Chapter 06. 객체 스트림

Originally made by Prof. Hanku Lee
Modified by Mingyu Lim

Collaborative Computing Systems Lab.
School of Internet & Multimedia Engineering
Konkuk University, Seoul, Korea

1. 첫번째 예제 : 생각해볼 문제와 간단한 예제

❑ 객체 스트림 개념 이해전에 생각해볼 문제 제시(Vector 예제)

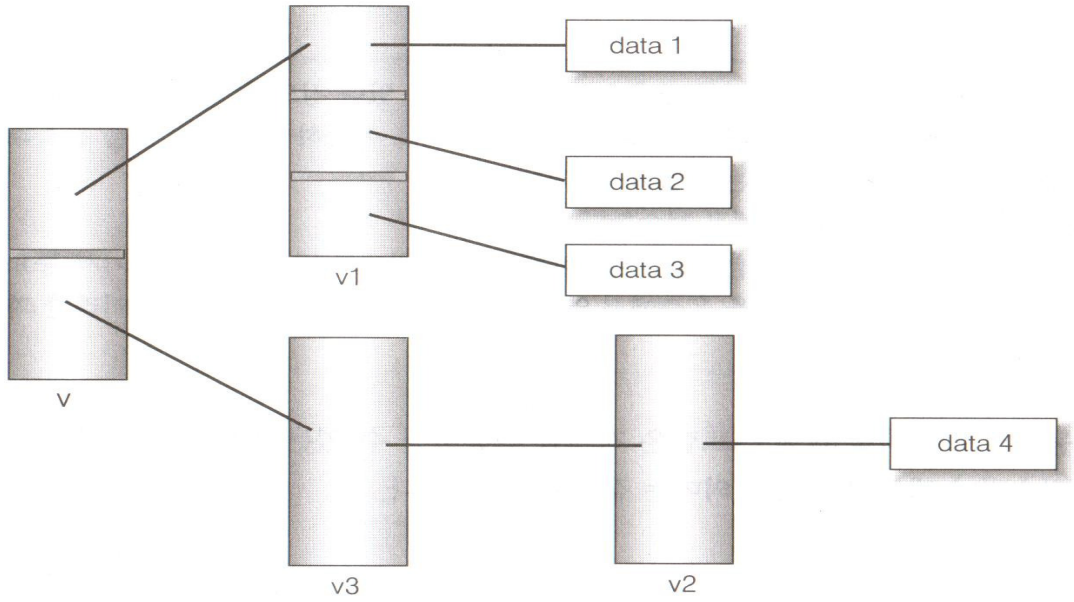
[예제 6-1] 객체 스트림의 간단한 예제

```
Vector v = new Vector();
Vector v1 = new Vector();
Vector v2 = new Vector();
Vector v3 = new Vector();
v1.addElement(new String("data 1"));
v1.addElement(new String("data 2"));
v1.addElement(new String("data 3"));
v2.addElement(new String("data 4"));
v3.addElement(v2);
v.addElement(v1);
v.addElement(v3);
```

• 위의 예제를 그림으로 도식화 해보아라.

1. 첫번째 예제 : 생각해볼 문제와 간단한 예제

❑ [예제 6-1] 그림으로 도식화



[그림 6-2] 객체 스트림의 간단한 예제를 실행했을 때의 메모리 상태

- 메모리에 저장까지는 어렵지 않으나 다시 읽어 들여 원래 형태로 구성하려면 메모리에 있는 객체의 형태가 복잡할수록 어려운 작업이 된다.

1. 첫번째 예제 : 생각해볼 문제와 간단한 예제

❑ 객체스트림의 간단한예제 (Vector 객체 저장)

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.Vector;

public class ObjectStreamTest1 {

    public static void main(String[] args) {
        FileOutputStream fout = null;
        ObjectOutputStream oos = null;
        Vector v = new Vector();
        Vector v1 = new Vector();
        Vector v2 = new Vector();
        Vector v3 = new Vector();
        v1.addElement(new String("data 1"));
        v1.addElement(new String("data 2"));
        v1.addElement(new String("data 3"));
        v2.addElement(new String("data 4"));
        v3.addElement(v2);
        v.addElement(v1);
        v.addElement(v3);

        try{
            fout = new FileOutputStream("object.dat");
            oos = new ObjectOutputStream(fout);

            oos.writeObject(v);
            oos.reset();
            System.out.println("저장되었습니다.");

