Serialization and File I/O: Saving Objects

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

Capture the Beat

사용자 데이터가 그 데이터를 생성한 Java 프로그램에서만 사용된다면:

직렬화(serialization)를 사용하라.

"평평한(직렬화된) 객체를 저장하는 파일로 작성하라. 따라서, 프로그램이 그 파일에서 직렬화된 객체를 읽어 들여서, 이를 살아있는 힙에서 서식하는 객체로 다시 부풀릴 수 있다."

데이터가 다른 프로그램에서도 사용된다면:

일반 **텍스트 파일**(plain text file)로 저장하라.

"다른 프로그램이 구문 분석할 수 있는 구분 기호가 있는 파일로 작성하라."

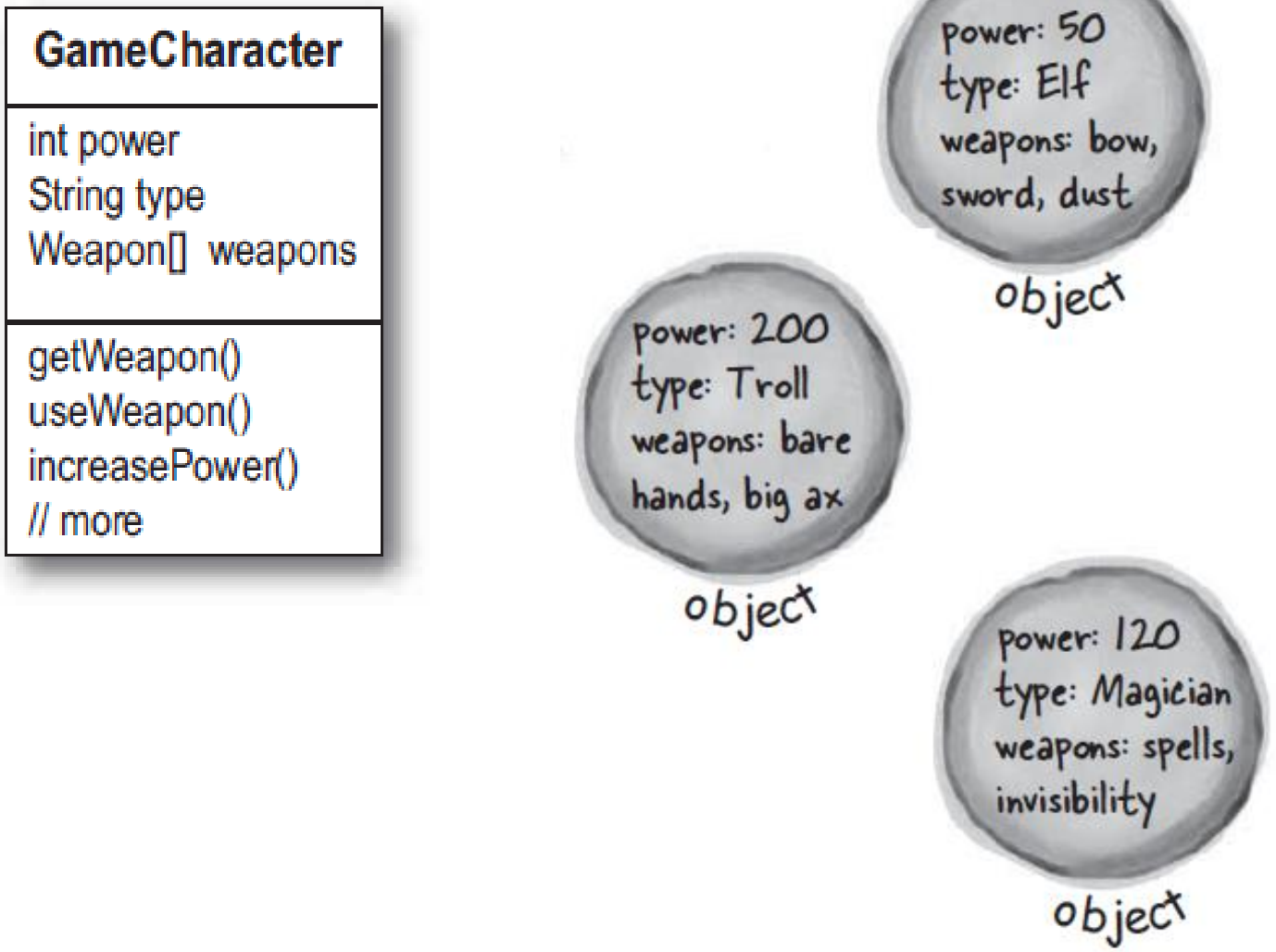
예를 들어, 스프레드 시트나 DB 애플리케이션에서 사용할 수 있는 탭 구분 파일 등으로 만들어라.

Saving State

두 개 이상의 세션을 사용하는 게임 프로그램이 있다고 가정하자.

- ✓ 게임이 진행되면서 게임 캐릭터는 '더 강해지거나, 약해지거나, 똑똑해지거나 ' 하는 식으로 변하기도 하고 '무기를 구하거나, 잃거나' 할 수도 있다.
- ✓ 그런데 게임을 시작할 때마다 처음부터 새로 시작하고 싶은 사용자는 없을 것이다.
- ✓ 따라서 캐릭터의 상태를 저장하는 방법, 게임을 다시 시작할 때 예전 상태를 불러오는 방법이 필요하다.
- ✓ **가능한 한 쉽게 저장하고 다시 불러오는 방법**이 필요하다.

3개의 게임 캐릭터를 저장해야 한다고 하자...



Option one

3개의 직렬화된 캐릭터 객체를 파일에 저장한다.

- ✓ 텍스트 파일처럼 읽으려고 하면 아래처럼 무의미한 내용이 보일 것이다:

```
¨ÌsrGameCharacter
¨%gê8MÛIpowerLjava/lang/
String;[weaponst[Ljava/lang/
String;xp2tlfur[Ljava.lang.String;≠“VÁ
È{Gxptbowtswordtdustsq˜»tTrolluq˜tb
are handstbig axsq˜xtMagicianuq˜tspe
llstinvisibility
```

Option two

일반 텍스트 파일로 저장한다.

- ✓ 파일을 생성하고 세 줄의 텍스트를, 캐릭터당 한 줄씩, 콤마로 구분하여 저장한다:

```
50, Elf, bow, sword, dust
200, Troll, bare hands, big ax
120, Magician, spells, invisibility
```

직렬화된 파일은 사람이 읽기가 훨씬 어렵다.
하지만 프로그램에서는 3개의 직렬화된 객체로부터 복구하는 것이 텍스트
파일로 저장한 것보다 훨씬 더 쉽다

직렬화된 객체를 파일에 저장하는 방법

객체를 직렬화하는 방법은 다음과 같다:

Make a **FileOutputStream**

```
FileOutputStream fileStream = new FileOutputStream("MyGame.ser");
```

"MyGame.ser"가 존재하지 않으면 자동으로 생성될 것이다.

FileOutputStream 객체를 만든다. FileOutputStream은 파일에 연결(생성)하는 방법을 알고 있다.

Make an **ObjectOutputStream**

```
ObjectOutputStream os = new ObjectOutputStream(fileStream);
```

ObjectOutputStream은 객체를 생성하도록 해주지만 파일에 직접 연결할 수는 없다. 헬퍼에 넣어질 필요가 있다. 이를 스트림간의 'chaining'이라 부른다.

3. Write the object

```
os.writeObject(characterOne);
os.writeObject(characterTwo);
os.writeObject(characterThree);
```

characterOne, characterTwo, characterThree에 의해 참조되는 객체들을 직렬화하여 그들을 "MyGame.ser" 파일에 저장한다(write).

