ObjectStream

경복대학교
소프트웨어융합과
배희호 교수
010-2369-4112
031-570-9600
hhbae@kbu.ac.kr

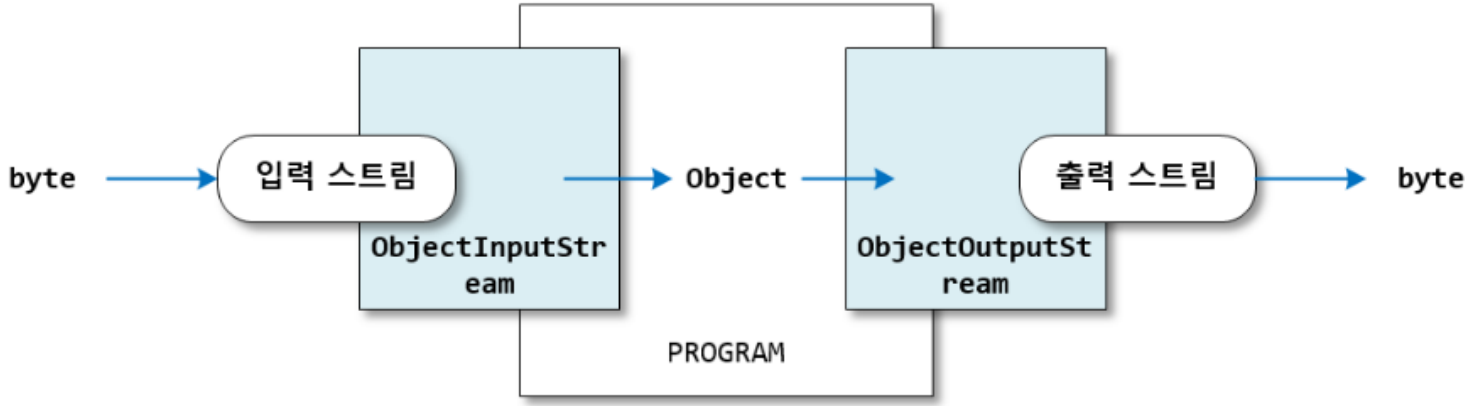
입출력 성능을 향상시키는 클래스들

- 이 클래스들은 모두 java.io 패키지에 속함
- 이 클래스들은 단독으로는 사용될 수 없음

클래스 이름	설명
DataInputStream	Primitive Type의 Data를 입출력하는 클래스
DataOutputStream	
ObjectInputStream	Primitive Type과 Reference Type의 Data를 입출력하는 클래스
ObjectOutputStream	

Object Stream

- 생성된 객체를 File 또는 Network로 출력할 수 있으며 객체는 문자가 아니기 때문에 Byte 기반 Stream으로 입출력
- 객체를 직렬화하고 다시 역직렬화 시킬 수 있는 클래스는 ObjectInputStream과 ObjectOutputStream 클래스
- ObjectInputStream과 ObjectOutputStream 클래스는 각각 InputStream과 OutputStream을 상속받지만 기반 Stream을 필요로 하는 보조 Stream



Object Stream

- 객체를 생성할 때 입출력(직렬화/역직렬화)할 Stream을 지정해주어야 함
- 사용법은 다른 Filter Stream(Buffered I/O, Data I/O)과 유사
- Object Stream의 입출력 대상이 되는 클래스는 Serializable Interface를 구현
- 클래스의 일부 Member 변수를 Serialization(직렬화) 대상에서 제외시키려면 'transient' Keyword를 사용

Serialization

- 객체의 Serialization(직렬화)는 Data들이 한 줄로 나열해서 Stream을 통해서 전송된다는 것
- 지금까지 살펴본 입출력 방식과 같이 Data들이 개별적으로 전송되는 것이 아니고 클래스 내부에 설계된 Member들이 객체 단위로 File에 기록하거나 쓴다는 의미임
- 지금부터 개별적으로 데이터를 전송할 경우 어떤 면에서 불편한 지를 살펴보고 객체 직렬화를 이용하면 어떻게 이런 불편함을 없앨 수 있는지 살펴보도록 하겠습니다.