        }catch(Exception ex){
        }finally{
            try{
                oos.close();
                fout.close();
            }catch(IOException ioe){}
        } // finally
    } // main end
} // class end
```

1. 첫번째 예제 : 생각해볼 문제와 간단한 예제

❑ 객체스트림의 간단한예제 (Vector 객체 읽기)

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.Vector;

public class ObjectStreamTest2 {

    public static void main(String[] args) {
        FileInputStream fin = null;
        ObjectInputStream ois = null;

        try{
            fin = new FileInputStream("object.dat");
            ois = new ObjectInputStream(fin);

            Vector v = (Vector)ois.readObject();
            Vector v1 = (Vector)v.get(0);
            String d1 = (String)v1.get(0);
            String d2 = (String)v1.get(1);
            String d3 = (String)v1.get(2);
            Vector v3 = (Vector)v.get(1);
            Vector v2 = (Vector)v3.get(0);
            String d4 = (String)v2.get(0);
            System.out.println(d1);
            System.out.println(d2);
            System.out.println(d3);
            System.out.println(d4);
        }catch(Exception ex){
        }finally{
            try{
                ois.close();
                fin.close();
            }catch(IOException ioe){}
        } // finally
    } // main end
} // class end
```

```
Console
<terminated> ObjectStreamTest2 [Java Application] C:₩Program Files₩Java₩jdk1.6.0_02₩bin₩javaw.exe (2007. 1
data 1
data 2
data 3
data 4
```

2. 객체 스트림 소개

❑ 객체 스트림 이란?

- 객체를 아주 쉽게 전송할수 있는 방법
- 객체가 전송된다는 의미 : 객체가 내부적으로 참조하는 객체들 역시 함께 전송

객체가 전송되거나 쓴다고 하면, 보통 데이터만 해당된다고 생각할수 있다 객체가 전송된다는 것은 애플릿(Applet), 윈도우 등도 전송될수 있다는 것을 의미한다.

❑ 객체 전송 순서

1. 데이터를 특정 통신 채널로 보낼수 있는 형태로 바꾼다.(마샬링)
2. 변환된 데이터를 전송하거나 쓴다.
3. 변환된 데이터를 읽어 들여 원래의 형태로 변환한다.(언마샬링)

2. 객체 스트림 소개(계속)

❑ 객체 전송 순서

2. 객체 스트림 소개(계속)

❑ 마샬링과 언마샬링

- 마샬링(객체 직렬화 : Object Serealization)
  - 데이터를 바이트의 흐름으로 만들어 TCP와 같은 통신 채널을 통해서 전송하거나 스트림으로써 줄수 있는 형태로 바꾸는 과정
  - 마샬링 될수 있는 자료형
    : 기본자료형 과 java.io.Serializable 인터페이스 구현한 객체만 가능
  - 객체 스트림인 ObjectOutputStream에 의해 제공되어지고 이에 의해 객체는 직렬화 되어 전달
  - writeObject() 메소드를 이용해서 객체를 직렬화

- 전송
  - 데이터를 발신지에서 목적지로 전달하는 과정
  - 객체 스트림은 바이트 기반의 표준 스트림을 이용
  - 네트워크에 대한 전송이외에도 파일에 쓰기를 할경우에도 객체 스트림은 바이트 기반의 표준 스트림 이용

2. 객체 스트림 소개(계속)

❑ 마샬링과 언마샬링

- 언마샬링(객체 역직렬화 : deserialization)

  - 전송받은 데이터를 원래의 형태로 변환하는 과정
    (전달받은 데이터는 원래 형태로 변환될수 있는 충분한 내용을 포함)
  - 역직렬화라고 ObjectInputStream에 의해 제공
  - readObject() 메소드를 이용해서 객체를 역직렬화 한다.

3. 두번째 예제 : 나의 책 목록

❑ 예제전 선행 내용

- 책 목록은 java.util.ArrayList에 저장
- 책의 정보를 저장하기 위한 Book 클래스 생성
  (마샬링 될수있도록 해당클래스는 Serializable 인터페이스 구현)
- Book 클래스 필드 : isbn, 제목, 저자, 가격정보 포함
  (String, String, String, int 형으로 선언)
- 마샬링을 하려면 ObjectOutputStream 이용
- 언마샬링을 하려면 ObjectInputStream 이용
- 객체를 직렬화해서 저장할 파일명은 booklist.dat 이다

3. 두번째 예제 : 나의 책 목록(계속)

