Конспект лекций по Java. Занятие 17

В.Фесюнов

1. Сериализация объектов

Разберемся с понятиями **object persistence** и **serialization** .

В программе мы создаем объекты, которые существуют до тех пор, пока существует, по крайней мере, одна ссылка на данный объект. Если все ссылки на объект уничтожены, то объект, хотя и существует какое-то время, пока его не утилизировал сборщик мусора, но для нас он безвозвратно потерян. По завершении программы все созданные в программе объекты уничтожаются.

В теории программирования существует понятие *object persistence* (постоянство объектов). Это свойство объектов существовать вне зависимости от программы. Другими словами, понятие *object persistence* относится к ситуации, когда время существования объектов превышает время работы программы.

На практике это означает следующее. Если мы обеспечим (в программе) сохранение образов объектов в некотором файле или в базе данных, а также обеспечим возможность их воссоздания при следующих запусках программы, то мы тем самым обеспечим *object persistence* . Естественно, это несколько упрощенный взгляд на данное понятие.

Вопросы сохранения объектов в базе данных — это отдельная тема и мы ее здесь затрагивать не будем. Мы сейчас рассмотрим средства Java, которые могут использоваться для простейшего обеспечения постоянства объектов путем, например, их сохранения в файле.

Для этого и подобных случаев язык Java имеет простое и, что очень важно, **стандартное** средство решения вопросов постоянства объектов, которое называется механизмом *сериализации (serialization)* .

Copyright © 1999-2004 Javable.com. All rights reserved.

Как указывалось, этот механизм очень прост, а с другой стороны имеет массу деталей и нюансов, которые подробно описаны в технической документации. Здесь просматривается противоречие в утверждениях (простота и наличие массы деталей). На самом деле противоречия нет. Механизм прост в своей основе, а детали и нюансы могут потребоваться в очень редких специальных случаях, которые на практике встречаются крайне редко.

Поэтому мы рассмотрим подробно основы и наиболее существенные детали.

1.1. Интерфейс Serializable

Механизм сериализации основан на интерфейсе `java.io.Serializable`. Простое указание "implements Serializable" в объявлении класса и выполнение ряда простых требований позволяет сохранять/восстанавливать объекты этого класса стандартным образом.

При сохранении объектов применяется класс `ObjectOutputStream` , при восстановлении — `ObjectInputStream`.

Рассмотрим абстрактный пример. Пусть у нас в программе используется некоторый класс `A` и нам нужно обеспечить сохранение/восстановление объектов этого класса.

Тогда сам класс должен быть описан примерно так:

```
public class A implements Serializable {
       . . .
}
```

Сохранение объектов класса `A` в файле может быть реализовано следующим образом.

Пусть у нас есть объект класса `A` и на него ссылается переменная `aObj` , описанная как

```
  A aObj;
```

Тогда следующий фрагмент обеспечит запись этого объекта в файл "A.ser".

```
ObjectOutputStream out = null;
try {
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
     out = new ObjectOutputStream(new BufferedOutputStream(
                                 new FileOutputStream("A.ser")));
     out.writeObject(aObj);
} catch ( IOException ex ) {
   ex.printStackTrace();
}
```

Для восстановления объекта из файла может быть использован следующий фрагмент:

```
ObjectInputStream in = null;
A restObj = null;
try {
    in = new ObjectInputStream(new BufferedInputStream(
                              new FileInputStream("A.ser")));
    restObj = (A)in.readObject();
} catch ( IOException ex ) {
   ex.printStackTrace();
}
```

Как видим, все достаточно просто.

Сам по себе интерфейс Serializable не имеет никаких методов и служит лишь для указания того, что мы собираемся сохранять и восстанавливать объекты, используя описанные выше возможности.

Данный пример демонстрирует лишь общую схему описания и использования сериализации. Имеется ряд дополнительных требований и опциональных возможностей сериализации, о которых нужно знать.

Во-первых, нужно сказать об ограничениях касающихся внутренних объектов сериализуемых классов. Если класс состоит из полей элементарных типов и `String` , то проблем нет. Но, если в состав класса `A` входит поле класса `B` , то класс `B` тоже должен быть сериализуемым, т.е. удовлетворять удовлетворять интерфейсу `Serializable` . Если это не так, то в процессе сериализации возникнет исключение `NotSerializableException` .

