Конспект лекций по Java. Занятие 7

В.Фесюнов

1. Наследование классов

Наследование классов (*inheritance*) один из существенных атрибутов ООП (объектно-ориентированного программирования). Оно позволяет строить новые классы на базе существующих, добавляя в них новые возможности или переопределяя существующие.

Что такое наследование.

Пусть есть класс A, он имеет поля a1, a2, ..., an и методы f1(), f2(), ... , fm(). Тогда мы можем на его основе построить класс B. Класс B наследует все поля и методы класса A (за исключением конструкторов). Кроме того, в B можно:

- добавить новые поля;
- добавить новые методы;
- переопределить какие-либо методы класса A.

Это в основном, без деталей.

Синтаксис:

```
class B extends A {
   . . .    // тело класса B
}
```

Внутри записываются "дополнения и изменения", вносимые классом B.

Класс A называют базовым классом или суперклассом (superclass), иногда - родительским классом, предком. B - порожденным, дочерним, подклассом (subclass), классом-потомком.

В свою очередь, класс A может быть порожден на базе другого класса, тогда этот класс является предком как для A, так и для B.

От одного класса может быть порождено произвольное количество новых классов. В результате получается иерархия классов, порожденных один от другого.

Пример (наследование полей)

```
class Base {
     int a, b, c;
     . . .
}

class Derived extends Base {
     long d, e;
     . . .
}
```

Объекты класса Base имеют три поля (a, b и c), объекты класса Derived - пять полей (a, b, c, d и e).

Пример (наследование методов)

```
class Base {
     int f() {
          . . .
     }
     void g(int p) {
          . . .
     }
}

class Derived extends Base {

     long f1() {
          . . .
     }

}
```

Класс Base имеет два метода (f() и g(...)), класс Derived - три метода (f(), g(...) и f1()).

Пример (переопределение методов)

```
class Base {
     int f() {
          . . .
     }
     void g(int p) {
          . . .
     }
}

class Derived extends B {

     int f() {
          . . .
     }

}
```

Класс Base имеет два метода (f() и g(...)) и класс Derived тоже имеет два метода (f() и g(...)). Но для объектов класса Derived будет вызываться не метод f() из Base, а метод f() из Derived. На английском переопределение методов называется overriding.

При переопределении (overriding) метод в порожденном классе должен иметь в точности то же описание, что и метод в базовом классе. В частности, попытка описать "long f()..." или даже "private int f()..." привела бы к ошибке при трансляции программы.

1.1. Класс Object

В Java фактически все классы строятся на базе наследования, т.к., даже если не написать "extends имя_класса", будет подразумеваться extends Object. Т.е. класс Object является базовым классом для всех классов Java.

Класс Object имеет ряд методов, при наследовании методы базового класса

наследуются его потомками. Следовательно, все классы Java имеют как минимум те методы, которые есть в классе Object.

Взглянем на документацию по классу Object. В классе Object определены методы, большая часть из которых имеют непонятное для нас (на текущий момент) назначение. Но некоторые из них мы можем уже сейчас рассмотреть.

public String toString() Предназначен для формирования некоторого текстового представления объекта.

protected Object clone() throws CloneNotSupportedException Предназначен для создания копии объекта.

public boolean equals(Object obj) Позволяет сравнивать объекты.

public final Class getClass() Выдает класс данного объекта в виде объекта класса Class.

protected void finalize() throws Throwable Этот метод вызывается при удалении объекта из памяти сборщиком мусора.

Все указанные методы, кроме `getClass` и, частично, `toString` , не предназначены для непосредственного использования (в том смысле, что их нужно переопределить в порожденных классах).

Метод `getClass` возвращает специальный объект класса `Class` , содержащий много полезной информации о классе объекта. Метод `toString` по умолчанию выдает полное имя класса объекта и его адрес, что можно использовать при отладке программы.

Если почитать документацию по остальным методам, то мы увидим, что она, в основном, определяет, что должен делать тот или иной метод и лишь в конце описывается, как этот метод реализован в классе Object.

