Язык программирования Java

Мультимедийный курс

автор: Васильев А.Н.

www.vasilev.kiev.ua

Киев 2017

Наследование и то, что с ним связано

- Основы наследования
- Суперкласс и создание подкласса
- Конструктор подкласса
- Пакеты и уровни доступа
- Переопределение методов
- Виртуальность методов и конструкторов
- Объект подкласса и переменная суперкласса

Наследование

- Наследование - один из базовых механизмов ООП
- При наследовании один класс создается на основе другого класса
- Класс, на основе которого создается новый класс, называется суперклассом
- Создаваемый на основе суперкласса класс называется подклассом
- Подкласс автоматически получает все открытые поля и методы из суперкласса
- У подкласса может быть только один суперкласс (запрет множественного наследования)
- Подкласс может быть суперклассом для другого класса (многоуровневое наследование)
- Класс может быть суперклассом сразу для нескольких классов

Реализация наследования

- В описании подкласса после его имени через ключевое слово `extends` указывается имя суперкласса
- В теле подкласса описываются "дополнительные" (по сравнению с суперклассом) члены

Схема описания подкласса:

```
class Подкласс extends Суперкласс{
    // Поля и методы подкласса
}
```

Например:

```
class Bravo extends Alpha{
    // Дополнительные (кроме членов класса Alpha)
    // поля и методы класса Bravo
}
```

Подкласс и суперкласс

Пример описания суперкласса и подкласса:

```
// Описание суперкласса:
class Alpha{
    // Целочисленное поле:
    int number;
    // Метод:
    void show(){
        System.out.println("Числовое поле: "+number);
    }
}
// Описание подкласса:
class Bravo extends Alpha{
    // Символьное поле:
    char symbol;
}
```

Использование наследования

Наследование класса JOptionPane

```
import javax.swing.*;
// Класс создается наследование класса JOptionPane:
class MyPane extends JOptionPane{
   // Статический метод с двумя аргументами
   // для отображения диалогового окна:
   static void showMessage(String txt,String title){
      // Вызов статического метода showMessageDialog() из
      // класса JOptionPane:
      showMessageDialog(null,txt,title,PLAIN_MESSAGE, new
ImageIcon("d:/books/pictures/giraffe.png"));
   }
   // Статический метод с одним аргументом
   // для отображения диалогового окна:
   static void showMessage(String txt){
      // Вызов версии метода с двумя аргументами:
      showMessage(txt,"Сообщение");
   }
   // Продолжение на следующем слайде!!!
```

Наследование класса JOptionPane (продолжение)

