1 | 10 | 45 | 120 | 210 | 252 | 210 | 120 | 45 | 10 | 1 |
```

Домашнее задание

- Напишите программу с классом, в котором есть поле - целочисленный массив. Опишите перегруженный статический метод, который на основании объекта данного класса возвращает копию массива (из данного объекта), а на основании переданного аргументом массива создает объект соответствующего класса

Массив объектов

При создании массива объектов создается массив объектных переменных, и каждой из переменных значением присваивается ссылка на объект

Массив объектов

```
// Класс с целочисленным полем:
class MyClass{
   // Закрытое целочисленное поле:
   private int number;
   // Конструктор:
   MyClass(int n){
      number=n;
   }
   // Метод для считывания значения поля:
   int get(){
      return number;
   }
}
// Продолжение на следующем слайде!!!
```

Массив объектов

Массив объектов (продолжение)

```
// Главный класс:
class ArrayOfObjectsDemo{
   // Статический метод для создания массива объектов:
   static MyClass[] createBinoms(int n){
      // Создается массив из объектных переменных:
      MyClass[] bins=new MyClass[n+1];
      // Создание объекта, ссылка на который записывается
      // в начальный элемент массива:
      bins[0]=new MyClass(1);
      // Создание объектов и заполнение массива:
      for(int k=1;k<=n;k++){
         // Создается новый объект и ссылка на него
         // присваивается значением элементу массива:
         bins[k]=new MyClass(bins[k-1].get()*(n-k+1)/k);
      }
      // Результат метода:
      return bins;
   }
   // Продолжение на следующем слайде!!!
```

Массив объектов

Массив объектов (продолжение)

```
    // Статический метод для отображения значений полей
    // объектов, формирующих массив:
    static void show(MyClass[] objs){
        // Начальное значение текстовой переменной:
        String txt="| ";
        // В текст дописываются значения полей объектов,
        // которые формируют массив, переданный
        // аргументом методу:
        for(int k=0;k<objs.length;k++){
            txt+=objs[k].get()+" | ";
        }
        System.out.println(txt);
    }
    // Главный метод программы:
    public static void main(String[] args){
        // Создание массивов из объектов:
        MyClass[] A=createBinoms(5);
        MyClass[] B=createBinoms(10);
        // Отображение значений полей объектов из массивов:
        show(A);
        show(B);
    }
}
```

Массив объектов

Массив объектов (результат)

```
  | 1 | 5 | 10 | 10 | 5 | 1 |

  | 1 | 10 | 45 | 120 | 210 | 252 | 210 | 120 | 45 | 10 | 1 |
```

Домашнее задание

- Напишите программу с классом, в котором есть целочисленное поле. Создайте массив таких объектов. Поля объектов заполните случайными числами. Опишите статический метод, который в массиве объектов находит объект с наибольшим значением поля и возвращает результатом текст с указанием индекса объекта и значения его поля

Цепочка объектов

В цепочке объектов каждый объект содержит ссылку на следующий объект. Такая ссылка реализуется через поле, которое является объектной переменной

Цепочка объектов

```
// Класс для создания объектов в цепочке:
class MyClass{
   // Целочисленное поле:
   int number;
   // Ссылка на следующий объект в цепочке:
   MyClass next;
}
// Главный класс:
class ListOfObjectsDemo{
   // Статический метод для создания цепочки объектов:
   static MyClass createList(int n){
      // Создание первого объекта:
      MyClass obj=new MyClass();
      // Целочисленное поле первого объекта:
      obj.number=1;
      // Ссылка на последний (и пока что единственный)
      // объект в цепочке:
      MyClass t=obj;
      // Продолжение на следующем слайде!!!
```

Цепочка объектов

Цепочка объектов (продолжение)

```
        for(int k=1;k<=n;k++){    // Создание цепочки объектов
            // Создается новый объект и ссылка на него
            // записывается в поле next последнего (на
            // данный момент) объекта в цепочке:
            t.next=new MyClass();
            // Вычисление значения числового поля вновь
            // созданного объекта:
            t.next.number=t.number*(n-k+1)/k;
            // Созданный объект становится последним объектом
            // в цепочке объектов:
            t=t.next;
        }
        // Пустая ссылка для поля next последнего объекта в цепочке:
        t.next=null;
        // Результат метода - ссылка на первый объект в цепочке:
        return obj;
    }
    // Метод для отображения значений полей объектов из цепочки:
    static void showList(MyClass obj){
        String txt="| "; // Начальное значение текстовой переменной
        // Ссылка на первый объект в цепочке:
        MyClass t=obj;
        // Продолжение на следующем слайде!!!
```

Цепочка объектов

Цепочка объектов (продолжение)

```
        // Добавление к тексту значений числовых полей:
        do{
            // К тексту дописывается значение числового поля
            // объекта, на который ссылается переменная t:
            txt+=t.number+" | ";
            // Переменная t указывает на следующий объект:
            t=t.next;
        }while(t!=null);
        // Отображение значений биномиальных коэффициентов:
        System.out.println(txt);
    }
    // Главный метод программы:
    public static void main(String[] args){
        // Создание цепочек объектов:
        MyClass A=createList(5);
        MyClass B=createList(10);
        // Отображение биномиальных коэффициентов:
        showList(A);
        showList(B);
    }
}
```

Цепочка объектов

Цепочка объектов (результат)

```
  | 1 | 5 | 10 | 10 | 5 | 1 |
  | 1 | 10 | 45 | 120 | 210 | 252 | 210 | 120 | 45 | 10 | 1 |
```

Домашнее задание

- Напишите программу с классом, в котором есть два поля: целочисленное и ссылка на объект того же класса. Опишите статический метод, который создает замкнутую цепочку объектов данного класса (каждый объект ссылается на предыдущий, а последний ссылается на первый). Результатом метод возвращает ссылку на первый объект в цепочке. Значения целочисленных полей объектов - ряд натуральных чисел (1, 2, 3 и так далее). Опишите статический метод, которому аргументом передается объект данного класса. Метод отображает значения целочисленных полей всех объектов в цепочке

Домашнее задание

- Проанализировать программный код примеров, представленных в презентации
- Набрать код и проверить его функциональность

Продолжение следует…