Во-вторых, в процессе сериализации/десериализации не участвуют статические поля класса, поскольку они фактически являются не полями объектов (экземпляров класса),

Copyright © 1999-2004 Javable.com. All rights reserved.

а полями класса в целом.

И, в-третьих, нужно учитывать влияние наследования на сериализацию. Пусть класс `A` объявлен, как сериализуемый. На его базе создан класс `B` , для которого не указано `implements Serializable` . И, наконец, от `B` порожден класс `C` тоже сериализуемый. Тогда при сериализации будут сохранены все поля классов `A` и `C` , но не `B` .

1.2. Классы ObjectOutputStream и ObjectInputStream

При сохранени объектов в файле при помощи сериализации используется некоторый внутренний формат. Этот формат обеспечивает сохранение информации о классе объекта и позволяет восстанавливать объект нужного класса по содержимому, прочитанному из потока `ObjectInputStream` . Знание деталей этого формата для использования сериализации не требуется.

Рассмотрим основные методы классов `ObjectOutputStream` и `ObjectInputStream` .

В классе `ObjectOutputStream` метод `writeObject` имеет следующее описание:

```
public final void writeObject(Object obj) throws IOException
```

Данный метод может записать в поток любой объект, поскольку в качестве параметра указан `Object` . При этом, однако, если класс этого объекта не удовлетворяет интерфейсу Serializable, то возникнет **NotSerializableException** .

В классе `ObjectInputStream` метод `readObject` имеет такое описание:

```
public final Object readObject() throws OptionalDataException,
    ClassNotFoundException, IOException
```

Он считывает из потока и создает в памяти объект того класса, который записан в потоке. В качестве результата он возвращает ссылку на считанный объект. Полученную ссылку мы можем привести к нужному типу (*downcasting*), после чего использовать как любую другую ссылку на объект заданного типа. При чтении объекта может возникнуть исключительная ситуация `ClassNotFoundException` , если класс прочитанного объекта неизвестен.

Copyright © 1999-2004 Javable.com. All rights reserved.

1.3. Промежуточные данные

В состав полей класса иногда входят поля, выполняющие служебную роль, не связанные с основной информацией, хранимой в объектах данного класса.

Например, класс может содержать поле state (состояние), которое устанавливается в зависимости от истории обработки такого объекта. Такие поля не требуется запоминать при сохранении объекта класса (хотя иногда требуется установка значения таких полей после восстановления объекта).

Для того чтобы такие поля не сохранялись, можно описать их с описателем transient (временный), например:

```
public class A implements Serializable {
        . . .
        private transient int state = 0;
        . . .
}
```

Методы чтения/записи классов ObjectInputStream/ObjectOutputStream распознают такие поля и не обрабатывают их при сериализации/десериализации объекта. Т.е. при сериализации объекта поля с описателем transient не сохраняются, а при десериализации — не восстанавливаются.

Еще одно очень важное замечание.

- Многие из классов стандартной библиотеки Java удовлетворяют интерфейсу Serializable. К числу таких классов относятс String, все классы коллекций из java.util и многие другие.

1.4. Пример

Этот пример является демонстрационным. Он показывает, как можно сохранить и восстановить список объектов. Список в данном случае строится при помощи класса ArrayList.

Кроме того, на этом примере мы рассмотрим один из специальных приемов,

Copyright © 1999-2004 Javable.com. All rights reserved.

используемых при восстановлении значений transient-полей.

Приложение SerializableDemo оперирует объектами класса DemoObject и состоит из двух файлов: SerializableDemo.java и DemoObject.java.