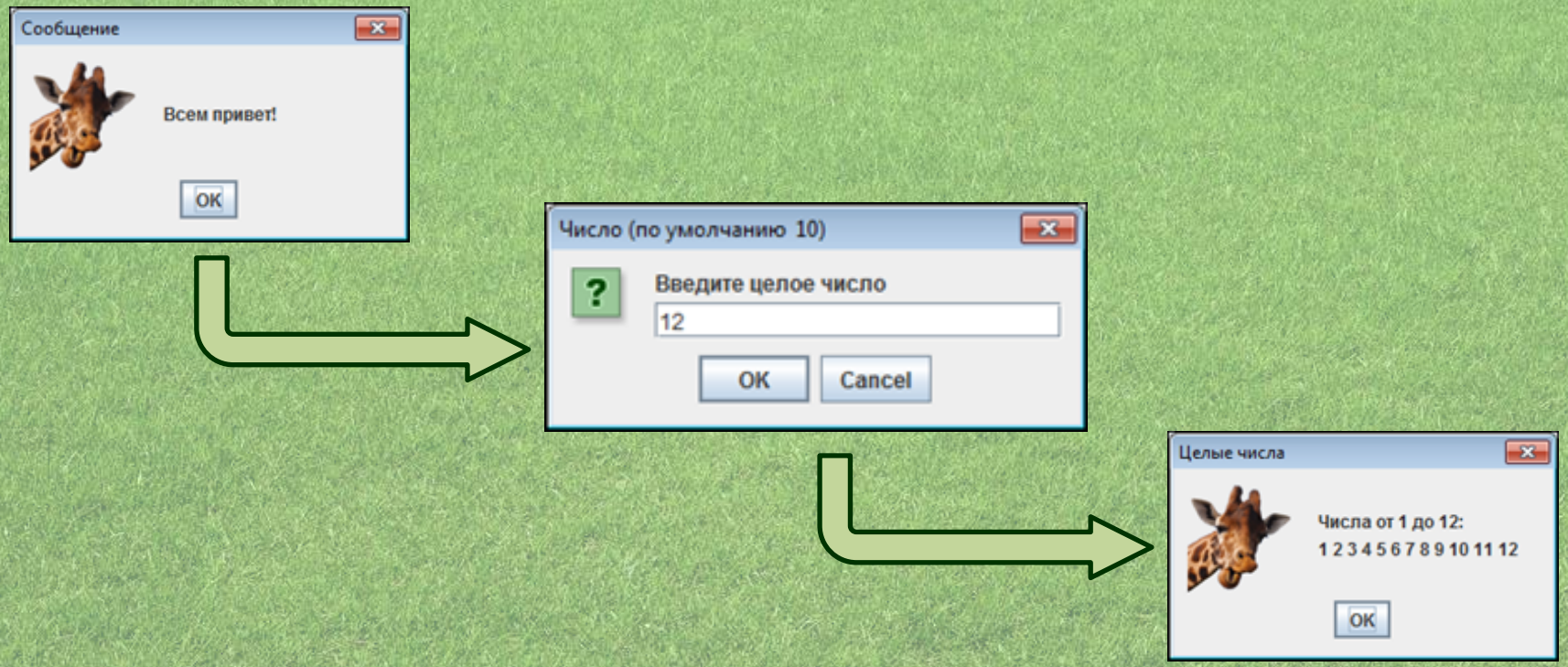
```
    // Статический метод для отображения окна с полем ввода
    // для считывания целого числа:
    static int getInteger(String txt){
        // Текстовая переменная:
        String res;
        // Отображение окна с полем ввода с помощью
        // метода showInputDialog() из класса JOptionPane:
        res=showInputDialog(null,txt,"Число (по умолчанию
10)",QUESTION_MESSAGE);
        // Проверяется значение текстовой переменной:
        if(res==null){
            // Если пользователь отменил ввод числа
            // (значение, возвращаемое по умолчанию):
            return 10;
        }
        else{
            // Преобразование текстового представления числа в число:
            return Integer.parseInt(res);
        }
    }
}
// Продолжение на следующем слайде!!!
```

Наследование класса JOptionPane (продолжение)

```
// Класс с главным методом программы:
class ExtendingJOptionPaneDemo{
   public static void main(String[] args){
      // Отображение диалогового окна с сообщением:
      MyPane.showMessage("Всем привет!");
      // Переменная для записи числового значения:
      int number;
      // Отображение диалогового окна с полем ввода
      // и считывание целочисленного значения:
      number=MyPane.getInteger("Введите целое число");
      // Текст для отображения в диалоговом окне:
      String txt="Числа от 1 до "+number+":\n";
      // Формирование текстовой строки с
      // последовательностью натуральных чисел:
      for(int k=1;k<=number;k++){
         txt+=k+" ";
      }
      // Отображение диалогового окна с сообщением:
      MyPane.showMessage(txt,"Целые числа");
   }
}
```

Наследование класса JOptionPane (результат)

Для корректного выполнения программы предварительно создается файл `giraffe.png` (с прозрачной подложкой) с графическим изображением размерами в `75` пикселей в ширину и высоту. Файл помещается в каталог `d:\books\pictures`



Конструктор подкласса

Существуют специальные правила описания конструктора подкласса:

- При создании объекта подкласса сначала вызывается конструктор суперкласса. Для этого указывается ключевое слово `super`, после которого в круглых скобках передаются аргументы для конструктора суперкласса

- Если аргументов нет, круглые скобки (пустые) все равно указываются

- В теле конструктора подкласса инструкция с ключевым словом `super` должна быть первой