4. Close the ObjectOutputStream

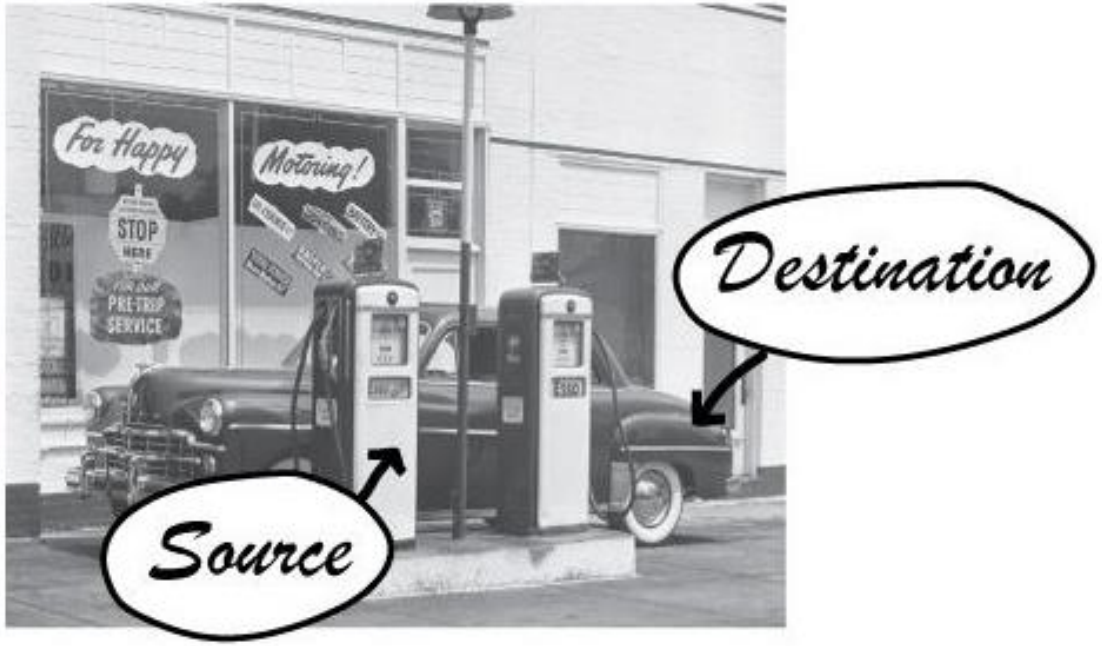
```
os.close();
```

최상위 스트림을 클로징하면 하위의 스트림들도 클로징된다. 따라서 FileOutputStream도 자동으로 클로징된다.

데이터는 스트림으로 이동한다

연결 스트림(connection stream)은 소스 또는 목적지(파일, 소켓 등)에 대한 연결을 표현한다.

반면에 **체인 스트림**(chain stream)은 혼자서는 목적지에 연결될 수 없고 반드시 연결 스트림에만 연결될 수 있다.



데이터는 스트림으로 이동한다

Java I/O API는 연결 스트림과 체인 스트림을 지원한다:

연결 스트림: 파일 또는 네트워크 소켓과 같은 목적지/소스에 대한 연결을 표현
체인 스트림: 다른 스트림에 연결될 때만 동작

적어도 2개의 스트림이 연결되어야 유용한 뭔가를 수행할 수 있다.

하나는 연결을 표현하고, 다른 하나는 메소드를 호출한다.

Why two?

왜냐하면 연결 스트림이 너무 저수준이기 때문에.

예를 들어, FileOutputStream은 바이트를 저장하기 위한 메소드를 가지고 있다.
그러나 우리는 바이트 단위를 취급하고 싶지 않다. 객체 단위를 취급하고 싶다.
따라서 **더 고수준의 체인 스트림**이 필요하다.

OK, 그렇다면 정확하게 필요한 기능을 수행하는 하나의 스트림을 만들어서 쓰면 되지 않나?

- 객체를 저장하는 기능과 그 밑에서 객체를 스트림으로 변환해주는 기능을 모두 갖춘 단 하나의 스트림

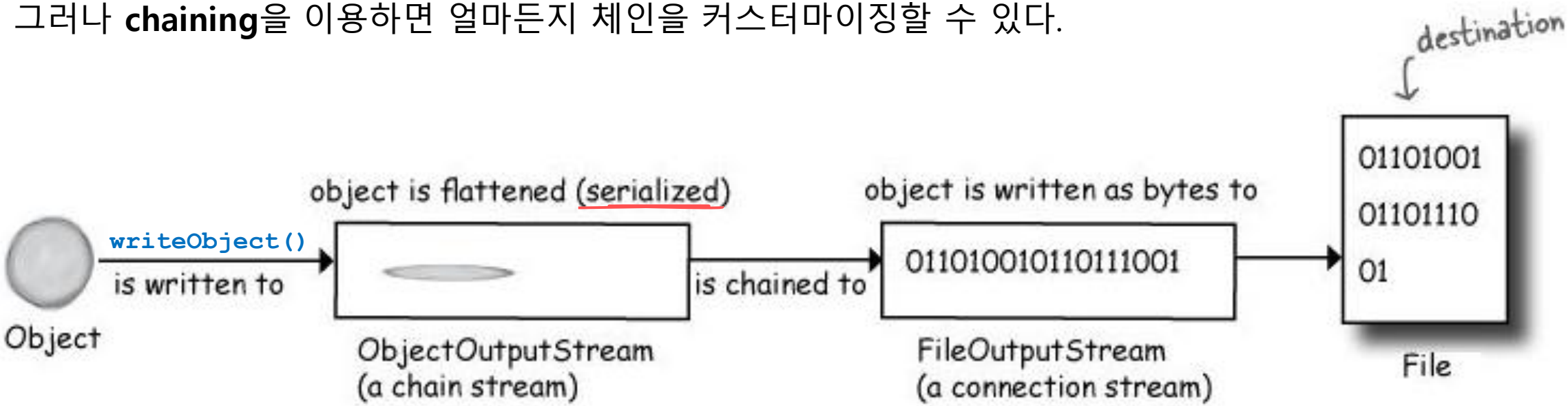
객체지향적으로 생각해보자.

- 각 클래스는 한 가지 일을 잘한다.
- FileOutputStream은 바이트를 파일로 저장한다.
- ObjectOutputStream은 객체를 스트림으로 저장될 수 있는 데이터로 변환한다.
- 따라서 연결 스트림인 FileOutputStream은 파일을 저장하게 만들고, 그 끝에다가 체인 스트림인 ObjectOutputStream을 연결하면 된다.
- 이제 ObjectOutputStream의 writeObject()를 호출하면 **객체가 스트림으로 흘러 들어가** 궁극적으로 바이트를 파일로 저장해주는 FileOutputStream으로 이동하게 된다.

연결 스트림과 체인 스트림을 다양하게 혼합하여 사용하는 능력은 엄청난 유연성을 제공해 줄 것이다!!

오로지 하나의 스트림 클래스만 써야 한다면 API 설계자가 우리가 원할지도 모르는 모든 것을 미리 알아서 설계에 반영했기를 바라는 수밖에 없다.

그러나 **chaining**을 이용하면 얼마든지 체인을 커스터마이징할 수 있다.



객체가 직렬화되면 어떻게 될까?