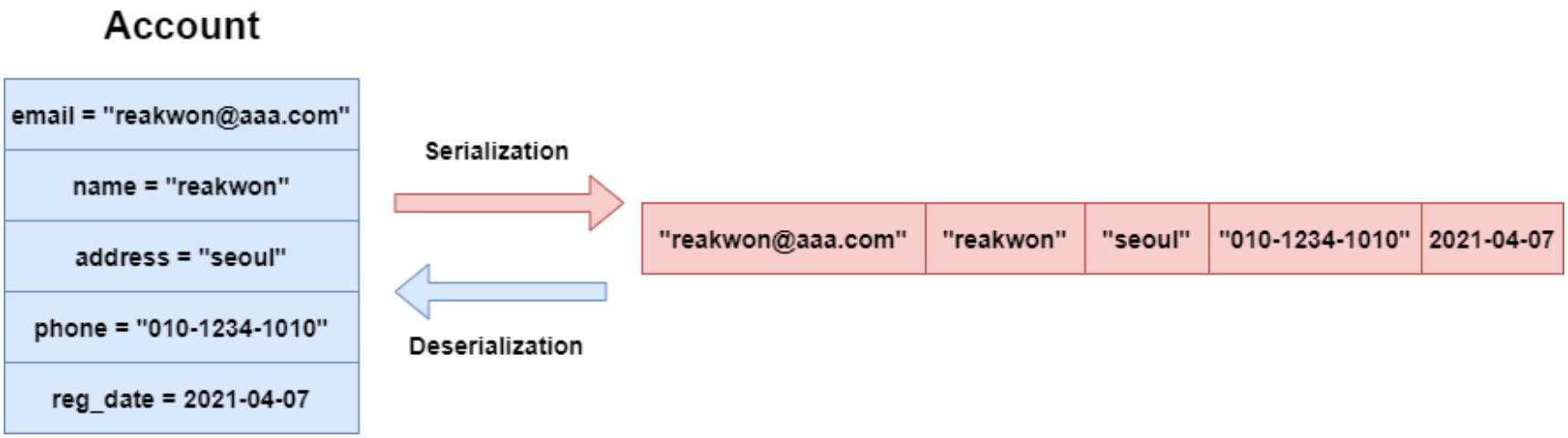
Serialization

- 객체가 가진 Data들을 순차적인 Data로 변환하는 것
- 객체를 순차적인 Byte로 표현한 Data를 의미

객체 직렬화의 개념

Serialization

- 객체를 File로 저장하거나 읽어올 수 있을까요?
  - 객체는 문자가 아니기 때문에 Byte 기반 Stream으로 입출력 해야함



Serialization

- 객체를 Data Stream(Stream에 쓰기(write)위한 연속적인 (serial) Data)으로 만드는 것
- 예) 객체를 Computer에 저장했다가 꺼내 쓰기, Network를 통한 Computer 간의 객체 전송

- Deserialization(역직렬화)
  - Stream으로부터 Data를 읽어서 객체를 만드는 것

Serialization

- 직렬화가 가능한 클래스(Serializable)
  - Serializable Interface를 구현한 클래스만이 직렬화가 가능
  - Serializable Interface는 Field나 메소드가 없는 빈 인터페이스
  - 객체를 직렬화 할 때 private Field를 포함한 모든 Field를 Byte로 변환해도 좋다는 표시 역할을 함
  - Field 선언에 static이나 transient가 붙어 있다면 직렬화가 안됨
  - 생성자와 메소드도 직렬화에 포함되지 않음

