Лекция 7

Классы - оболочки

Каждому простому типу в Java соответствует класс-оболочка.

Классы-оболочки выполняют две основные функции.

Первая состоит в поддержке контейнера методов и переменных, относящихся к определенному типу (скажем, методов преобразования строк и констант, задающих границы интервала допустимых значений).

Рассмотрим пример кода, проверяющий можно ли использовать float вместо double:

```
double aval  = Math.abs(value);
if (Float.MAX_VALUE >= aval && aval  >=
                                        Float.MIN_VALUE)
  return fasterFloatCalc((float) value);
else
  return slowerDoubleCalc(value);
```

Вторая функция классов-оболочек заключается в обеспечении возможности создания объектов для хранения значений простых типов и использования их в контексте классов, которые "умеют" обращаться только со ссылками на объекты типа Object.

Класс-оболочка определяет неизменяемый объект для представления значения соответствующего типа. После создания объекта значение, которое в нем хранится, изменить нельзя.

Так, например, объект, созданный в результате выполнения инструкции new Integer(1), будет всегда содержать значение 1 — в составе класса Integer нет методов, которые позволили бы его заменить.

Некоторые классы-оболочки:

1. Number (это абстрактный класс – наследуемый всеми классами – оболочками
   Short, Byte, Integer,Long, Float, Double)
2. Short
3. Byte
4. Integer
5. Long
6. Float
7. Double
8. Boolean
9. Character
10. Void

В каждом классе –оболочке определены следующие конструкторы:

1. принимающий в качестве параметра значение простого типа и создающий объект соответствующего класса-оболочки; например, конструкторы Character(char) и integer(int);
2. преобразующий содержимое единственного параметра типа String в исходное значение объекта (за исключением класса Character, в составе которого такого конструктора нет), например new Float("6.02e23"); для числовых типов строка параметра должна содержать представление величины в десятичной системе счисления — если преобразование недопустимо, выбрасывается исключение типа NumberFormatException.

В каждом из классов-оболочек определены методы:

1. **public static Type valueOf(String str)**

Возвращает объект заданного типа *Type,* содержащий значение, преобразованное из строки str.

Для числовых типов подразумевается, что строка содержит представление аргумента в системе счисления с основанием 10.

Например, **Float.valueOf ("6.02e23")** и **Integer.valueOf("16").** Вызов метода равнозначен использованию оператора new предполагающего обращение к конструктору с параметром типа String.

Если содержимое str не может быть преобразовано, выбрасывается исключение типа NumberFormatException.

2. **public String toString()**

Переопределенная версия метода Object. toString(), обеспечивающая получение строкового представления содержимого объекта класса-оболочки.

3. **public type typeValue()**

Возвращает значение простого типа type, соответствующее содержимому текущего объекта класса-оболочки. Например,

new Integer(6).intValue() возвращает 6.

4. **public int compareTo(Type other)**

Возвращает значение, меньшее (большее) нуля или равное нулю, если содержимое текущего объекта соответственно меньше (больше) значения other того же типа Type или равно ему.

Метод не определен в классе Boolean.

5. **public int compareTo(Object obj)**

Если obj относится к тому же типу, что и текущий объект, метод равнозначен compareTo(Type).

В противном случае выбрасывается исключение типа ClassCastException.

Метод не определен в классе Boolean

6. **public boolean equals(Object obj)**

Возвращает true, если текущий объект и объект, переданный в качестве аргумента, относятся к одному типу и содержат одинаковые значения.

Например, для двух объектов x и y типа Integer выражение x.equals (y) равно true, если и только если x.intValue() == y.intValue().

Если значение obj не принадлежит тому же типу, что и текущий объект, либо равно null, метод возвращает false.

7. **public int hachCode()**

Возвращает хеш-код текущего объекта, основанный на его содержимом.

Рефлексия

Пакет **java.lang.reflect** содержит набор метаклассов рефлексии (reflection), позволяющих подробно исследовать любой тип Java.

С их помощью можно создать полнофункциональный браузер типов или приложение, которое интерпретирует пользовательский код и преобразует его в фактические инструкции объявления классов, создания объектов, вызова методов и т.д.

Все типы содержатся в пакете java.lang.reflect, за исключением классов Class и Package, которые служат частью пакета java.lang.

В основе механизма рефлексии находится объект типа **Class**.

Он позволяет получить полный список членов конкретного класса, проследить иерархию базовых типов (реализуемых интерфейсов и расширяемых классов) и извлечь информацию о самом классе (например, об используемых в его объявлении модификаторах — public, abstract, final и т.д.) и о пакете, в котором он содержится.

Механизм рефлексии позволяет создавать код, выполняющий те же действия, которые можно осуществить с помощью кода, написанного непосредственно.