❑ Book.java(예제 6-4)

```
import java.io.Serializable;

public class Book implements Serializable {
    private String isbn;
    private String title;
    private String author;
    private int price;

    public Book(String isbn, String title, String author, int price){
        this.isbn = isbn;
        this.title = title;
        this.author = author;
        this.price = price;
    }

    public String getAuthor() {
        return author;
    }
    public String getIsbn() {
        return isbn;
    }
    public int getPrice() {
        return price;
    }
    public String getTitle() {
        return title;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
    public void setPrice(int price) {
        this.price = price;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String toString(){
        return getIsbn() + "," + getTitle() + "," + getAuthor() + "," + getPrice();
    }
```

1. Book클래스 마샬링 되기 위해서 Serializable 인터페이스 구현
2. 필드는 String 형과 기본형인 int로 구성
3. 객체를 직렬화 할수 있다는 것은 객체에 있는 필드값과 필드가 참조하는 객체들이 마샬링된다는 것을 의미

3. 두번째 예제 : 나의 책 목록(계속)

❑ BookObjectOutputTest.java(예제 6-5)

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

public class BookObjectOutputTest{

    public static void main(String[] args) {
        FileOutputStream fout = null;
        ObjectOutputStream oos = null;

        ArrayList list = new ArrayList();
        Book b1 = new Book("a0001", "자바완성", "홍길동", 10000);
        Book b2 = new Book("a0002", "스트럿츠", "김유신", 20000);
        Book b3 = new Book("a0003", "기초 EJB", "김성박", 25000);
        list.add(b1);
        list.add(b2);
        list.add(b3);

        try{
            fout = new FileOutputStream("booklist.dat");
            oos = new ObjectOutputStream(fout);

            oos.writeObject(list);

            System.out.println("저장되었습니다.");

        }catch(Exception ex){
        }finally{
            try{
                oos.close();
                fout.close();
            }catch(IOException ioe){}
        } // finally
    } // main end
} // class end
```

1. 책 3권에 대한정보를 **ArrayList** 에 저장한후 객체스트림을 이용해서 **booklist.dat** 파일에 저장한다.

3. 두번째 예제 : 나의 책 목록(계속)

❑ BookObjectInputTest.java(예제 6-6)

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.ArrayList;

public class BookObjectInputTest{

    public static void main(String[] args) {
        FileInputStream fin = null;
        ObjectInputStream ois = null;

        try{
            fin = new FileInputStream("booklist.dat");
            ois = new ObjectInputStream(fin);

            ArrayList list = (ArrayList)ois.readObject();
            Book b1 = (Book)list.get(0);
            Book b2 = (Book)list.get(1);
            Book b3 = (Book)list.get(2);

            System.out.println(b1.toString());
            System.out.println(b2.toString());
            System.out.println(b3.toString());

        }catch(Exception ex){
        }finally{
            try{
                ois.close();
                fin.close();
            }catch(IOException ioe){}
        } // finally
    } // main end
} // class end
```

1. **readObject()** 메소드를 이용해서 저장된 객체를 읽어들인다.
2. 저장된 객체가 **ArrayList**였기때문에 원래의 형태로 형변환했다.

```
Console
<terminated> BookObjectInputTest [Java Application] C:₩Program Files₩Java₩jdk1.6.0_02₩bin₩javaw.exe (2007. 10. 02 오후 6:21:49)
a0001,자바완성,홍길동,10000
a0002,스트럿츠,김유신,20000
a0003,기초 EJB,김성박,25000
```

4. 마샬링하고 싶지 않은 필드에 대한 처리

❑ 예제

```
import java.io.Serializable;

public class Book implements Serializable {
    private String isbn;
    private String title;
    transient private String author;
    private int price;

    public Book(String isbn, String title, String author, int price){
        this.isbn = isbn;
        this.title = title;
        this.author = author;
        this.price = price;
    }

    public String getAuthor() {
        return author;
    }
    public String getIsbn() {
        return isbn;
    }
    public int getPrice() {
        return price;
    }
    public String getTitle() {
        return title;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
    public void setPrice(int price) {
        this.price = price;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String toString(){
        return getIsbn() + "," + getTitle() + "," + getAuthor() + "," + getPrice();
    }

}
```

변경된 코드

1. 특별한 경우에 필드가 마샬링 되지 않기를 원할때 사용 (예 : 보안상 중요한 필드)
2. 자바 키워드인 **transient** 사용

```
Console
<terminated> BookObjectInputTest [Java Application] C:₩Program Files₩Java₩jdk1.6.0_02₩bin₩javaw.exe (2007. 10. 02 오후 6:23:10)
a0001,자바완성,null,10000
a0002,스트럿츠,null,20000
a0003,기초 EJB,null,25000
```

5. 세번째 예제 : 윈도우 저장과 읽기

❑ 윈도우 저장과 읽기

객체 직렬화가 불가능한 객체는 전송되지 않는다.