- Если в теле конструктора подкласса инструкция с ключевым словом `super` отсутствует, то автоматически вызывается конструктор суперкласса без аргументов

Конструктор подкласса

Пример использования конструктора подкласса - 1

```
// Суперкласс:
class Alpha{
   String name; // Текстовое поле
   int code;    // Целочисленное поле
   // Конструктор с тремя аргументами:
   Alpha(String name,int code){
      this.name=name;
      this.code=code;
      System.out.println("Класс Alpha:");
      System.out.println("Поле name - "+this.name);
      System.out.println("Поле code - "+this.code);
   }
   // Конструктор с одним текстовым аргументом:
   Alpha(String name){
      // Вызов конструктора с двумя аргументами:
      this(name,0);
   }
   // Конструктор с одним целочисленным аргументом:
   Alpha(int code){
      // Вызов конструктора с двумя аргументами:
      this("Белый",code);
   }   // Продолжение на следующем слайде!!!
```

Конструктор подкласса

Пример использования конструктора подкласса - 2

```
    // Конструктор без аргументов:
    Alpha(){
        this("Желтый",1);
    }
}
// Подкласс:
class Bravo extends Alpha{
    // Символьное поле:
    char symbol;
    // Закрытый метод для отображения значения
    // символьного поля:
    private void show(){
        System.out.println("Класс Bravo:");
        // Отображение значения символьного поля:
        System.out.println("Поле symbol - "+this.symbol);
        // Отображение "горизонтальной линии":
        for(int k=1;k<=18;k++){
            System.out.print("-");
        }
        System.out.println("");
    }
    // Продолжение на следующем слайде!!!
```

Конструктор подкласса

Пример использования конструктора подкласса - 3

```
// Конструктор с тремя аргументами:
Bravo(String name,int code,char symbol){
    // Вызов конструктора суперкласса
    // с двумя аргументами:
    super(name,code);
    // Присваивание символьному полю значения:
    this.symbol=symbol;
    // Вызов закрытого метода:
    show();
}
// Конструктор с одним символьным аргументом:
Bravo(char symbol){
    // Вызов конструктора суперкласса без аргументов:
    super();
    // Присваивание символьному полю значения:
    this.symbol=symbol;
    // Вызов закрытого метода:
    show();
}
// Продолжение на следующем слайде!!!
```

Конструктор подкласса

Пример использования конструктора подкласса - 4

```
// Конструктор с одним текстовым аргументом:
Bravo(String name){
    // Вызов конструктора суперкласса
    // с одним текстовым аргументом:
    super(name);
    // Присваивание символьному полю значения:
    this.symbol='A';
    // Вызов закрытого метода:
    show();
}
// Конструктор с одним целочисленным аргументом:
Bravo(int code){
    // Вызов конструктора суперкласса
    // с одним целочисленным аргументом:
    super(code);
    // Присваивание символьному полю значения:
    this.symbol='B';
    // Вызов закрытого метода:
    show();
}
// Продолжение на следующем слайде!!!
```

Конструктор подкласса

Пример использования конструктора подкласса - 5

```
    // Конструктор без аргументов:
    Bravo(){
        // Сначала неявно вызывается конструктор
        // суперкласса без аргументов.
        // Присваивание значения символьному полю:
        this.symbol='C';
        // Вызов закрытого метода:
        show();
    }
    // Конструктор с двумя аргументами:
    Bravo(String name,int code){
        // Вызов конструктора подкласса с тремя аргументами:
        this(name,code,'D');
    }
}
// Продолжение на следующем слайде!!!
```

Конструктор подкласса

Пример использования конструктора подкласса - 6

```
// Класс с главным методом программы:
class SubclassConstructorDemo{
   public static void main(String[] args){
      // Объектная переменная подкласса:
      Bravo obj;
      // Розные способы создания объекта подкласса:
      obj=new Bravo();
      obj=new Bravo("Красный");
      obj=new Bravo(100);
      obj=new Bravo("Зеленый",200);
      obj=new Bravo('Y');
      obj=new Bravo("Синий",300,'Z');
   }
}
```

Конструктор подкласса

Детали

Инструкция `this` с круглыми скобками означает вызов версии конструктора того же самого класса, в конструкторе которого размещена соответствующая команда. Инструкция `super` с круглыми скобками означает вызов версии конструктора другого класса, но являющегося при этом суперклассом для класса, в конструкторе которого находится команда с `super`-инструкцией.