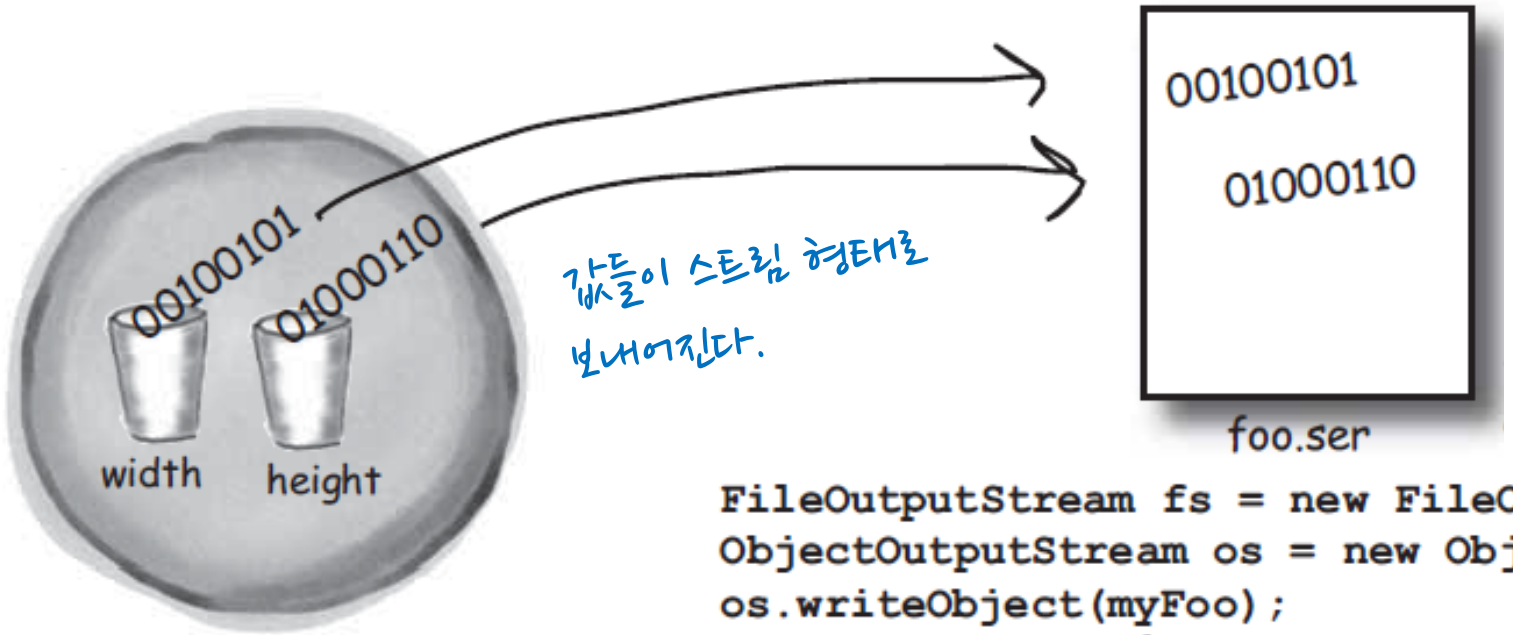
1. 힙에 있는 객체

힙에 있는 객체는 **상태**(객체 인스턴스 변수의 값)를 갖는다.
이들 값은 같은 클래스의 인스턴스마다 다르다.

2. 직렬화된 객체

직렬화된 객체도 인스턴스 변수 값을 갖는다.

그래야만 동일한 인스턴스[객체]가 다시 힙으로 되돌아 올 수 있다.



값들이 스트림 형태로 보내어진다.

폭과 높이에 대한 인스턴스 변수 값들이 파일 **foo.ser**에 저장된다. 여기에는 JVM이 객체를 복구할 때 필요로 하는 약간의 정보도 포함된다.

```
FileOutputStream fs = new FileOutputStream("foo.ser");
ObjectOutputStream os = new ObjectOutputStream(fs);
os.writeObject(myFoo);
```

```
Foo myFoo = new Foo();
myFoo.setWidth(37);
myFoo.setHeight(70);
```

파일 **foo.ser**에 연결하는 FileOutputStream을 생성한 다음, ObjectOutputStream에 체인으로 연결한 후 객체를 쓰도록 ObjectOutputStream에 말해준다.

객체의 상태란? 무엇이 저장되어야 하나?

원시 값인 37, 70을 저장하는 일은 너무 쉽다.

- 그러나 객체가 다른 객체의 레퍼런스인 인스턴스 변수를 가지고 있다면 어떨까?
- 객체 레퍼런스인 인스턴스 변수를 5개나 가지고 있는 객체의 경우는?
- 또 객체 인스턴스 변수 자체가 인스턴스 변수를 가지고 있는 경우는?

생각해보자!

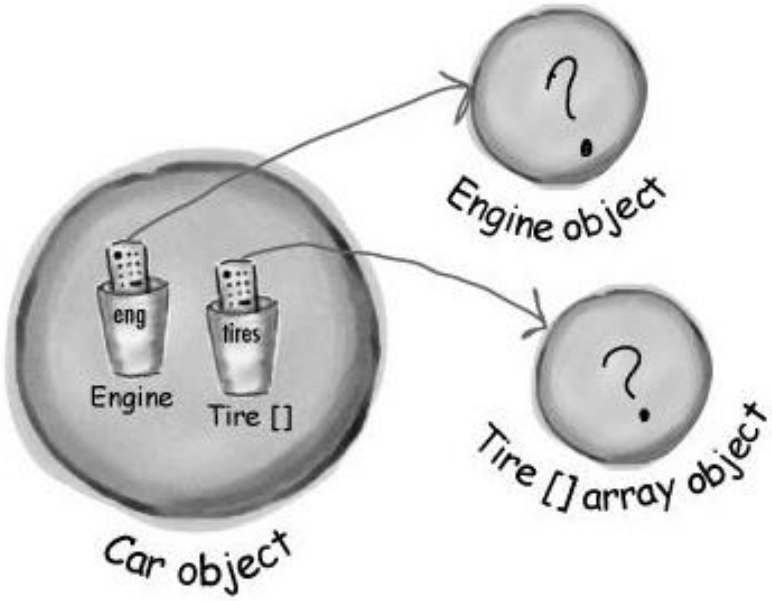
- 객체의 어떤 부분이 잠재적으로 유일한가?
- 저장되었던 것과 동일한 객체를 얻기 위해서 어떤 것이 복구되어야 할까?
- 분명히 메모리 위치는 달라졌을 것이다. (그것은 우리가 제어할 수 없다)
- 우리가 할 수 있는 일은 힙의 어딘가로부터 객체가 저장될 때와 **동일한 상태**를 가지는 객체를 가져오는 것이다.

Brain Barbell

Car 객체를 원래의 상태로 복구될 수 있게 하기 위해서는 어떻게 저장되어야 할까?

Car 객체는 2개의 다른 객체를 참조하는
2개의 인스턴스 변수를 가지고 있다.

Car 객체에서 어떤 것들을 저장해야 할까?



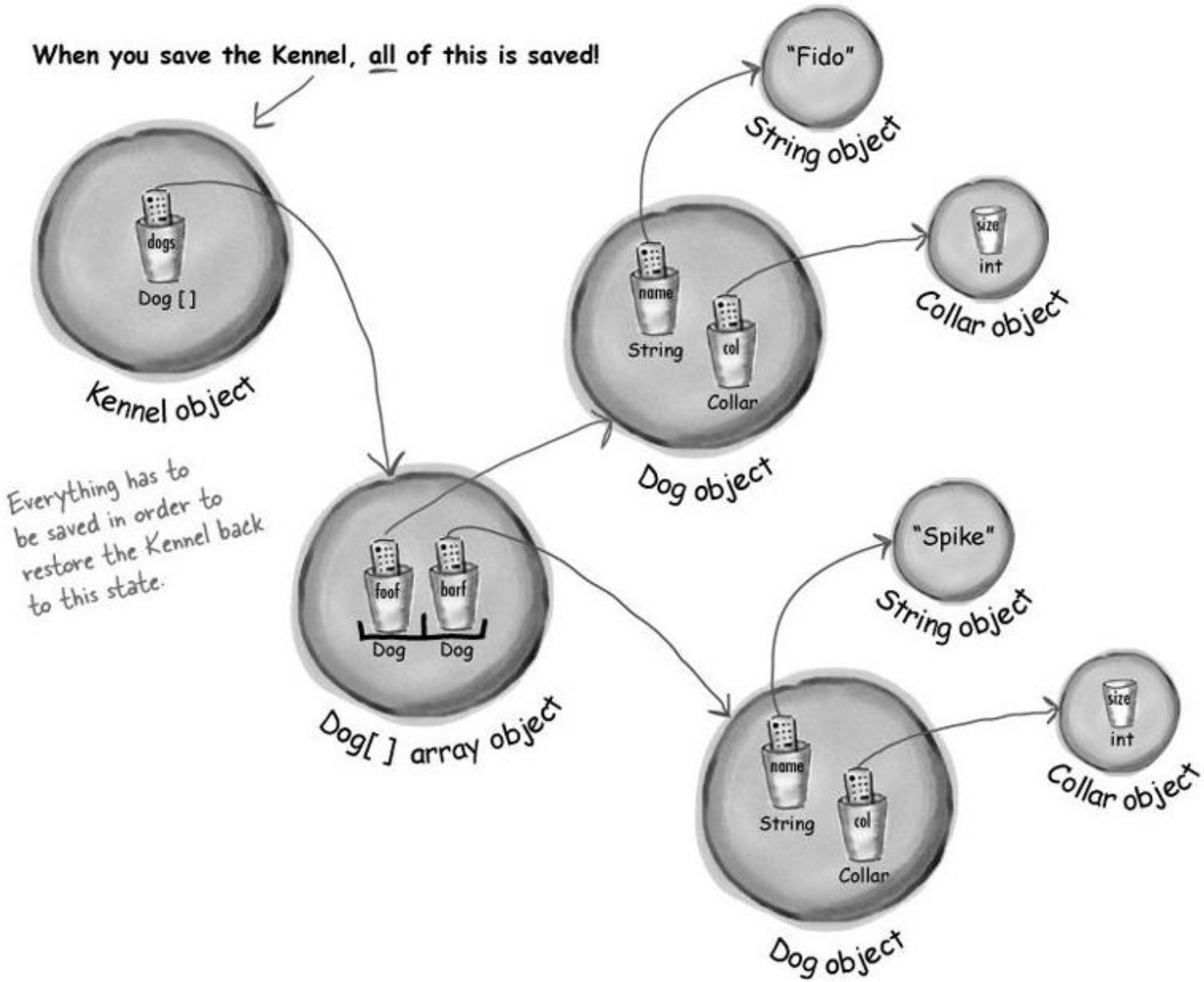
객체가 직렬화될 때 인스턴스 변수가 참조하는 모든 객체들도 함께 직렬화된다.