Serialization

- serialVersionUID 버전이 필요한 이유?
  - Serializable이 포함된 직렬화 객체에 serialVersionUID 값을 포함시키는 이유는 다시 역직렬화를 해서 읽어 들일 때 현재 캐스팅한 클래스에 포함되어 있는 serialVersionUID 번호와 역직렬화한 객체의 serialVersionUID 번호가 맞는지 확인하기 위해서 임
  - 이것이 맞지 않는다면 상태가 변경되었다고 판단하고 예외를 발생

OjbectInput(Output)Stream 클래스

- Serializable 인터페이스를 포함하는 클래스

- 영속성
  - 객체가 자신의 상태를 기록해 두어 다음 기회에 또는 다른 환경에서 재생될 수 있는 능력을 의미

ObjectInputStream 클래스

- File이나 Network를 통해 전달 받은 직열화된 Data를 다시 원래대로 돌리는 역할
- 직렬화하기전 객체로 다시 만들기 위해서 readObject() 이며 직렬화하기전 객체로 Casting을 해야 함

ObjectInputStream 클래스

- 생성자
  - 연결할 Byte 입출력 Stream을 생성자의 매개변수의 값으로 받아서 생성

  ✓ ObjectOutputStream(OutputStream out)
  ✓ ObjectInputStream(InputStream in)

ObjectInputStream 클래스

- 메소드

메소드	설명
int available()	객체에서 읽을 수 있는 바이트 값을 반환
void close()	객체를 닫음
void defaultReadObject()	현재 Stream에서 static,transient가 아닌 객체를 읽음
protected boolean enableResolveObject(boolean enable)	현재 Stream에서 객체를 읽는 것 허용 여부를 설정
int read()	데이터를 바이트 단위로 읽음
int read(byte[] buf, int off, int len)	buf 바이트 배열에 off 부터 len까지 읽음
boolean readBoolean()	객체의 boolean 값을 읽음
byte readByte()	객체의 1 byte를 읽는다. (8 비트)
char readChar()	객체의 1 Char를 읽는다. (16 비트)

ObjectInputStream 클래스

- 메소드

메소드	설명
protected ObjectStreamClass readClassDescriptor()	직렬화된 스트림에서 Descriptor를 읽음
double readDouble()	객체에서 1 double을 읽음 (64 비트)
ObjectInputStream.GetField. readFileds()	객체에서 영속성이 보장된 형의 이름을 가져옴
float readFloat()	객체에서 1 float을 읽음 (32 비트)
void readFully(byte[] buf)	객체에서 buf 만큼 바이트를 읽음
void readFully(byte[] buf, int off, int len)	객체에서 buf 만큼 off 부터 len만큼 읽음
int readInt()	객체에서 1 int를 읽음 (32 비트)
Long readLong()	객체에서 1 Long을 읽음 (64 비트)
Object readObject()	객체에서 Object를 읽음

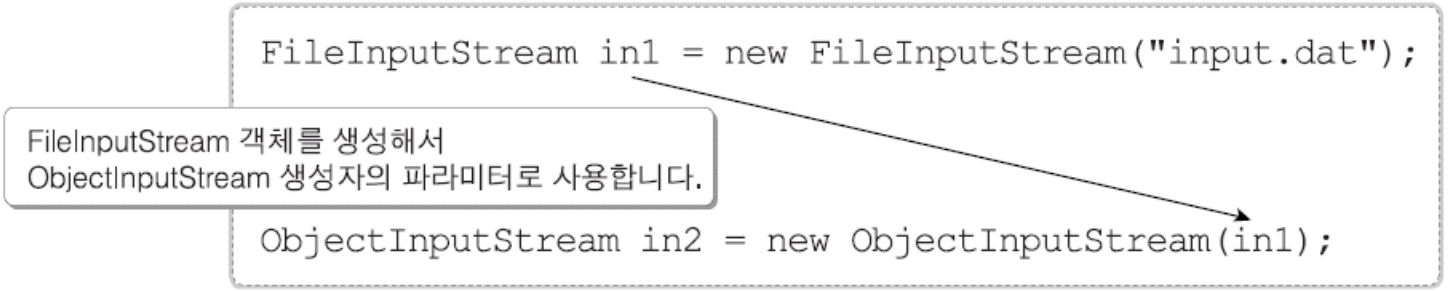
ObjectInputStream 클래스

- 메소드

메소드	설명
Short readShort()	객체에서 1 Short를 읽음 (16비트)
protected void readStreamHeader()	스트림의 헤더를 읽음
Object readUnshared()	스트림에서 "unshared" 객체를 읽음
String readUTF()	String을 UTF-8 방식으로 읽음