Что касается непосредственно конструктора класса `Bravo`, в котором выполняется команда `this(name,code,'D')`, то стоит заметить, что явно указанной `super`-инструкции там нет, но при это и конструктор суперкласса без аргументов не вызывается. Ситуация более "хитрая", по сравнению с тем, когда просто явно не указана команда вызова конструктора суперкласса. Дело в том, то соответствующая инструкция по вызову конструктора суперкласса "спрятана" в команде `this(name,code,'D')`: вызов конструктора суперкласса происходит при вызове версии конструктора класса `Bravo` с тремя аргументами.

Конструктор подкласса

Результат программы

```
Класс Alpha:
Поле name - Желтый
Поле code - 1
Класс Bravo:
Поле symbol - C
------------------
Класс Alpha:
Поле name - Красный
Поле code - 0
Класс Bravo:
Поле symbol - A
------------------
Класс Alpha:
Поле name - Белый
Поле code - 100
Класс Bravo:
Поле symbol - B
------------------
```

```
Класс Alpha:
Поле name - Зеленый
Поле code - 200
Класс Bravo:
Поле symbol - D
------------------
Класс Alpha:
Поле name - Желтый
Поле code - 1
Класс Bravo:
Поле symbol - Y
------------------
Класс Alpha:
Поле name - Синий
Поле code - 300
Класс Bravo:
Поле symbol - Z
------------------
```

Наследование и закрытые члены

Закрытые члены в суперклассе - 1

```
// Суперкласс:
class Alpha{
   // Закрытое поле:
   private int code;
   // Открытый метод для присваивания значения
   // закрытому полю:
   public void set(int code){
      this.code=code;
   }
   // Открытый метод для отображения значения
   // закрытого поля:
   public void show(){
      System.out.println("Поле code: "+code);
   }
   // Конструктор класса:
   Alpha(int code){
      set(code);
   }
}
// Продолжение на следующем слайде!!!
```

Наследование и закрытые члены

Закрытые члены в суперклассе - 2

```
// Подкласс:
class Bravo extends Alpha{
   // Конструктор подкласса:
   Bravo(int code){
      // Вызов конструктора суперкласса:
      super(code);
   }
}
// Класс с главным методом программы:
class UsingPrivatesDemo{
   public static void main(String[] args){
      // Создание объекта подкласса:
      Bravo obj=new Bravo(100);
      // Проверка значения поля:
      obj.show();
      // Присваивание значения полю:
      obj.set(200);
      // Отображение значения поля:
      obj.show();
   }
}
```

Результат

```
Поле code: 100
Поле code: 200
```

Пакеты

- Классы группируются по пакетам, которые играют роль своеобразных "контейнеров", или объединений классов (и других пакетов, которые обычно называют подпакетами)
- В пределах контейнера название класса должно быть уникальным, но если классы находятся в разных пакетах, то их названия могут совпадать

Чтобы создать пакет, необходимо в начале программного кода поместить ключевое слово `package` и имя создаваемого пакета:

```
package имя_пакета;
```

При создании подпакета имя пакета и подпакета разделяется точкой:

```
package имя_пакета.имя_подпакета;
```

Для создания пакета `mydata` используем инструкцию

```
package mydata;
```

При создании подпакета `mybox` в пакете `mydata` используем инструкцию

```
package mydata.mybox;
```

Файл может содержать только одну `package`-инструкцию, и она должна быть первой инструкцией в файле

Импорт пакетов

- Чтобы в программе можно было использовать класс или классы из некоторого пакета, класс или весь пакет импортируют
- Для импорта класса или пакета используют инструкцию `import`, после которой указывается имя пакета и, через точку, имя импортируемого класса или звездочка * (если импортируются все `public`-классы из пакета)

Для импортирования класса с названием `MyClass` из пакета `mydata` используем инструкцию

```
import mydata.MyClass;
```

Чтобы импортировать все `public`-классы из пакета `mydata`, вместо имени класса следует поставить звездочку *

```
import mydata.*;
```

Чтобы импортировать классы из подпакета `mybox` пакета `mydata` используем инструкцию

```
import mydata.mybox.*;
```

Файл может содержать несколько `import`-инструкций. Такие инструкции размещаются в начале программного кода, а если в программном коде есть `package`-инструкция, то сразу после нее.