Разберем, например, метод `equals` . Его назначение - сравнивать объекты на равенство. Наличие его в классе Object (базовом для всех остальных классов) позволяет нам применять этот метод для любых объектов, что очень удобно. Но в классе `Object` он реализован "самым жестким" образом - как сравнение адресов объектов. Т.е. при сравнении двух объектов мы получим равенство только в том случае, если на самом деле это один и тот же объект. Естественно, что чаще всего нам

потребуется какая-то другая реализация данного метода.

Наследование классов имеет много связанных с ним нюансов и особенностей. И в дальнейшем рассмотрении мы будем постепенно разбираться с ними.

1.2. Инициализация полей при наследовании классов

Уточним определение инициализации объектов в случае наследования классов.

Все действия по инициализации выполняются этап за этапом в порядке наследования классов, т.е. сначала п.1 для класса `Object` , потом п.1 для следующего класса и т.д., потом п.2 в том же порядке и т.д.

- При первом обращении к классу выделяется память под статические поля класса и выполняется их инициализация.
- Выполняется распределение памяти под создаваемый объект.
- Выполняются все инициализаторы нестатических полей класса.
- Выполняется вызов конструктора класса.

Рассмотрим абстрактный пример.

```
class A {
   ...
   A() {...}
   ...
}

class B extends A {
   ...
   B() { ... }
   ...
}
```

где-то

B b = new B();

При этом сначала выполнится конструктор A(), потом конструктор B().

В данном примере класс A имеет конструктор по умолчанию. Но, что делать, если

базовый класс, например, не имеет конструктора по умолчанию, или же нужно, чтобы при создании объекта отработал не конструктор по умолчанию, а какой-либо другой конструктор.

В Java эта ситуация предусмотрена. Используя ключевое слово super, можно в конструкторе порожденного класса вызвать нужный конструктор базового класса.

Пример

```
class X {
   X(int a) { ... }
   ...
}

class Y extends X {
    Y() {
        super(0);
        ...
    }
   ...
}
```

Ключевое слово super может использоваться и для явного вызова методов базового класса. Это необходимо, если некоторый метод базового класса был переопределен в порожденном классе.

Пример

```
class Base {
    int x = 1;
    long y;

    Base(long y) {
        this.y = y;
    }

    Base() {
         this(0);   // вызов конструктора Base(long y)
```

```
    }

    public long f() {
        return x*y;
    }
}

class Derived extends Base {

    String name = "";

    Derived(String name, long par) {
          super(par);   // вызов конструктора Base(long y)
          this.name = name;
    }

    public long g(int r) {
         return r+super.f();   // вызов метода f() класса Base
    }

    public long f() {
          x++;
          return 2*y;
    }
}
```

Рассмотрим подробно, из каких полей и методов состоит класс Derived. Из класса Base в него вошли поля x и y и метод f(). Плюс в нем определено поле name и методы g(...) и f(), причем метод f() переопределяет одноименный метод класса Base.

В нашем примере, в методе g(...), требуется вызвать не метод f() класса Derived, а метод f() класса Base. При этом применяется ключевое слово super. Без него вызвался бы метод f() класса Derived.

1.3. Контрольная задача

Рассмотрим пример использования класса Derived. Требуется определить, что произойдет при выполнении такого фрагмента.

```
Derived d = new Derived("test", 10);
long c = d.g(5);
long p = d.f();
```

1.4. В Java нет множественного наследования

В отличие от C++ в Java нет множественного наследования. Т.е. у класса может быть только один базовый класс. Соответственно отношение наследования формирует строгую иерархию классов - иерархию наследования (дерево классов). Это хорошо видно на примере документации.

Откроем документацию по стандартной библиотеке Java. Вверху любой страницы имеется банер с гиперссылками. Кликнем на ссылке "Tree". Мы увидим дерево классов стандартной библиотеки Java.