Зная наименование определенного класса (которого в момент написания программы может не быть в наличии), удается получить соответствующий объект типа **Class** и использовать его для создания объектов этого класса.

Класс Class

Каждому типу Java — простым типам, классам, интерфейсам и массивам — соответствует собственный объект класса Class.

Существует также специальный объект Class, представляющий служебное слово void. Объекты Class могут использоваться для получения информации о типах и — если тип относится к ссылочным — для создания новых объектов этого типа.

Существует четыре способа получения объекта Class:

1. применение метода getClass;
2. использование литерала типа Class (имени класса, сопровождаемого суффиксом .class, — например, String.class);
3. применение статического метода Class - forName с указанием полного имени класса, включающего название пакета, которому он принадлежит;
4. получение данных с помощью методов рефлексии, которые возвращают объекты Class для вложенных классов и интерфейсов.

Рассмотрим пример(программа отображает на экране сведения об иерархии типов для типа, значение которого передается в качестве параметра командной строки)

```
public class TypeDesc {
  public static void main(String[] args){
   TypeDesc desc = new TypeDesc();
   for(int i=0; i< args.length; i++){
    try {
      Class startClass = Class.forName(args[i]);
      desc.printType(startClass,0,basic); }
    catch (ClassNotFoundException e){
      System.err.println(e);  // Если возникла ошибка
    }
   }
  }
```

```
private static String[]
  basic= {"class", "interface" },
  supercl= {"extends", "implements"},
  iFace= { null, "extends"};
private void  printType(Class type, int depth,
                                          String[] labels){
   if(type==null) return; //Завершение рекурсии

   //вывод информации о текущем типе
   for (int i =0; i < depth; i++)
     System.out.print("  ");
   System.out.print(labels[type.isInterface()? 1 : 0] + " ");
   System.out.println(type.getName());
```

```
// вывести информацию обо всех интерфейсах,
   реализуемых текущим классом
 Class[] interfaces = type.getInterfaces();
   for (int i =0; i < interfaces.length; i++){
      printType(interfaces[i], depth + 1,
                     type.isInterface()? iFace : supercl);
    }
// выполнить рекурсию по базовым классам
   printType(type.getSuperclass(), depth + 1,
                                                  supercl);
 }
}
```

Результат работы программы при задании в качестве параметра строки java.util.HashMap будет

class java.util.HashMap

implements java.util.Map

implements java.lang.Cloneable

implements java.io.Serializable

extends java.util.AbstractMap

implements java.util.Map

extends java.lang.Object

В составе класса **Class** определен ряд простых методов-запросов, позволяющих проверить тип объекта Class:

1. **public boolean isInterface()**

Возвращает true, если объект Class представляет интерфейс.

2. **public boolean isArray()**

Возвращает true, если объект Class представляет массив.

3. **public boolean isPrimitive()**

Возвращает true, если объект Class представляет один из простых типов или служебное слово void.

4. **public Class[] getInterfaces()**

Возвращает массив объектов Class, элементы которого соответствуют интерфейсам, реализуемым текущим типом.

Если интерфейсов, базовых по отношению к типу, не существует — из-за того, например, что тип не реализует интерфейсы непосредственно или является простым, — возвращается массив нулевой длины.

5. **public Class getSuperclass()**

Возвращает объект Class, который соответствует классу, базовому по отношению к текущему типу.

Метод возвращает null, если текущий объект Class представляет класс Object, интерфейс, простой тип или служебное слово void (все они не обладают базовыми классами).

Если текущим типом является массив, возвращается объект Class для класса Object.

6. **public int getModifiers()**

Возвращает модификаторы, используемые в объявлении типа, в виде кодированного целочисленного значения. Значение должно декодироваться с помощью констант и методов класса Modifier.

К числу модификаторов типа относятся модификаторы доступа (public, protected и private) и признаки abstract, final и static.

Для удобства факт принадлежности типа к категории интерфейсов также кодируется, для чего применяется константа INTERFACE.

7. **public Class getComponentType()**

Возвращает объект класса Class, который представляет компонентный тип массива, описываемого текущим объектом Class.

Если текущий объект Class не соответствует массиву, возвращается null.

Пусть, например, задан массив значений int; тогда метод getClass возвратит объект Class, для которого метод-запрос isArray вернет значение true, а метод getComponentType — объект int.class.

8. **public static Class forName(String name, boolean initialize, ClassLoader loader) throws ClassNotFoundException**

Возвращает объект Class, соответствующий названному по имени классу или интерфейсу, с помощью заданного загрузчика классов. Используя полное имя класса или интерфейса, переданное в качестве параметра name, метод предпринимает попытку обнаружить и загрузить класс или интерфейс.