Файл SerializableDemo.java:

```
import java.util.*;
import java.io.*;

public class SerializableDemo {

    public static void main(String args[]) {

//--- 1. Создадим ArrayList из 20 элементов DemoObject
        ArrayList list = new ArrayList();
        Random r = new Random();
        for ( int i = 0; i < 20; i++ ) {
            DemoObject obj = new DemoObject(r.nextInt()%1000);
            list.add(obj);
        }
//--- 2. Добавим еще один элемент в 10-ю позицию списка
        list.add(10, new DemoObject(1111));
//--- 3. Распечатаем результат
        print("Исходный список", list);
//--- 4. Сохраним это в файле
        ObjectOutputStream out = null;
        try {
             out = new ObjectOutputStream(new BufferedOutputStream(
                                          new FileOutputStream("Demo.ser")));
             out.writeObject(list);
        } catch ( IOException ex ) {
             ex.printStackTrace();
        } finally {
             if ( out != null )
                 try {
                     out.close();
                 } catch ( IOException ex ) {
                     ex.printStackTrace();
                 }
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
        }
//--- 5. Восстановим все из файла
        DemoObject.dropCounter();     // сброс счетчика объектов
        ObjectInputStream in = null;
        try {
            in = new ObjectInputStream(new BufferedInputStream(
                                       new FileInputStream("Demo.ser")));
            list = (ArrayList)in.readObject();
        } catch ( IOException ex ) {
            ex.printStackTrace();
        } catch ( Exception ex ) {
            ex.printStackTrace();
        } finally {
             if ( in != null )
                 try {
                     in.close();
                 } catch ( IOException ex ) {
                     ex.printStackTrace();
                 }
        }
//--- 6. Отпечатаем результат
        print("Восстановленный список", list);
    }

    static void print(String title, List list) {
        System.out.println(title);
        Iterator iter = list.iterator();
        for ( int i = 0; iter.hasNext(); i++ ) {
            System.out.println("N "+i+"="+iter.next());
        }
    }
}
```

Файл DemoObject.java:

```
import java.io.*;

public class DemoObject implements Serializable {
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
    private static int counter = 0;

    private transient int cnt = 0;
    private String number;

    public DemoObject(int numb) {
        cnt = counter++;
        number = String.valueOf(numb);
    }

    public DemoObject() {
        number = null;
    }

    public String toString() {
        return " "+cnt+": "+number;
    }

    public static void dropCounter() {
        counter = 0;
    }

    private void readObject(ObjectInputStream in)
                     throws IOException, ClassNotFoundException {
        in.defaultReadObject();
        cnt = counter++;
    }
}
```

Действия программы описаны в комментариях в файле SerializableDemo.java. Требует пояснения только реализация класса DemoObject.

Но сначала обратим внимание на основные моменты головной программы.

Самое главное здесь то, что весь список сохраняется одним оператором -

out.writeObject(list);

и восстанавливается также одним оператором -

list = (ArrayList)in.readObject();

Copyright © 1999-2004 Javable.com. All rights reserved.

Рассмотрим теперь сам сериализуемый объект (класс DemoObject).

В нем три поля.

- Статическое поле counter для счетчика обектов. Поскольку оно статическое, то в процессе сериализации/десериализации не участвует.
- Временное (transient) поле cnt — хранит порядковый номер объекта (объекты нумеруются в порядке их создания в программе). Поскольку оно описано как transient, то оно тоже не сериализуется.
- Поле number, которое демонстрирует содержимое объекта.

Для работы с полем counter в классе создан метод dropCounter(), который применяется в основной программе тогда, когда нужно сбросить счетчик порожденных объектов класса DemoObject.

Для установки значения поля cnt в классе DemoObject предприняты специальные усилия. Для этого определен метод

private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException;

Наличие такого метода в сериализуемом классе гарантирует, что он будет вызван при десериализации объекта. При этом в таком методе собственно считывание объекта из потока следует выполнять при помощи специального метода класса ObjectInputStream — defaultReadObject(). Т.е. метод readObject(...) должен состоять из вызова метода defaultReadObject() и какого-то дополнительного кода. Этот код, в частности, может содержать установку значений transient-полей, что и продемонстрировано на данном примере.

Следует отметить, что метод defaultReadObject() может быть вызван только из метода readObject(...), в противном случае возникает NotActiveException.