그렇게 직렬화된 객체들이 참조하는 모든 객체들도 직렬화되고...

그리고 그렇게 직렬화된 객체들이 참조하는 모든 객체 또한 직렬화되고...

그런데 가장 좋은 점은 이러한 모든 일이 **자동으로 이루어진다**는 것이다!!

객체의 상태란 정확히 무엇일까? 무엇이 저장되어야 하나?



17

클래스를 직렬화될 수 있도록 하려면 Serializable을 구현하라

```
objectOutputStream.writeObject(myBox);
```

여기에 오는 것이 무엇이던 간에 반드시 Serializable을 구현해야 한다.

```
import java.io.*;   <- Serializable is in the java.io package, so you need the import
public class Box implements Serializable {

    private int width;
    private int height;   <- these two values will be saved

    public void setWidth(int w) {
       width = w;
    }

    public void setHeight(int h) {
       height = h;
    }

public static void main (String[] args) {

      Box myBox = new Box();
      myBox.setWidth(50);
      myBox.setHeight(20);

      try {   <- I/O operations can throw exceptions.
         FileOutputStream fs = new FileOutputStream("foo.ser");
         ObjectOutputStream os = new ObjectOutputStream(fs);
         os.writeObject(myBox);
         os.close();
      } catch(Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

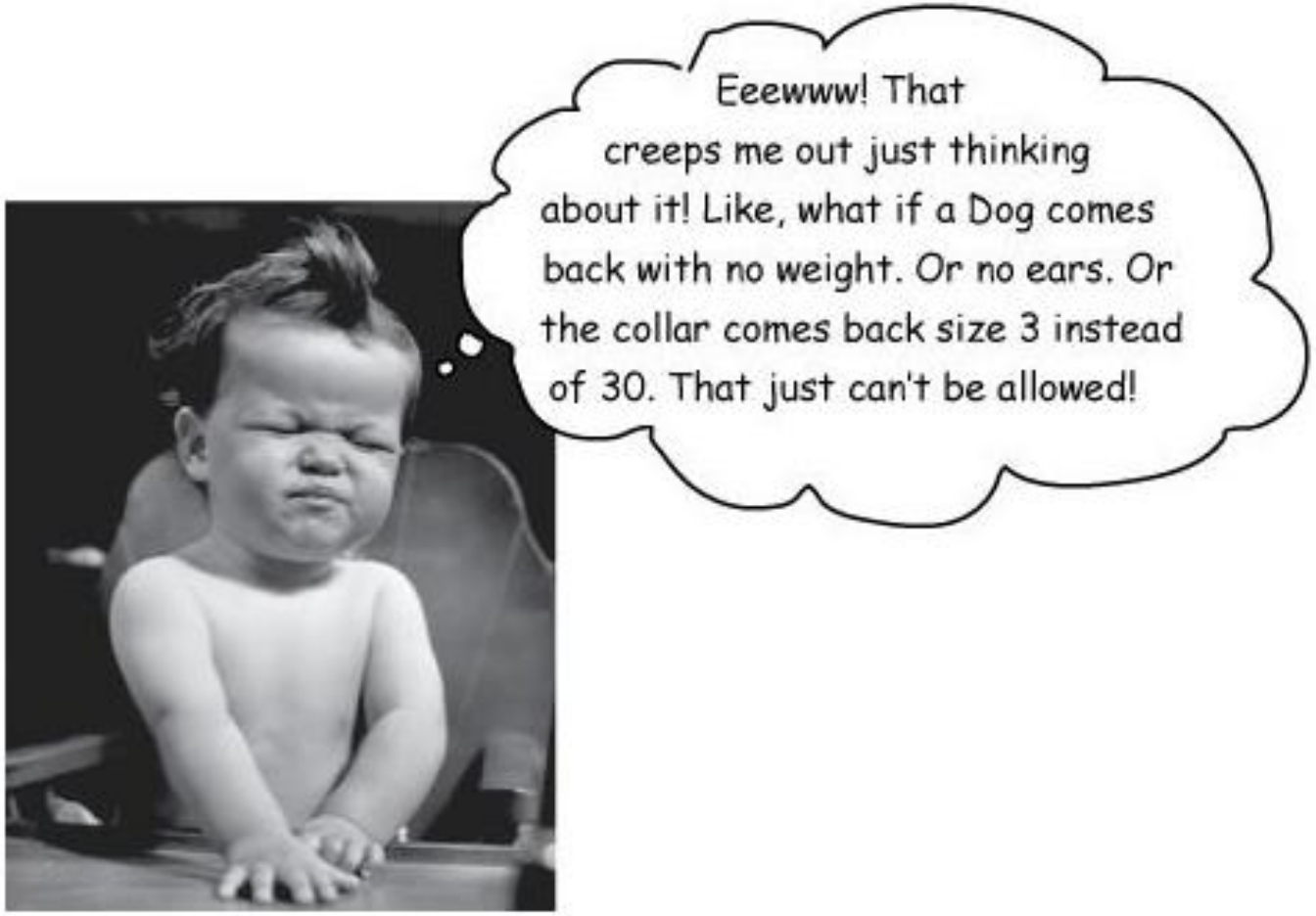
정작 구현해야 할 메소드는 한 개도 없다!!

foo.ser라는 파일이 존재하면 연결한다. 존재하지 않으면 새로 생성한다.

Make an ObjectOutputStream chained to the connection stream. Tell it to write the object.

Serialization is all or nothing.

객체의 상태 중에 일부가 제대로 저장되지 않았다면 어떻게 될까?



이 경우 직렬화는 완료되지 않는다!

```
import java.io.*;

public class Pond implements Serializable {

    private Duck duck = new Duck();

    public static void main (String[] args) {
       Pond myPond = new Pond();
       try {
         FileOutputStream fs = new FileOutputStream("Pond.ser");
         ObjectOutputStream os = new ObjectOutputStream(fs);

         os.writeObject(myPond);
         os.close();

       } catch(Exception ex) {
           ex.printStackTrace();
       }
    }
}
```

Pond 객체는 직렬화될 수 있다.

Class Pond has one instance variable, a Duck.

myPond(Pond 객체)를 직렬화할 때 Duck 인스턴스 변수도 자동으로 직렬화된다.