❑ 예제전 선행 내용

- 윈도우 객체인 HelloWindow.java 를 작성한다.
- 객체 직렬화 기술을 이용해서 HelloWindow를 파일로 저장할 WindowObjectOutputStreamTest.java 를 작성
- 파일에 저장된 윈도우 객체를 읽어 들일 WindowObjectInputStreamTest.java 를 작성

5. 세번째 예제 : 윈도우 저장과 읽기(계속)

❑ HelloWindow.java(예제 6-7)

```
import java.io.Serializable;
import java.awt.event.*;
import java.awt.*;

public class HelloWindow extends Frame implements Serializable{

    public HelloWindow(){
        super("Hello Window");
        setLayout(new BorderLayout());
        add("Center", new Label("Hello Window"));
        addWindowListener(new WindowEventHandler());
        setSize(200, 200);
    }


    class WindowEventHandler extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            System.out.println("윈도우를 종료합니다.");
            System.exit(0);
        }
    }
}
```

1. 실행된 상태에서 윈도우 x 버튼을 클릭할 경우 "윈도우를 종료합니다." 메시지를 출력하고 종료된다.

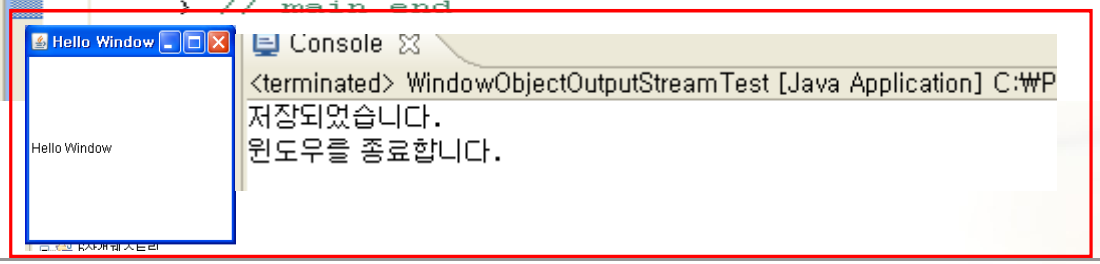
5. 세번째 예제 : 윈도우 저장과 읽기(계속)

❑ WindowObjectOutputStreamTest.java(예제 6-8)

ObjectTest.java | ObjectStreamTest1.java | ObjectStreamTest2.java | B

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class WindowObjectOutputStreamTest {
    public static void main(String[] args) {
        FileOutputStream fout = null;
        ObjectOutputStream oos = null;

        HelloWindow hwin = new HelloWindow();

        try{
            fout = new FileOutputStream("hwin.dat");
            oos = new ObjectOutputStream(fout);

            oos.writeObject(hwin);
            oos.reset();

            System.out.println("저장되었습니다.");
            hwin.setVisible(true);
        }catch(Exception ex){
        }finally{
            try{
                oos.close();
                fout.close();
            }catch(IOException ioe){}
        } // finally
    } // main end
```

1. 실행된 상태에서 윈도우 x 버튼을 클릭할 경우 "윈도우를 종료합니다." 메시지를 출력하고 종료된다.



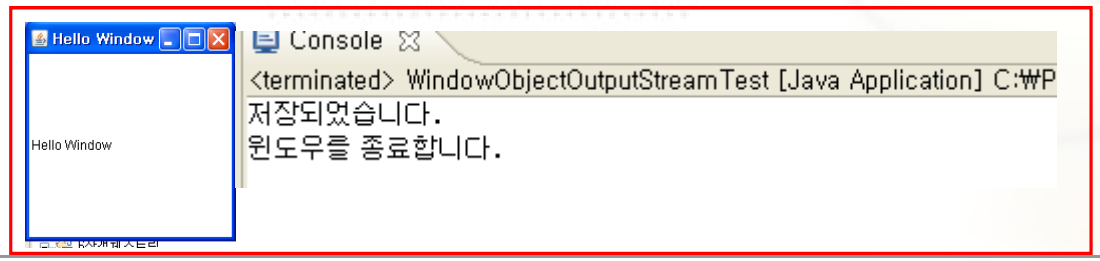
5. 세번째 예제 : 윈도우 저장과 읽기(계속)

❑ WindowObjectInputStreamTest.java(예제 6-9)

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class WindowObjectInputStreamTest {

    public static void main(String[] args) {
        FileInputStream fin = null;
        ObjectInputStream ois = null;

        try{
            fin = new FileInputStream("hwin.dat");
            ois = new ObjectInputStream(fin);

            HelloWindow hwin = (HelloWindow)ois.readObject();
            hwin.setVisible(true);
        }catch(Exception ex){
        }finally{
            try{
                ois.close();
                fin.close();
            }catch(IOException ioe){}
        } // finally
    } // main end
}
```