ObjectInputStream 클래스

- 객체를 읽는 클래스
- 사용 방법
  - 다음과 같은 방법으로 ObjectInputStream 객체를 생성



ObjectInputStream 클래스

- 사용 방법
  - 객체를 읽는 readObject() 메소드를 호출한다

```
GregorianCalendar obj = (GregorianCalendar) in2.readObject();
```

캐스트 연산이 필요

객체를 읽어서 리턴하는 메소드

  - 파일을 닫는다
    - close() 메소드 호출 방법은 다른 클래스와 동일

ObjectOutputStream 클래스

- ObjectOutput Interface를 구현한 클래스로 객체를 File에 기록 할 수 있는 클래스
- Byte 출력 Stream과 연결되어 객체를 직렬화
- ObjectOutput Interface는 writeObject(Object obj) 메소드를 포함하고 있는데 이 메소드가 객체의 Data를 직렬화 시켜주는 메소드(직렬화 메소드)
- 만약 obj가 Serializable 인터페이스로 구현되어 있지 않다면 NotSerializableException 예외가 발생
- 직렬화된 Data를 저장하는 File 확장자는 serialization의 약자인 ".ser"로 설정하는 것이 관례

ObjectOutputStream 클래스

- 메소드

메소드	설명
void close()	객체를 닫음
void defaultWriteObject()	현재 Stream에 static,transient가 아닌 객체를 씀
protected void drain()	ObjectOutputStream의 버퍼에 있는 객체를 내보냄
protected boolean enableReplaceObject(boolean enable)	현재 Stream에서 객체를 쓰는것 허용 여부를 설정
void flush()	스트림에 데이터를 내보냄
protected Object replaceObject(Object obj)	객체에 Object를 교체
void reset()	스트림을 리셋

ObjectOutputStream 클래스

- 메소드

메소드	설명
void useProtocolVersion(int version)	스트림을 내보낼 때 사용하는 프로토콜의 버전을 설정
void write(byte[] buf)	buf를 씀
void write(byte[] buf, int off, int len)	buf의 off부터 len 길이 만큼을 스트림에 씀
void write(int val)	val 바이트만큼 스트림에 씀
void writeBoolean(boolean val)	val을 스트림에 씀
void writeByte(int val)	val을 byte로 스트림에 씀 (8 비트)
void writeBytes(String str)	str을 sequence 바이트로 스트림에 씀
void writeChar(int val)	val을 Char로 스트림에 씀(16 비트)