1.5. Практическая работа

Вернемся к примеру Dialog1. Его можно реализовать иначе, если использовать аппарат наследования.

```
// Dialog2.java
// 2-й пример визульного приложения на Java.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Dialog2 extends JFrame {

  Dialog2() {
     super("Первое визуальное приложение");

     try  {
       UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
     }
     catch(Exception e) {
     }
```

```
    setSize(300, 200);
    Container c = getContentPane();
    c.add(new JLabel("Hello, привет"));
    WindowListener wndCloser = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    addWindowListener(wndCloser);

    setVisible(true);
  }

  public static void main(String[] args) {
    new Dialog2();
  }
}
```

Этот пример более соответствует стилю Java, чем Dialog1.

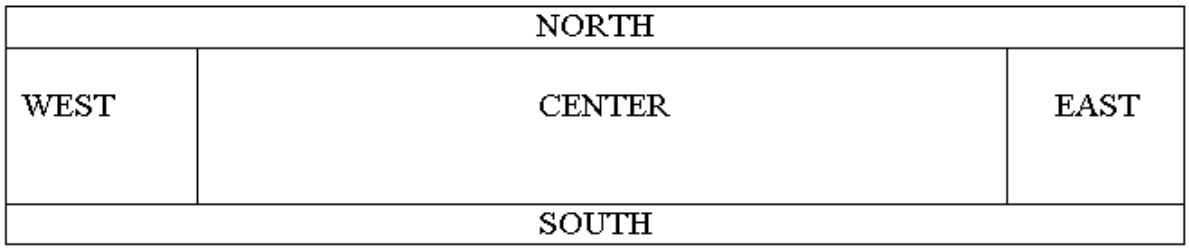
Рассмотрим более сложный пример.

```
// Dialog3.java
// Визульное приложения с текстовой областью.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Dialog3 extends JFrame {

  JTextArea txt;

  Dialog3() {
     super("Визульное приложения с текстовой областью");

     try  {
       UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
     }
```

```
     catch(Exception e) {
     }

    setSize(400, 200);
    Container c = getContentPane();
    c.add(new JLabel("Hello, привет"), BorderLayout.NORTH); // 0
    txt = new JTextArea(5, 30);                             // 1
    JScrollPane pane = new JScrollPane(txt);                // 2
    c.add(pane, BorderLayout.CENTER);                       // 3

    WindowListener wndCloser = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    addWindowListener(wndCloser);

    setVisible(true);
  }

  public void test() {
    txt.append("Первая строка\n");
    txt.append("Вторая строка\n");
  }

  public static void main(String[] args) {
    Dialog3 d = new Dialog3();
    d.test();
  }
}
```

Этот пример требует некоторых пояснений по принципам организации диалога в Java. В отличие от других языков, в частности VB, в Java все визуальные компоненты масштабируемы. Поэтому при их добавлении на окно приложения нельзя указать их координаты и размеры. Вместо этого используется понятие Layout'a ("разместитель", компоновщик). В этом примере задействован BorderLayout (он является Layout'ом по умолчанию для JFrame). Потом мы подробнее познакомимся с этим понятием. В данном случае просто разберем, что обеспечивает BorderLayout и как с ним работать.

Компоновщик BorderLayout разбивает область окна (панели) на следующие части.



Он позволяет добавлять компоненты в любую из этих частей. При добавлении (метод add(...)) нужно указать в какую часть панели мы добавляем компоненту (см.строки 0 и 3).

Класс JTextArea позволяет создать многострочную область ввода/вывода данных. Для того, чтобы эта область была скроллируемой, дополнительно используется JScrollPane, внутрь которого помещается объект txt класса JTextArea.

Предупреждение:

На панель окна мы заносим не txt, а объект pane класса JScrollPane.

2. Домашнее задание

На базе Dialog3 сделать программу. Эта программа принимает N параметров вызова (для получения N использовать args.length). Эти параметры - элементы вектора. Она строит массив типа double. Потом на базе этого массива - класс DoubleVector. Потом выводит в текстовую область txt значения элементов вектора в виде:

Исходный вектор: 1.0 10.0 9.0 8.0