Статический импорт

- При работе со статическими членами классов из импортируемых пакетов существует возможность упростить процедуру обращения к таким членам. Для этого используют статический импорт
- При статическом импорте после инструкции `import` указывается ключевое слово `static`, а далее полная ссылка на статический член класса.

Пример такой инструкции приведен ниже:

```
import static javax.swing.JOptionPane.showMessageDialog;
```

Инструкция статического импорта всех статических членов класса `JOptionPane`:

```
import static javax.swing.JOptionPane.*;
```

Уровни доступа и наследование

Место, из которого получаем доступ к члену класса	Спецификатор уровня доступа, с которым описан член класса			
	`public`	`protected`	не указан	`private`
В том же классе	+	+	+	+
В подклассе в том же пакете	+	+	+	–
В обычном класса в том же пакете	+	+	+	–
В подклассе другого пакета	+	+	–	–
В обычном классе в другом пакете	+	–	–	–

- Классы могут описываться со спецификатором доступа `public`
- Если класс описан без спецификатора доступа `public`, то он доступен в пределах пакета
- Если класс описан со спецификатором доступа `public`, то он доступен и вне пакета. Такой класс называют `public`-классом или открытым классом
- В файле может быть только один открытый класс, а название файла должно совпадать с названием открытого класса

Переопределение методов

- **Наследуемые из суперкласса в подклассе методы можно переопределять**
- **Чтобы переопределить унаследованный метод, в подклассе данный метод описывается заново в явном виде**

Детали

В языке Java есть ключевое слово `final`, которое позволяет решать ряд важных задач. Так, если с ключевым словом `final` описана переменная, то такая переменная является константой: после инициализации значение этой переменной изменить нельзя. Если с ключевым словом `final` описан метод в суперклассе, то в подклассе такой метод не переопределяется. Наконец, если описать с ключевым словом `final` класс, то такой класс не может быть суперклассом - другими словами, на основе такого класса нельзя создать (путем наследования) другой класс.

Переопределение методов

Переопределение методов - 1

```
class Alpha{          // Суперкласс
   String name;       // Текстовое поле
   void show(){       // Метод для отображения значения текстового поля
      System.out.println("Объект класса Alpha:");
      System.out.println("Поле name - "+name);
   }
   Alpha(String name){          // Конструктор класса
      this.name=name;
   }
}
class Bravo extends Alpha{     // Подкласс
   int code;                   // Целочисленное поле
   // Переопределение метода. Новой версией метода
   // отображаются значения двух полей:
   void show(){
      System.out.println("Объект класса Bravo:");
      System.out.println("Поле name - "+name);
      System.out.println("Поле code - "+code);
   }
   // Конструктор класса:
   Bravo(String name,int code){
      // Вызов конструктора суперкласса:
      super(name);
      this.code=code;
   }
}  // Продолжение на следующем слайде!!!
```

Переопределение методов

Переопределение методов - 2

```
// Главный класс:
class OverrideMethodDemo{
   // Главный метод:
   public static void main(String[] args){
      // Создание объекта суперкласса:
      Alpha objA=new Alpha("objA");
      // Создание объекта подкласса:
      Bravo objB=new Bravo("objB",123);
      // Вызов метода из объекта суперкласса:
      objA.show();
      // Вызов метода из объекта подкласса:
      objB.show();
   }
}
```

Результат

```
Объект класса Alpha:
Поле name - objA
Объект класса Bravo:
Поле name - objB
Поле code - 123
```