ObjectOutputStream 클래스

- 메소드

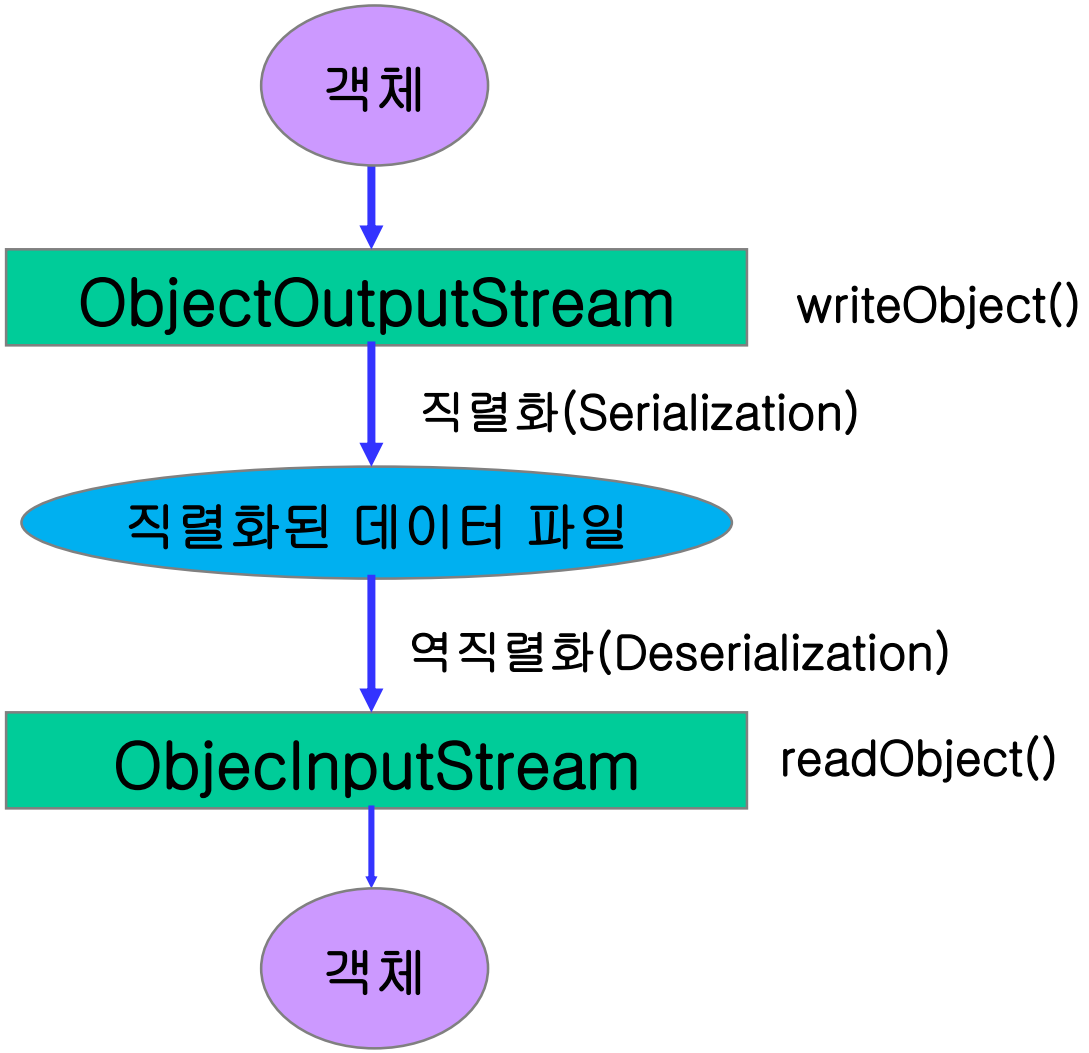
메소드	설명
void writeChars(String str)	str을 sequence char로 스트림에 씀
protected void writeClassDescriptor(ObjectStreamClass desc)	스트림에 desc를 씀
void writeDouble(double val)	val을 Double로 스트림에 씀 (64 비트)
void writeFileIds()	스트림에 버퍼에 있는 필드를 씀
void writeFloat(float val)	val을 Float으로 스트림에 씀 (32 비트)
void writeInt(int val)	val을 Int로 스트림에 씀(32 비트)
void writeLong(long val)	val을 long로 스트림에 씀(64 비트)

ObjectOutputStream 클래스

- 메소드

메소드	설명
void writeObject(Object obj)	Obj 객체를 스트림에 씀
void writeShort(int val)	val을 Short로 스트림에 씀
protected void writeStreamHeader()	스트림에 StreamHeader를 씀
void writeUnshared(Object obj)	"unshared" 객체를 스트림에 씀
void writeUTF(String str)	객체의 문자의 인코딩을 UTF-8로 설정