```
public class Duck {
   // duck code here
}
```

잉!! Duck이 Serializable되지 않는다!
즉, Serializable을 구현하지 않고 있다.
따라서 Pond 객체를 직렬화하려고 하면 Pond의 Duck 인스턴스 변수가 저장될 수 없기 때문에 실패한다.

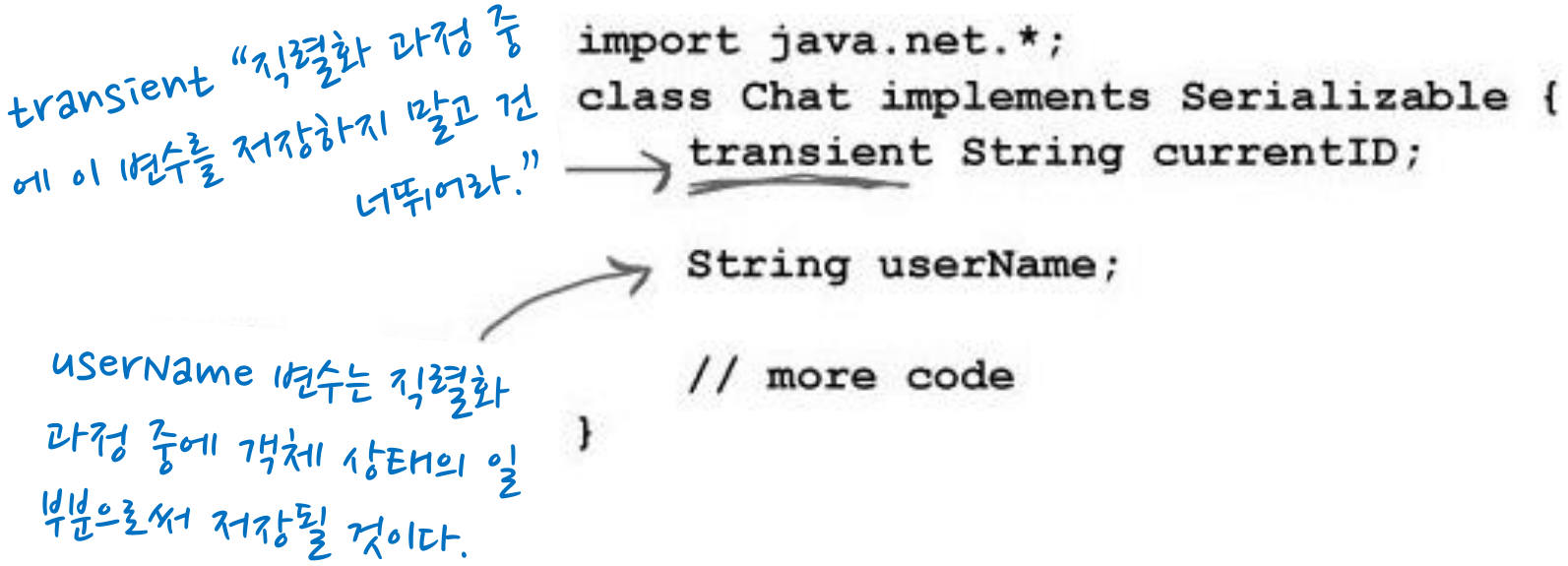
When you try to run the main in class Pond:

```
File Edit Window Help Regret
% java Pond
java.io.NotSerializableException: Duck
        at Pond.main(Pond.java:13)
```


It's hopeless, then? I'm completely screwed if the idiot who wrote the class for my instance variable forgot to make it Serializable?

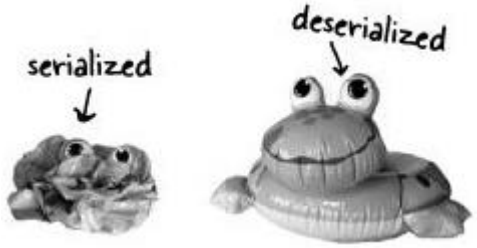
어떤 인스턴스 변수를 저장할 수 없다면 transient로 지정하라.

직렬화 과정에서 어떤 인스턴스 변수를 건너뛰고 싶다면 해당 변수에 **transient** 키워드를 써서 표시하면 된다.



Deserialization: 객체 복구

객체 직렬화에서 가장 중요한 점은 객체를 **원래 상태 그대로** 다시 복구시킬 수 있어야 한다는 것이다.



1. Make a FileInputStream

```
FileInputStream fileStream = new FileInputStream("MyGame.ser");
```

If the file "MyGame.ser" doesn't exist, you'll get an exception.

FileInputStream 객체를 만든다.
FileInputStream은 기존 파일에 연결하는 방법을 알고 있다.

2. Make an ObjectInputStream

```
ObjectInputStream os = new ObjectInputStream(fileStream);
```

ObjectInputStream은 객체를 읽을 수 있게 해주지만, 파일에 직접 연결시켜 주지는 못한다. 연결 스트림(FileInputStream)에 체인으로 연결되어야 한다.

3. Read the objects

```
Object one = os.readObject();
Object two = os.readObject();
Object three = os.readObject();
```

4. Cast the objects

```
GameCharacter elf = (GameCharacter) one;
GameCharacter troll = (GameCharacter) two;
GameCharacter magician = (GameCharacter) three;
```

readObject()의 반환 값은 Object 타입이다. (ArrayList에서 처럼) 그래서 다시 원하는 타입으로 캐스팅 할 필요가 있다.

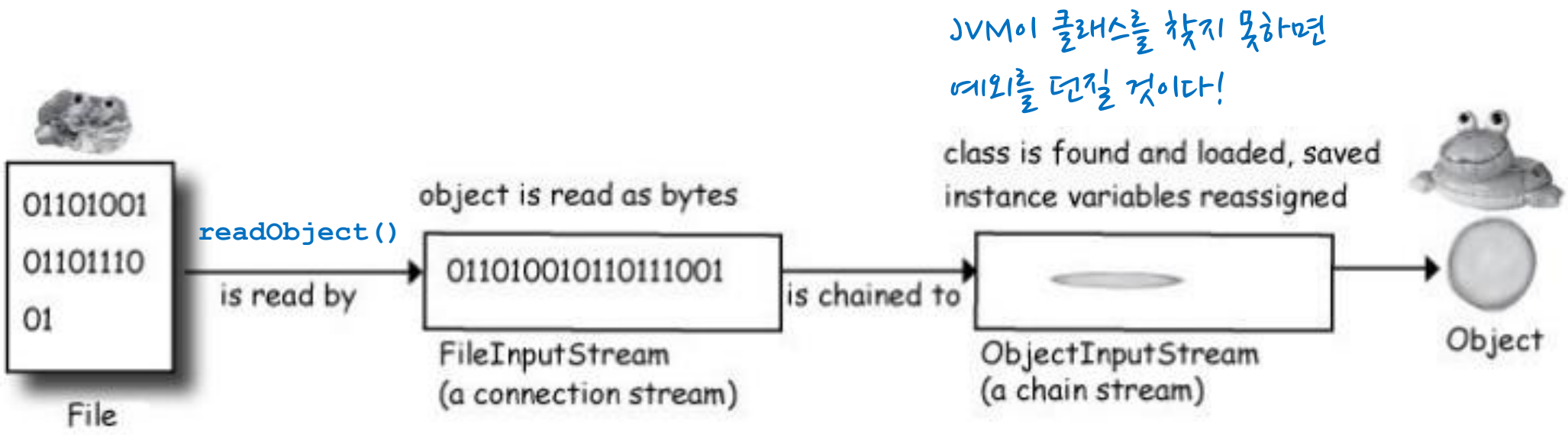
5. Close the ObjectInputStream

```
os.close();
```

최상위 스트림을 클로징하면 하위의 스트림들도 클로징된다. 따라서 FileOutputStream도 자동으로 클로징된다.

What happens during deserialization?

객체가 역직렬화될 때 JVM은 객체가 직렬화될 때와 동일한 상태를 가지는 새로운 객체를 힙에 만들려고 시도한다.



실습과제 14-1 게임 캐릭터 저장 및 복구

```
import java.io.*;

public class GameSaverTest {
   public static void main(String[] args) {
     GameCharacter one = new GameCharacter(50, "Elf", new String[] {"bow", "sword", "dust"});
     GameCharacter two = new GameCharacter(200, "Troll", new String[] {"bare hands", "big ax"});
     GameCharacter three = new GameCharacter(120, "Magician", new String[] {"spells", "invisibility"});

     // imagine code that does things with the characters that might change their state values

     try {
        ObjectOutputStream os = new ObjectOutputStream(new FileOutputStream("Game.ser"));
        os.writeObject(one);
        os.writeObject(two);
        os.writeObject(three);
        os.close();
     } catch(IOException ex) {
        ex.printStackTrace();
     }
     one = null;
     two = null;
     three = null;

     try {
       ObjectInputStream is = new ObjectInputStream(new FileInputStream("Game.ser"));
        GameCharacter oneRestore = (GameCharacter) is.readObject();
        GameCharacter twoRestore = (GameCharacter) is.readObject();
        GameCharacter threeRestore = (GameCharacter) is.readObject();

       System.out.println("One's type: " + oneRestore.getType());
       System.out.println("Two's type: " + twoRestore.getType());
       System.out.println("Three's type: " + threeRestore.getType());
     } catch(Exception ex) {
        ex.printStackTrace();
     }
   }
```

Make some characters...

We set them to null so we can't access the objects on the heap.

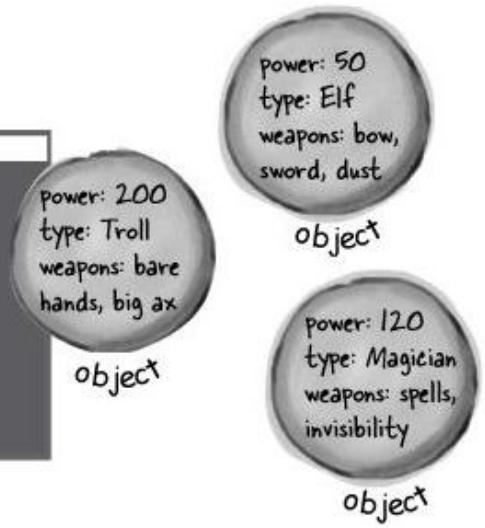
Now read them back in from the file...

Check to see if it worked.

The GameCharacter class

```
import java.io.*;

public class GameCharacter implements Serializable {
   int power;
   String type;
   String[] weapons;

   public GameCharacter(int p, String t, String[] w) {
       power = p;
       type = t;
       weapons = w;
   }

   public int getPower() {
     return power;
   }

   public String getType() {
      return type;
   }

   public String getWeapons() {
       String weaponList = "";

       for (int i = 0; i < weapons.length; i++) {
          weaponList += weapons[i] + " ";
       }
       return weaponList;
   }
```

```
File Edit Window Help Resuscitate
% java GameSaverTest
One's type: Elf
Two's type: Troll
Three's type: Magician
```



Writing a String to a Text File

What the game character data might look like if you wrote it out as a human-readable text file.