Вызов разных версий методов

Вызов разных версий методов

```
class Alpha{
   String name;
   void show(){
      System.out.println("Объект "+name);
   }
   Alpha(String name){
      this.name=name;
   }
}
class Bravo extends Alpha{
   int code;
   void show(){
      super.show(); // Вызов версии метода из суперкласса
      System.out.println("Числовое поле "+code);
   }
   Bravo(String name,int code){
      super(name);  // Вызов конструктора суперкласса
      this.code=code;
   }
}
class MoreOverridingDemo{
   public static void main(String[] args){
      Alpha objA=new Alpha("objA");
      Bravo objB=new Bravo("objB",123);
      objA.show();
      objB.show();
   }
}
```

Результат

```
Объект objA
Объект objB
Числовое поле 123
```

Перекрытие полей

Перекрытие полей

```
class Alpha{                    // Суперкласс
   String name;                 // Текстовое поле
}
class Bravo extends Alpha{      // Подкласс
   String name;                 // Текстовое поле
   void show(){                 // Метод для отображения значений полей
      // Значение унаследованного из суперкласса поля:
      System.out.println("Из класса Alpha: "+super.name);
      // Значение описанного в подклассе поля:
      System.out.println("Из класса Bravo: "+name);
   }
   Bravo(String a,String b){ // Конструктор
      // Вызов конструктора (по умолчанию) суперкласса:
      super();
      // Значение унаследованного из суперкласса поля:
      super.name=a;
      // Значение описанного в подклассе поля:
      name=b;
   }
}
class HidingFieldDemo{          // Главный класс
   public static void main(String[] args){
      // Создание объекта подкласса:
      Bravo obj=new Bravo("alpha","bravo");
      // Проверка значений полей:
      obj.show();
   }
}
```

Результат

```
Из класса Alpha: alpha
Из класса Bravo: bravo
```

Виртуальность методов и конструкторов - 1

```
// Суперкласс:
class Alpha{
   // Текстовое поле:
   String name;
   // Метод для отображения сообщения:
   void objectCreated(){
      System.out.println("Создан объект класса Alpha");
   }
   // Метод для отображения сообщения:
   void hello(){
      System.out.println("Объект класса Alpha");
   }
   // Метод для отображения значения поля:
   void show(){
      // Вызов метода для отображения сообщения:
      hello();
      // Отображение значения поля:
      System.out.println("Поле name: "+name);
   }
   // Конструктор:
   Alpha(String txt){
      // Вызов метода для отображения сообщения:
      objectCreated();
      // Присваивание значения полю:
      name=txt;
   }
} // Продолжение на следующем слайде!!!
```

Виртуальность методов и конструкторов - 2

```
class Bravo extends Alpha{       // Подкласс
   // Переопределение метода:
   void objectCreated(){
      System.out.println("Создан объект класса Bravo");
   }
   // Переопределение метода:
   void hello(){
      System.out.println("Объект класса Bravo");
   }
   // Конструктор:
   Bravo(String txt){
      // Вызов конструктора суперкласса:
      super(txt);
   }
} // Главный класс:
class VirtualMethodsDemo{
   // Главный метод:
   public static void main(String[] args){
      // Создание объекта суперкласса:
      Alpha objA=new Alpha("alpha");
      // Отображение значения поля:
      objA.show();
      // Создание объекта подкласса:
      Bravo objB=new Bravo("bravo");
      // Отображение значения поля:
      objB.show();
   }
}
```

Результат

```
Создан объект класса Alpha
Объект класса Alpha
Поле name: alpha
Создан объект класса Bravo
Объект класса Bravo
Поле name: bravo
```

Переопределение и перегрузка методов

```
class Alpha{                              // Суперкласс
   void show(){                           // Версия метода без аргумента
      System.out.println("Класс Alpha");
   }
   void show(String txt){                 // Версия метода с текстовым аргументом
      System.out.println(txt);
   }
}  // Подкласс:
class Bravo extends Alpha{
   void show(){                            // Переопределение версии метода без аргументов
      System.out.println("Класс Bravo");
   }
   void show(int num){                     // Версия метода с целочисленным аргументом
      System.out.println("Число "+num);
   }
} // Главный класс:
class OverloadingAndOverridingDemo{
   public static void main(String[] args){  // Главный метод
      Alpha objA=new Alpha();              // Создание объекта суперкласса
      objA.show();                         // Вызов разных версий метода
      objA.show("Объект objA");
      Bravo obj=new Bravo();               // Создание объекта подкласса
      obj.show();                          // Вызов разных версий метода
      obj.show("Объект objB");
      obj.show(123);
   }
}
```