ObjectOutput(input)Stream 클래스



ObjectOutput(input)Stream 예제 1

- Person 클래스

```
public class Person implements Serializable {
    private static final long serialVersionUID = 12345L;
    private String name;
    private String job;

    public Person(String name, String job) {
        this.name = name;
        this.job = job;
    }

    @Override
    public String toString() {
        return "[name=" + name + ", job=" + job + "]";
    }
}
```

ObjectOutput(input)Stream 예제 1

- Main 클래스

```
public static void main(String[] args) {
    String path = ".₩₩data₩₩person.ser";

    Person lee = new Person("이순신", "대표이사");
    Person kim = new Person("김유신", "상무이사");

    try {
        OutputStream outputStream = new FileOutputStream(path);
        ObjectOutputStream output = new ObjectOutputStream(outputStream);
        output.writeObject(lee);
        output.writeObject(kim);
        output.close();
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
```

ObjectOutput(input)Stream 예제 1

```
File file = new File(path);
if (file.exists()) {
    ArrayList<Person> people =  new ArrayList<>();
    try {
        InputStream inputStream = new FileInputStream(file);
        ObjectInputStream input = new ObjectInputStream(inputStream);
        while (inputStream.available() > 0) {
            Person list = (Person) input.readObject();
            people.add(list);
        }
        input.close();
        for (int i = 0; i < people.size(); i++)
            System.out.println(people.get(i));
    } catch (IOException | ClassNotFoundException e) {
        System.err.println(e.getMessage());
    }
} else {
    System.out.println("파일이 존재하지 않아요");
}
```

ObjectOutput(input)Stream 예제 2

- Main 클래스

```
public static void main(String[] args) {
    String path = ".₩₩data₩₩person.ser";

    Person lee = new Person("이순신", "대표이사");
    Person kim = new Person("김유신", "상무이사");
    ArrayList<Person> people = new ArrayList<>();
    people.add(lee);
    people.add(kim);

    try {
        OutputStream outputStream = new FileOutputStream(path,false);
        ObjectOutputStream output = new ObjectOutputStream(outputStream);
        output.writeObject(people);
        output.flush();
        output.close();
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
```

ObjectOutput(input)Stream 예제 2

```
File file = new File(path);
if (file.exists()) {
    try {
        InputStream inputStream = new FileInputStream(file);
        ObjectInputStream input = new ObjectInputStream(inputStream);
        ArrayList<Person> list = (ArrayList<Person>) input.readObject();
        input.close();
        for (int i = 0; i < list.size(); i++)
            System.out.println(list.get(i));
    } catch (IOException | ClassNotFoundException e) {
        System.err.println(e.getMessage());
    }
} else {
    System.out.println("파일이 존재하지 않아요");
}
```

ObjectOutput(input)Stream 예제 2

- 객체가 많을 때는 ArrayList 같은 컬렉션에 저장해서 직렬화 하는 것이 편리함
  - 배열
    - 개수가 고정 일 때
  - ArrayList
    - 개수가 변할 때
    - 삽입 및 삭제가 중간에서 일어나지 않을 때
  - LinkedList
    - 개수가 변할 때
    - 검색보다는 데이터 삽입 삭제를 자주 할 때
  - Set
    - 데이터를 중복 없이 저장하는 경우