```
50,Elf,bow,sword,dust
200,Troll,bare hands,big ax
120,Magician,spells,invisibility
```

To write a serialized object:

```
objectOutputStream.writeObject(someObject);
```

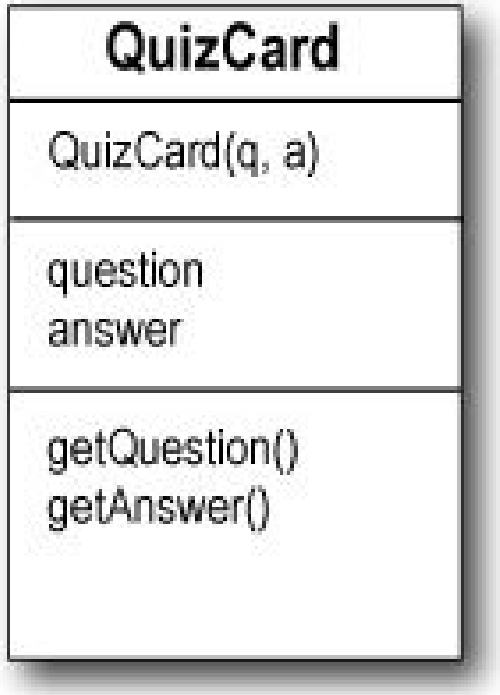
To write a String:

```
fileWriter.write("My first String to save");
```

```
import java.io.*;                          // We need the java.io package for FileWriter

class WriteAFile {
    public static void main (String[] args) {

        try {
            FileWriter writer = new FileWriter("Foo.txt");   // If the file "Foo.txt" does not exist, FileWriter will create it

            writer.write("hello foo!");    // The write() method takes a String

            writer.close();                // Close it when you're done!

        } catch(IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

모든 I/O 관련 작업은 try/catch 문 안에 있어야 한다. 모든 것이 IOException을 던질 수 있다!!

Text File Example: e-Flashcards



세 개의 클래스가 필요하다:

1) **QuizCardBuilder**, e-플래시카드 집합을 생성하고 저장하기 위한 저작 도구

2) **QuizCardPlayer**, 플래시카드를 로드하고 사용자가 플레이할 수 있게 해주는 엔진

3) **QuizCard**, 카드 데이터를 표현하는 클래스

Text File Example: e-Flashcards

QuizCardBuilder

Has a File menu with a "Save" option for saving the current set of cards to a text file.

QuizCardPlayer

Has a File menu with a "Load" option for loading a set of cards from a text file.

Quiz Card Builder (code outline)

```
public class QuizCardBuilder {

  public void go() {
    // build and display gui
  }

  private class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // add the current card to the list and clear the text areas
    }
  }

  private class SaveMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // bring up a file dialog box
      // let the user name and save the set
    }
  }

  private class NewMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // clear out the card list, and clear out the text areas
    }
  }

  private void saveFile(File file) {
    // iterate through the list of cards, and write each one out to a text file
    // in a parseable way (in other words, with clear separations between parts)
  }
}
```

Builds and displays the GUI, including making and registering event listeners.

Inner class

Inner class

Inner class

Called by the SaveMenuListener; does the actual file writing.

사용자가 'Next card' 버튼을 클릭하면 트리거된다; 사용자가 현재 카드를 리스트에 저장하고 새로운 카드를 시작하기 원한다는 것을 의미한다.

사용자가 File 메뉴로부터 'Save' 버튼을 클릭하면 트리거된다; 사용자가 현재 리스트에 있는 모든 카드를 'Set'로써 저장하고 싶다는 의미이다. (Hollywood Trivia, Java Rules 처럼)

사용자가 File 메뉴로부터 'New' 버튼을 클릭하면 트리거된다; 의미는 사용자가 새로운 Set로 시작하고 싶다는 것이다.

Quiz Card Builder

```
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;

public class QuizCardBuilder {

    private JTextArea question;
    private JTextArea answer;
    private ArrayList<QuizCard> cardList;
    private JFrame frame;


    public static void main (String[] args) {
       QuizCardBuilder builder = new QuizCardBuilder();
       builder.go();
    }


    public void go() {
        // build gui

        frame = new JFrame("Quiz Card Builder");
        JPanel mainPanel = new JPanel();
        Font bigFont = new Font("sanserif", Font.BOLD, 24);
        question = new JTextArea(6,20);
        question.setLineWrap(true);
        question.setWrapStyleWord(true);
        question.setFont(bigFont);

        JScrollPane qScroller = new JScrollPane(question);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

        answer = new JTextArea(6,20);
        answer.setLineWrap(true);
        answer.setWrapStyleWord(true);
        answer.setFont(bigFont);

        JScrollPane aScroller = new JScrollPane(answer);
        aScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        aScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);


        JButton nextButton = new JButton("Next Card");

        cardList = new ArrayList<QuizCard>();

        JLabel qLabel = new JLabel("Question:");
        JLabel aLabel = new JLabel("Answer:");

        mainPanel.add(qLabel);
        mainPanel.add(qScroller);
        mainPanel.add(aLabel);
        mainPanel.add(aScroller);
        mainPanel.add(nextButton);
        nextButton.addActionListener(new NextCardListener());
        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        JMenuItem newMenuItem = new JMenuItem("New");
```

This is all GUI code here. Nothing special, although you might want to look at the MenuBar, Menu, and MenuItems code.

```
        JMenuItem saveMenuItem = new JMenuItem("Save");
        newMenuItem.addActionListener(new NewMenuListener());

        saveMenuItem.addActionListener(new SaveMenuListener());
        fileMenu.add(newMenuItem);
        fileMenu.add(saveMenuItem);
        menuBar.add(fileMenu);
        frame.setJMenuBar(menuBar);
        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(500,600);
        frame.setVisible(true);
    }

    public class NextCardListener implements ActionListener {
       public void actionPerformed(ActionEvent ev) {

          QuizCard card = new QuizCard(question.getText(), answer.getText());
          cardList.add(card);
          clearCard();
       }
    }

    public class SaveMenuListener implements ActionListener {
       public void actionPerformed(ActionEvent ev) {
          QuizCard card = new QuizCard(question.getText(), answer.getText());
          cardList.add(card);

          JFileChooser fileSave = new JFileChooser();
          fileSave.showSaveDialog(frame);
          saveFile(fileSave.getSelectedFile());
       }
    }

    public class NewMenuListener implements ActionListener {
       public void actionPerformed(ActionEvent ev) {
          cardList.clear();
          clearCard();
       }
    }

    private void clearCard() {
       question.setText("");
       answer.setText("");
       question.requestFocus();
    }

    private void saveFile(File file) {
      try {
          BufferedWriter writer = new BufferedWriter(new FileWriter(file));

          for(QuizCard card:cardList) {
             writer.write(card.getQuestion() + "/");
             writer.write(card.getAnswer() + "\n");
          }
         writer.close();

       } catch(IOException ex) {
          System.out.println("couldn't write the cardList out");
          ex.printStackTrace();
       }
    }
}
```

메뉴 바를 만들고, File 메뉴를 만든다. 다음은 'new'와 'Save'메뉴 항목들을 File 메뉴에 넣는다. 메뉴를 메뉴 바에 넣은 다음 frame에게 이 메뉴 바를 사용하도록 말해준다. 메뉴 항목들은 ActionEvent를 점화시킬 수 있다.