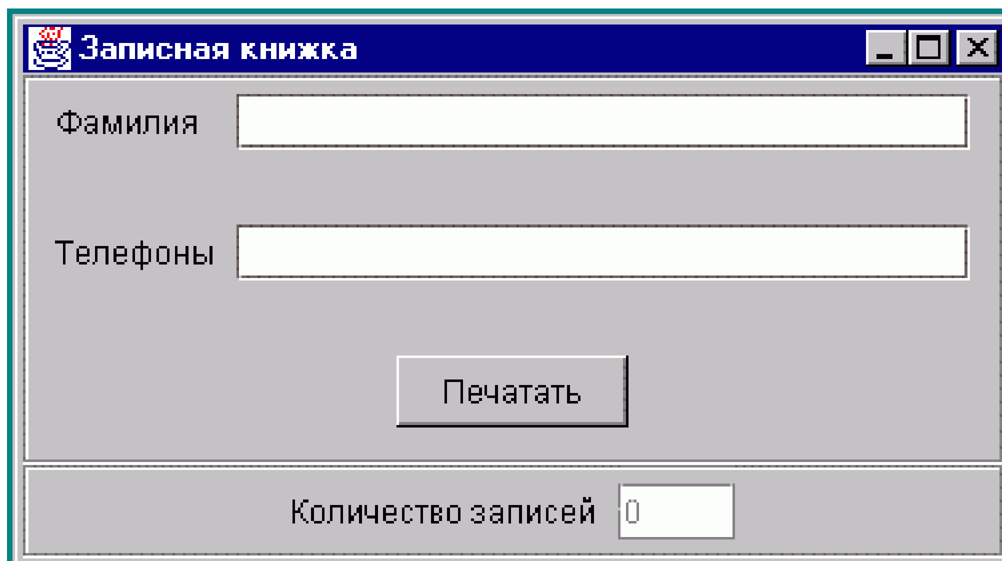
Оттранслируем и запустим программу.

Результаты ее работы показывают, что сам список объектов восстановлен правильно, он состоит из тех же объектов, расположенных в том же порядке. Но поля cnt в этих двух списках отличаются, как это и должно быть по смыслу этого поля.

Copyright © 1999-2004 Javable.com. All rights reserved.

2. Практическая работа

На 11-м занятии мы построили приложение для ведения простейшей записной книжки. Это приложение имело следующий внешний вид диалога



Продолжим работу над этим приложением.

Сейчас мы уже можем

1. Организовать диалог выбора файла для чтения/сохранения записной книжки.

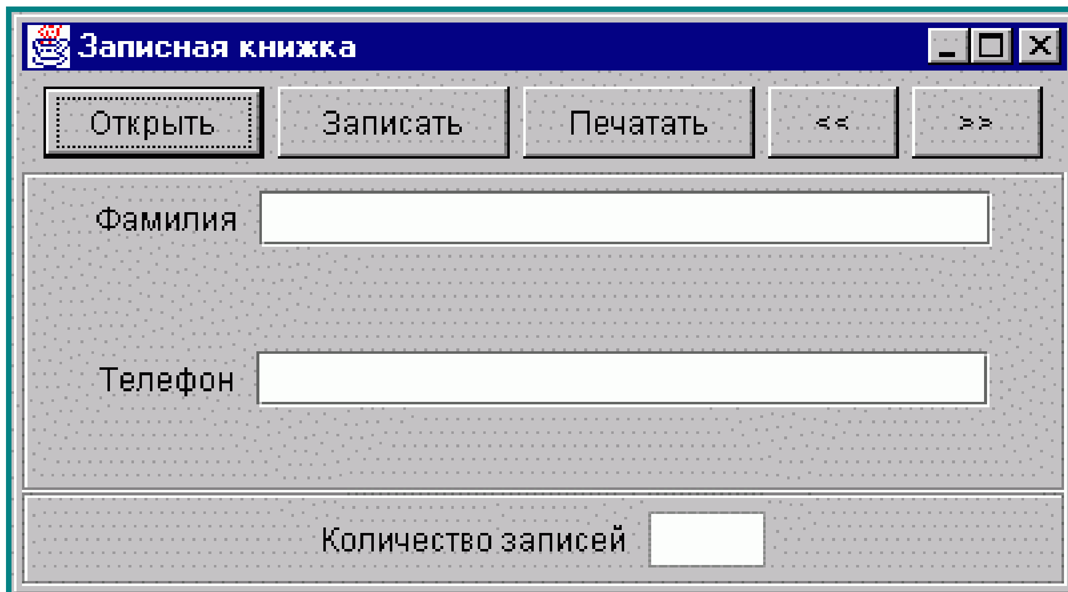
2. Реализовать собственно запись в выбранный файл или чтение из выбранного файла используя механизм сериализации.

Кроме того, для диалогового просмотра записной книжки можно реализовать простейший диалог перехода на следующую/предыдущую запись.

- Более сложный диалог с просмотром списка записей мы рассмотрим позже.

Внешний вид диалога нашего приложения должен быть следующим.

Copyright © 1999-2004 Javable.com. All rights reserved.



Copyright © 1999-2004 Javable.com. All rights reserved.