ObjectOutput(input)Stream 예제 3

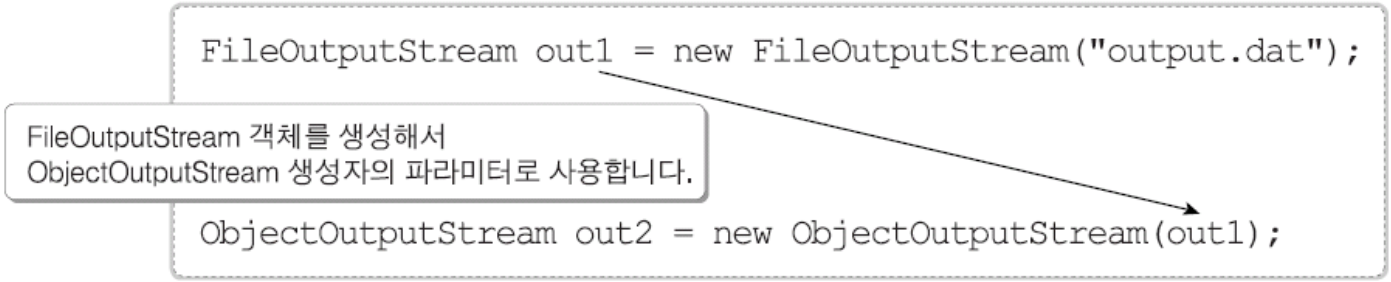
```
public class Main {
    public static void main(String[] args) {
        String file = ".//data//date.txt";
        try {
            FileOutputStream output = new FileOutputStream(file);
            ObjectOutputStream outputStream = new ObjectOutputStream(output);
            outputStream.writeObject(new GregorianCalendar(2023, 7, 14));
            outputStream.writeObject(new GregorianCalendar(2023, 7, 15));
            outputStream.writeObject(new GregorianCalendar(2023, 7, 16));
            outputStream.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
```

ObjectOutput(input)Stream 예제 3

```
        try {
            FileInputStream stream = new FileInputStream(file);
            ObjectInputStream inputStream = new ObjectInputStream(stream);
            while (true) {
                GregorianCalendar calendar = (GregorianCalendar)
                                                    inputStream.readObject()
                int year = calendar.get(Calendar.YEAR);
                int month = calendar.get(Calendar.MONTH);
                int date = calendar.get(Calendar.DATE);
                System.out.println(year + "/" + month + "/" + date);
            }
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error : " + e.getMessage());
        }
    }
}
```

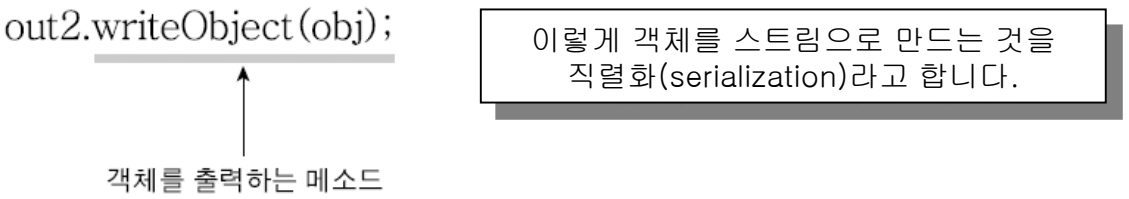
ObjectOutput(input)Stream 예제 3

- 객체를 출력하는 클래스
  - 다음과 같은 방법으로 ObjectOutputStream 객체를 생성



ObjectOutput(input)Stream 예제 3

- 객체를 출력하는 writeObject() 메소드를 호출



- 파일을 닫음
  - close() 메소드 호출 방법은 다른 클래스와 동일

ObjectOutput(input)Stream 예제 4

```
public class Box implements Serializable {
    private int width;
    private int height;
    private int depth;

    public Box(int width, int height, int depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
    }
    public int getWidth() {
        return width;
    }
    public int getHeight() {
        return height;
    }
    public int getDepth() {
        return depth;
    }
}
```

ObjectOutput(input)Stream 예제 4

```
public class Main {
    public static void main(String[] args) {
        String file = ".//data//temp.txt";
        String message = "***박스의 가로,세로,높이***";
        Box box = new Box(10, 20, 30);

        try {
            FileOutputStream outputStream = new FileOutputStream(file);
            ObjectOutputStream objectOutputStream =
                                                new ObjectOutputStream(outputStream);
            objectOutputStream.writeObject(message);
            objectOutputStream.writeObject(box);
            objectOutputStream.writeDouble(123.456);
            objectOutputStream.close();
            outputStream.close();
```