File 다이얼로그 박스를 가져와서 사용자가 'Save'를 선택할 때까지 이 라인에서 기다린다. 모든 파일 다이얼로그 네비게이션과 파일 선택 등은 JFilechooser에 의해 수행된다.

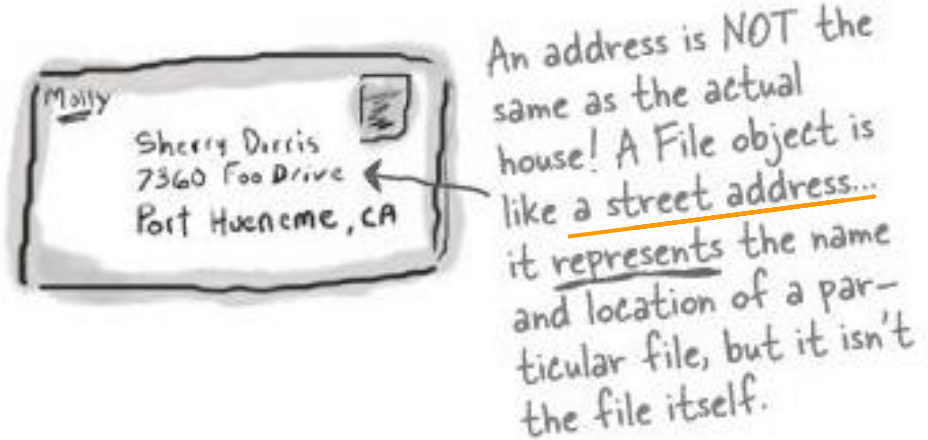
The method that does the actual file writing (called by the SaveMenuListener's event handler). The argument is the 'File' object the user is saving. We'll look at the File class on the next page.

Writing을 효율적으로 하기 위해 FileWriter를 BufferedWriter에 체인으로 연결한다.

Walk through the ArrayList of cards and write them out, one card per line, with the question and answer separated by a "/", and then add a newline character ("\n")

java.io.File 클래스

java.io.File 클래스는 디스크의 파일을 표현해 주지만 실제로 파일의 내용을 표현해 주지는 않는다.



File 객체로 할 수 있는 것들:

Make a File object representing an existing file

```
File f = new File("MyCode.txt");
```

Make a new directory

```
File dir = new File("Chapter7");
dir.mkdir();
```

List the contents of a directory

```
if (dir.isDirectory()) {
        String[] dirContents = dir.list();
        for (int i = 0; i < dirContents.length; i++) {
                System.out.println(dirContents[i]);
        }
}
```

Get the absolute path of a file or directory

```
System.out.println(dir.getAbsolutePath());
```

Delete a file or directory (returns true if successful)

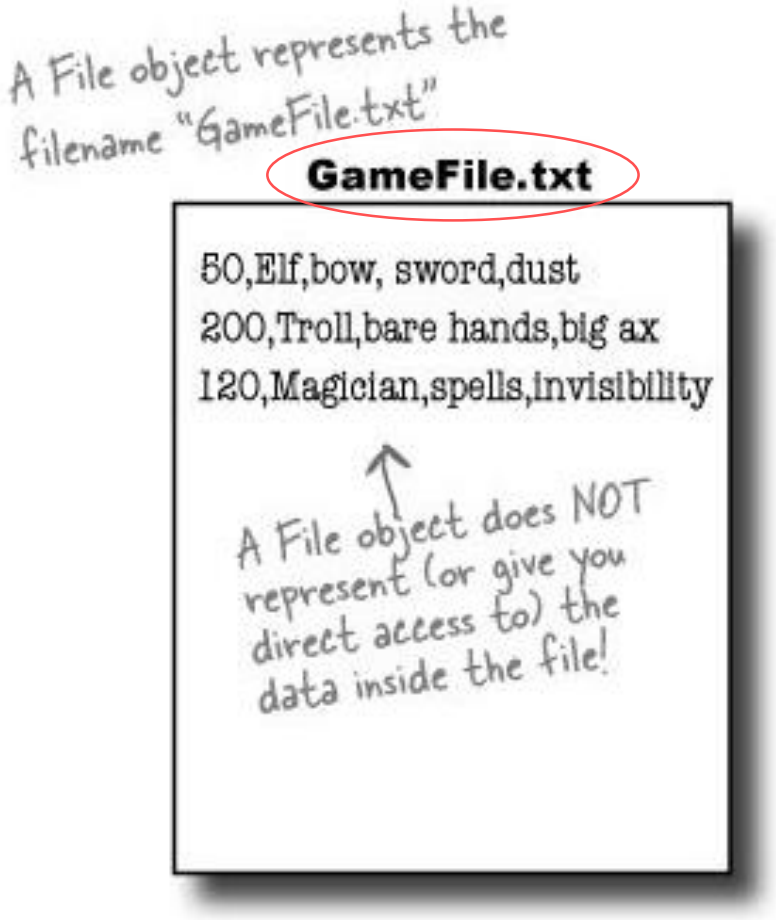
```
boolean isDeleted = f.delete();
```

NOTE

A File object represents the name and path of a file or directory on disk, for example:

/Users/Kathy/Data/GameFile.txt

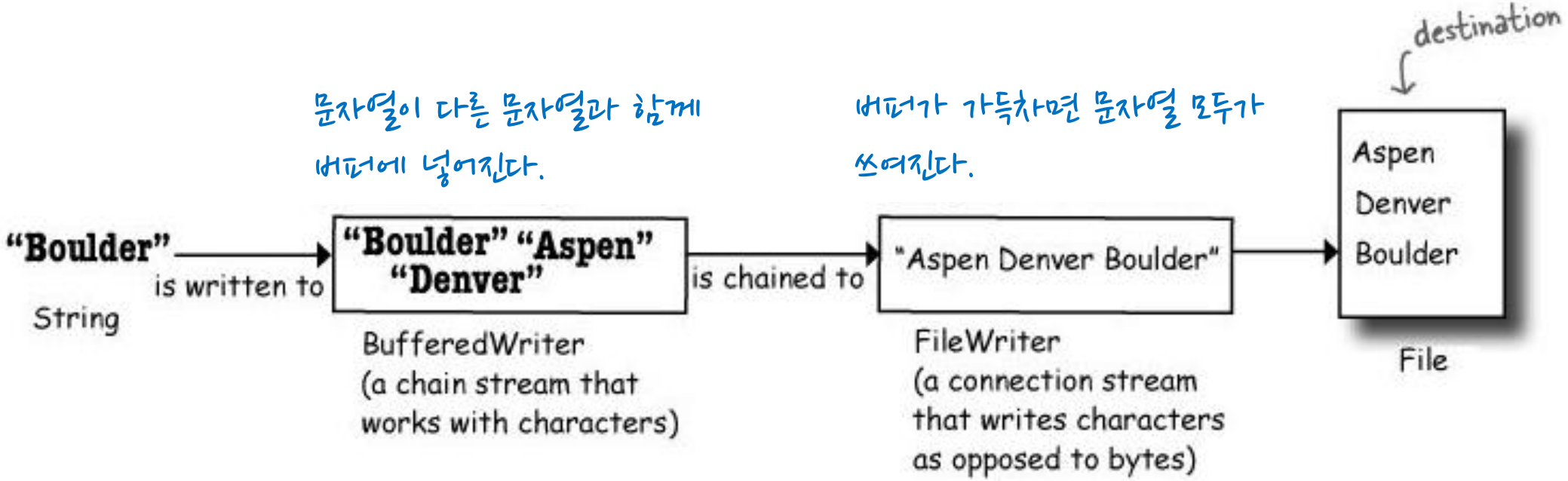
But it does NOT represent, or give you access to, the data *in* the file!



The beauty of buffers

만일 버퍼가 없다면 카트없이 쇼핑하는 것과 같다.

⇒ 물건 하나를 살 때마다 매번 하나씩 자동차로 날라야 할 것이다.




```
BufferedWriter writer = new BufferedWriter(new FileWriter(aFile));
```

Buffers give you a temporary holding place to group things until the holder (like the cart) is full. You get to make far fewer trips when you use a buffer.