Результат

```
Класс Alpha
Объект objA
Класс Bravo
Объект objB
Число 123
```

Метод toString()

Метод toString()

```
class MyClass{                          // Пользовательский класс
   String name;                         // Текстовое поле
   int code;                            // Целочисленное поле
   MyClass(String txt,int num){         // Конструктор
      name=txt;
      code=num;
   }
   // Переопределение метода toString():
   public String toString(){
      String txt="Объект класса MyClass\n"; // Локальная текстовая переменная
      txt+="Поле name: "+name+"\n";
      txt+="Поле code: "+code+"\n";
      // Импровизированная "линия":
      for(int k=1;k<=21;k++){
         txt+="-";
      }
      return txt;    // Результат метода
   }
} // Главный класс:
class UsingToStringDemo{
   // Главный метод:
   public static void main(String[] args){
      // Создание объекта:
      MyClass obj=new MyClass("объект obj",123);
      // Объект передан аргументом методу:
      System.out.println(obj);
   }
}
```

Результат

```
Объект класса MyClass
Поле name: объект obj
Поле code: 123
---------------------
```

Объектная переменная суперкласса

- Объектная переменная суперкласса может ссылаться на объект подкласса
- Доступ через объектную переменную суперкласса есть только к тем полям и методам, которые объявлены в суперклассе

Объектная переменная суперкласса - 1

```
class Alpha{                                    // Суперкласс
   String name;                                 // Текстовое поле
   void show(){                                 // Метод для отображения значения поля
      String txt="Объект класса Alpha\n";       // Локальная текстовая переменная
      txt+="Поле name: "+name+"\n";
      for(int k=1;k<=20;k++){
         txt+="-";
      } // Отображение сообщения:
      System.out.println(txt);
   }
} // Подкласс:
class Bravo extends Alpha{
   int code;                                    // Целочисленное поле
   void show(){                                 // Переопределение метода
      String txt="Объект класса Bravo\n";       // Локальная текстовая переменная
      txt+="Поле name: "+name+"\n";
      txt+="Поле code: "+code+"\n";
      for(int k=1;k<=20;k++){
         txt+="-";
      } // Отображение сообщения:
      System.out.println(txt);
   }
} // Продолжение на следующем слайде!!!
```

Объектная переменная суперкласса

Объектная переменная суперкласса - 2

```
// Главный класс:
class SuperAndSubDemo{
   // Главный метод:
   public static void main(String[] args){
      // Создание объекта суперкласса:
      Alpha objA=new Alpha();
      // Присваивание значения полю объекта суперкласса:
      objA.name="alpha";
      // Вызов метода из объекта суперкласса:
      objA.show();
      // Создание объекта подкласса:
      Bravo objB=new Bravo();
      // Присваивание значений полям объекта подкласса:
      objB.name="bravo";
      objB.code=123;
      // Вызов метода из объекта подкласса:
      objB.show();
      // Переменной суперкласса значением присваивается
      // ссылка на объект подкласса:
      objA=objB;
      // Значение поля объекта подкласса изменяется через переменную суперкласса:
      objA.name="charlie";
      // Вызов метода из объекта подкласса через переменную суперкласса:
      objA.show();
      // Вызов метода из объекта подкласса через переменную подкласса:
      objB.show();
   }
}
```

Результат

```
Объект класса Alpha
Поле name: alpha
--------------------
Объект класса Bravo
Поле name: bravo
Поле code: 123
--------------------
Объект класса Bravo
Поле name: charlie
Поле code: 123
--------------------
Объект класса Bravo
Поле name: charlie
Поле code: 123
--------------------
```

Домашнее задание

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

Продолжение следует…