Для загрузки класса или интерфейса применяется заданный загрузчик классов.

Если значение параметра **loader** равно null, используется системный загрузчик классов.

Класс инициализируется только при условии, что значение параметра **initialize** равно **true** и ранее инициализация класса не выполнялась.

Если класс с заданным именем не может быть найден, выбрасывается объявляемое исключение типа **ClassNotFoundException**.

Имена, передаваемые в метод forName должны быть полными, т.е. содержать наименование пакета и собственное имя класса, как, например, java.util.HashMap или java.lang.Object.

Для представления имен типов массивов используется специальная система обозначений — символ [сопровождается специальным кодом, соответствующим определенному компонентному типу массива.

Компонентные типы кодируются согласно следующей схеме.

Код	Тип
B	byte
C	char
D	double
F	float
I	int
J	long
L *Имякласса*	class или interface
S	short
Z	boolean

Например, массив элементов типа int получит имя [I, а массив объектов типа Object — [Ljava.lang.Object; Многомерный массив — это массив, компонентный тип которого также является массивом; поэтому, например, тип, объявленный как int[][], получит имя [[I .

9. **public Constructor[] getConstructors()**

   public Field[] getFields()

   public Method[] getMethods()

   public Class[] getClasses()

С помощью данных методов можно получить сведения о public-членах, которые либо объявлены в текущем классе (или интерфейсе), либо унаследованы.

10. **public Constructor[ ] getDeclaredConstructors()**

public Field[ ] getDeclaredFields()

public Method[ ] getDeclaredMethods()

public Class[ ] getDeclaredClasses()

С помощью данных методов можно получить данные о членах (не обязательно обозначенных как public) определенной категории, которые объявлены в текущем классе (или интерфейсе), но не унаследованы.

11. **public Field getField(String name)**
public Field getDeclaredField(String name)

С помощью данных методов можно получить сведения о конкретном члене класса.

Первый метод находит public-поле, объявленное в текущем классе или унаследованное от базовых типов, а второй возвращает только такое поле (не обязательно обозначенное как public), которое принадлежит текущему классу.

Если поле с заданным именем не может быть найдено, выбрасывается исключение типа NoSuchFieldException.

2. **public Method getMethod(String name,**
Class[] parameterTypes)

public Method getDeclaredMethod(String name,
Class[] parameterTypes)

С помощью данных методов можно получить информацию о конкретных методах класса,

в массиве parameterTypes передаются типы параметров метода.

13. **public Constructor getConstructor(**
Class[] parameterTypes)

public Constructor getDeclaredConstructor(
Class[] parameterTypes)

Аналогично предыдущим методам.

Перед выполнением каждого из рассмотренных выше методов осуществляется проверка полномочий доступа — все методы взаимодействуют с менеджером безопасности, установленным в системе.

Если менеджер безопасности не подключен, разрешается использование всех методов без каких-либо ограничений.

Обычно менеджеры безопасности позволяют любому коду вызывать методы для получения информации о public-членах типа— это соответствует правилам нормального уровня доступа, принятым в языке.

Однако возможности обращения к членам, не обозначенным признаком public, как правило, ограниченны — получение подобных данных разрешено коду, обладающему привилегиями внутри системы.

Стоит заметить, что менеджеры безопасности способны различать только уровни доступа public и не-public — для распознавания членов, имеющих признак доступа уровня пакета или protected, информации не достаточно.

Если доступ запрещен, выбрасывается исключение типа SecurityException.

Рассмотрим пример:

```
import Java.lang.reflect.*;
public class ClassContents {
  public static void main(String[] args){
     try {
      Class c = class.forName(args[0]);
      System.out.println(c);
      printMembers(c.getFields());
      printMembers(c.getConstructors()) ;
      printMembers(c.getMethods());}
    catch(ClassNotFoundException e)  {
     System.out.println("Неизвестный класс:   " + args[0]); }
  }
  private static void printMembers(Member[] mems){
          for(int i=0; i< mems.length;  i++)  {
            if (mems[i].getDeclaringClass() == Object.class)
                  continue;
            String decl  = mems[i].toString();
            System.out.print("   ");
            System.out.println(decl);
          }
    }
}
```

Классы Field, Constructor,Method

Каждый из классов Field, Constructor,Method реализует интерфейс Member, в составе которого объявлены три метода, общие для всех категорий членов классов.

1. **Class getDeclaringClass()**

Возвращает объект Class, соответствующий классу, в котором объявлен текущий член.

2. **String getName()**

Возвращает наименование текущего члена.

3. **int getModifiers()**

Возвращает модификаторы, используемые в объявлении члена, в виде кодированного целочисленного значения. Значение должно декодироваться с помощью констант и методов класса Modifier