1. 결과는 예제 6-8과 동일하다.
2. 그러나 x 버튼을 클릭해도 윈도우 종료되지 않는다.
3. WindowEventHandler 객체가 java.io.Seriable 인터페이스를 구현하지 않았기 때문이다.
   * java.io.Serializable 인터페이스를 구현하지 않으면 마샬링되지 않는다.

***위의 문제 해결* (HelloWindow.java 에서 수정)**
Class WindowEventHandler extends WindowAdapter
-> Class WindowEventHandler extends WindowAdapter implements Serializable



6. ObjectOutputStream클래스의 생성자와 메소드

❑ ObjectOutputStream 클래스 선언

```
public class ObjectOutputStream  extends OutputStream
implements ObjectOutput, ObjectStreamConstants
```

❑ ObjectOutputStream 클래스 생성자

```
public ObjectOutputStream(OutputStream out) throws
IOException
```

- OutputStream을 상속받는 어떤 객체든지 모두 생성자로 가능

6. ObjectOutputStream클래스의 생성자와 메소드

❑ ObjectOutputStream 중요 메소드

[표 6-1] ObjectOutputStream의 중요 메소드

메소드	설명
public final void writeObject(Object obj) throws IOException	ObjectOutputStream에 있는 가장 중요한 메소드다. 인자로 지정된 객체를 직렬화해서 전송한다.
public void reset( ) throws IOException	스트림을 방금 전에 생성된 것처럼 초기 상태로 만든다. 네트워크 프로그래밍에서 사용할 때에는 writeObject( ) 메소드를 호출한 후 반드시 호출해 줘야 한다.
public void flush( ) throws IOException	전송해야 할 데이터를 버퍼링함으로써 ObjectInputStream에서 읽어 들이지 못하는 경우가 발생할 수 있다. 이런 문제를 해결하기 위해서 네트워크 프로그래밍에서 사용할 때에는 writeObject( ) 메소드를 호출한 후 flush( ) 메소드를 바로 호출해줘야 한다.

7. ObjectInputStream클래스의 생성자와 메소드

❑ ObjectInputStream 클래스 선언

```
public class ObjectInputStream  extends InputStream
implements ObjectInput, ObjectStreamContants
```

❑ ObjectInputStream 클래스 생성자

```
public ObjectInputStream(InputStream in) throws
IOException
```

- InputStream을 상속받는 어떤 객체든지 모두 생성자로 가능

7. ObjectInputStream클래스의 생성자와 메소드

❑ ObjectInputStream 중요 메소드

[표 6-2] ObjectInputStream의 중요 메소드

메소드	설명
public final Object readObject( ) throws IOException, ClassNotFoundException	ObjectInputStream에 있는 가장 중요한 메소드다. 데이터를 읽어 들여 역직렬화한다.

8. 객체스트림의 예외

❑ 객체 스트림의 예외

[표 6-3] 직렬화와 역직렬화할 경우 발생할 수 있는 중요 예외

예외	설명
InvalidCalssException	객체를 역직렬화할 경우에 전송한 객체에 대한 클래스는 존재하지만 일치하지 않을 경우에 발생한다. 그리고 객체에 대한 버전 관리가 잘못 되었을 경우에 자주 발생한다.
NotSerializableException	객체를 읽거나 쓸 경우에 발생한다. 그리고 객체(혹은 객체의 필드 중 하나)를 직렬화 할 수 없을 경우에 발생한다.
StreamCorruptedException	스트림에 포함된 객체 스트림의 데이터가 유효한 것이 아니거나 헤더 정보가 잘못되었을 경우에 발생한다.
InvalidObjectException	객체를 역직렬화한 후 검증 과정이 성공적으로 끝나지 못했을 경우에 발생한다.
WriteAbortedException	객체를 출력하는 과정에서 예외가 발생해서 제대로 출력되지 않은 객체를 다른 한쪽에 읽어 들일 때 발생한다. 이 경우에는 객체가 전송될 때 발생한 예외 내용을 포함하게 된다.