ObjectOutput(input)Stream 예제 4

```
            FileInputStream inputStream = new FileInputStream(file);
            ObjectInputStream objectInputStream =
                                        new ObjectInputStream(inputStream);
            String data = (String) objectInputStream.readObject();
            Box myBox = (Box) objectInputStream.readObject();
            System.out.println(data);
            System.out.println("박스의 가로는 : " + myBox.getWidth());
            System.out.println("박스의 세로는 : " + myBox.getDepth());
                        System.out.println("박스의 높이는 : "+myBox.getHeight());
            System.out.println("Double 값은 : " + objectInputStream.readDouble());
            objectInputStream.close();
            inputStream.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Externalizable

- JAVA에서 Externalizable과 Serializable는 객체의 Serialization와 관련된 두 가지 인터페이스
  - Serializable
    - java.io.Serializable 인터페이스를 구현하는 클래스는 자동으로 직렬화할 수 있음
    - 객체의 상태를 직렬화하기 위해 특별한 메소드를 구현할 필요가 없음
    - 하지만 이러한 자동 직렬화는 클래스의 내부 변경에 민감할 수 있음

Externalizable

- Externalizable
  - java.io.Externalizable 인터페이스를 구현하는 클래스는 직렬화를 수동으로 제어할 수 있음
  - 객체를 직렬화하고 역직렬화하는 과정을 개발자가 직접 구현해야 함
  - writeExternal()과 readExternal() 메소드를 구현하여 객체의 상태를 외부로 출력하고 다시 읽을 수 있도록 함
  - 이러한 수동 제어는 성능을 향상시킬 수 있으며, 클래스의 내부 변경에 덜 민감함
  - 따라서, Serializable은 간편하게 사용할 수 있지만 내부 동작을 제어할 수 없으며, Externalizable은 직렬화 과정을 직접 제어하고 성능을 최적화할 수 있음
  - 개발자는 클래스의 요구에 맞게 두 인터페이스 중 적절한 것을 선택할 수 있음

Externalizable 예제 1

- Main.JAVA

```
public class Main {
    public static void main(String[] args) {
        Customer customer = new Customer(2, "홍길동", 19, 180);
        try {
            System.out.println(customer);
            FileOutputStream outputStream =
                                new FileOutputStream(".//data//iotest3.dat");
            ObjectOutputStream objectOutputStream =
                                                new ObjectOutputStream(outputStream);
            objectOutputStream.writeObject(customer);
            objectOutputStream.close();
            outputStream.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Externalizable 예제 1

- Customer.JAVA

```
public class Customer implements Externalizable {
    private int ID;
    private String name;
    private int age;
    private double height;

    public Customer(int ID, String name, int age, double height) {
        this.ID = ID;
        this.name = name;
        this.age = age;
        this.height = height;
    }

    public void writeExternal(ObjectOutput oos) throws IOException {
        oos.writeObject(new Integer(ID));
        oos.writeObject(name);
        oos.writeObject(new Integer(age));
        oos.writeObject(new Double(height));
    }
```

Externalizable 예제 1

- Customer.JAVA

```
    public void readExternal(ObjectInput ois)
                                        throws ClassNotFoundException, IOException {
        ID = (Integer) ois.readObject();
        name = (String) ois.readObject();
        age = (Integer) ois.readObject();
        height = (Double) ois.readObject();
    }

    public String toString() {
        String temp;
        temp = ID + "₩t " + name + "₩t " + age + "₩t " + height;
        return temp;
    }
}
```

Externalizable 예제 2

```
public class Main {
    public static void main(String[] args) {
        try {
            FileInputStream inputStream = new FileInputStream(".//data//iotest3.dat");
            ObjectInputStream objectInputStream =
                                                new ObjectInputStream(inputStream);
            Customer customer = (Customer) objectInputStream.readObject();
            System.out.println("ID ₩t name ₩t age ₩t height");
            System.out.println(customer);
            objectInputStream.close();
            inputStream.close();
        } catch (ClassNotFoundException | IOException ce) {
            System.out.println(ce.getMessage());
        }
    }
}
```