Reading from a Text File

A file with two lines of text

What's 2 + 2?/4
What's 20+22/42

MyText.txt

```
import java.io.*;

class ReadAFile {
    public static void main (String[] args) {

        try {
            File myFile = new File("MyText.txt");
            FileReader fileReader = new FileReader(myFile);

            BufferedReader reader = new BufferedReader(fileReader);

            String line = null;

            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();

        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Don't forget the import.

FileReader는 문자들을 위한 연결 스트림이다.
텍스트 파일로 연결해준다.

효율적인 reading을 위해 FileReader를
BufferedReader에 체인으로 연결한다.
버퍼가 비어 있게 될 때만 파일을 읽기
위해 돌아갈 것이다.

각 라인이 읽혀지는 대로 저장
할 String 변수를 만든다.

This says, "Read a line of text, and assign it to the String variable 'line'. While that variable is not null (because there WAS something to read) print out the line that was just read."

Or another way of saying it, "While there are still lines to read, read them and print them."

Quiz Card Player (code outline)

```
public class QuizCardPlayer {

  public void go() {
     // build and display gui
  }

  class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // if this is a question, show the answer, otherwise show next question
      // set a flag for whether we're viewing a question or answer
    }
  }

  class OpenMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // bring up a file dialog box
      // let the user navigate to and choose a card set to open
    }
  }

  private void loadFile(File file) {
      // must build an ArrayList of cards, by reading them from a text file
      // called from the OpenMenuListener event handler, reads the file one line at a time
      // and tells the makeCard() method to make a new card out of the line
      // (one line in the file holds both the question and answer, separated by a "/")
  }

  private void makeCard(String lineToParse) {
     // called by the loadFile method, takes a line from the text file
     // and parses into two pieces – question and answer – and creates a new QuizCard
     // and adds it to the ArrayList called CardList
  }
}
```

Quiz Card Player

```
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;

public class QuizCardPlayer {

    private JTextArea display;
    private JTextArea answer;
    private ArrayList<QuizCard> cardList;
    private QuizCard currentCard;
    private int currentCardIndex;
    private JFrame frame;
    private JButton nextButton;
    private boolean isShowAnswer;


    public static void main (String[] args) {
       QuizCardPlayer reader = new QuizCardPlayer();
       reader.go();
    }

    public void go() {

        // build gui

        frame = new JFrame("Quiz Card Player");
        JPanel mainPanel = new JPanel();
        Font bigFont = new Font("sanserif", Font.BOLD, 24);

        display = new JTextArea(10,20);
        display.setFont(bigFont);

        display.setLineWrap(true);
        display.setEditable(false);

        JScrollPane qScroller = new JScrollPane(display);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        nextButton = new JButton("Show Question");
        mainPanel.add(qScroller);
        mainPanel.add(nextButton);
        nextButton.addActionListener(new NextCardListener());

        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        JMenuItem loadMenuItem = new JMenuItem("Load card set");
        loadMenuItem.addActionListener(new OpenMenuListener());
        fileMenu.add(loadMenuItem);
        menuBar.add(fileMenu);
        frame.setJMenuBar(menuBar);
        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(640,500);
        frame.setVisible(true);

    } // close go
```

Just GUI code on this page; nothing special

```
public class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
       if (isShowAnswer) {
          // show the answer because they've seen the question
          display.setText(currentCard.getAnswer());
          nextButton.setText("Next Card");
          isShowAnswer = false;
       } else {
          // show the next question
          if (currentCardIndex < cardList.size()) {

             showNextCard();

           } else {
             // there are no more cards!
             display.setText("That was last card");
             nextButton.setEnabled(false);
          }
       }
    }
  }

  public class OpenMenuListener implements ActionListener {
      public void actionPerformed(ActionEvent ev) {
          JFileChooser fileOpen = new JFileChooser();
          fileOpen.showOpenDialog(frame);
          loadFile(fileOpen.getSelectedFile());
      }
  }

  private void loadFile(File file) {

     cardList = new ArrayList<QuizCard>();
     try {
         BufferedReader reader = new BufferedReader(new FileReader(file));
         String line = null;
         while ((line = reader.readLine()) != null) {
            makeCard(line);
         }
         reader.close();

     } catch(Exception ex) {
         System.out.println("couldn't read the card file");
         ex.printStackTrace();
     }

    // now time to start by showing the first card
    showNextCard();
  }

  private void makeCard(String lineToParse) {
     String[] result = lineToParse.split("/");
     QuizCard card = new QuizCard(result[0], result[1]);
     cardList.add(card);
     System.out.println("made a card");
  }

  private void showNextCard() {
       currentCard = cardList.get(currentCardIndex);
       currentCardIndex++;
       display.setText(currentCard.getQuestion());
       nextButton.setText("Show Answer");
       isShowAnswer = true;
  }
} // close class
```

Check the isShowAnswer boolean flag to see if they're currently viewing a question or an answer, and do the appropriate thing depending on the answer.

Bring up the file dialog box and let them navigate to and choose the file to open.

Make a BufferedReader chained to a new FileReader, giving the FileReader the File object the user chose from the open file dialog.

Read a line at a time, passing the line to the makeCard() method that parses it and turns it into a real QuizCard and adds it to the ArrayList.

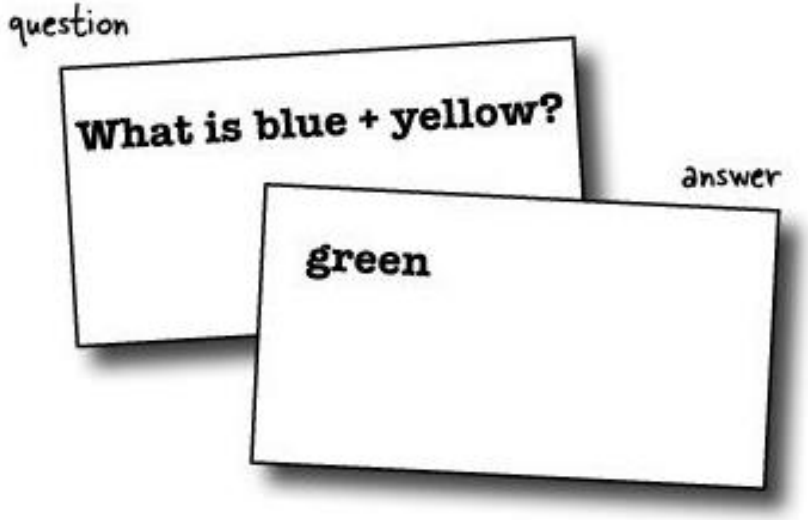
Each line of text corresponds to a single flashcard, but we have to parse out the question and answer as separate pieces. We use the String split() method to break the line into two tokens (one for the question and one for the answer). We'll look at the split() method on the next page.

QuizCard.java

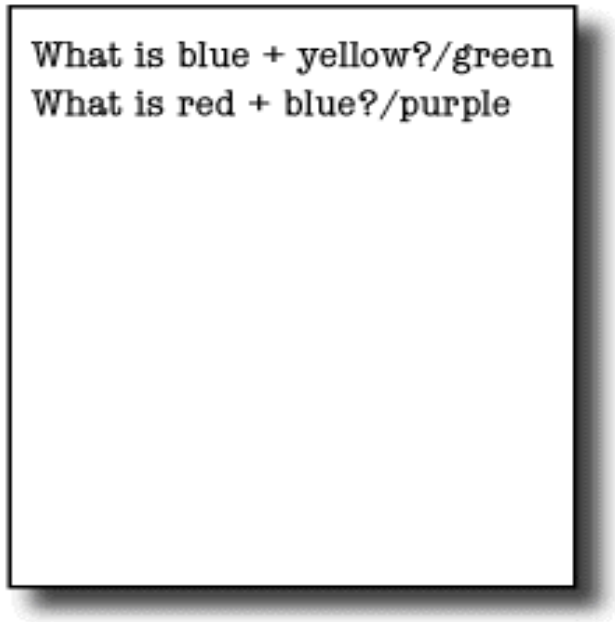
```
1  package com.skimok;
2  import java.io.*;
3
4  public class QuizCard implements Serializable {
5
6        private String uniqueID;
7        private String category;
8        private String question;
9        private String answer;
10       private String hint;
11
12       public QuizCard(String q, String a) {
13           question = q;
14           answer = a;
15       }
16
17       public void setUniqueID(String id) {
18          uniqueID = id;
19       }
20
21       public String getUniqueID() {
22          return uniqueID;
23       }
24
25       public void setCategory(String c) {
26          category = c;
27       }
28
29       public String getCategory() {
30           return category;
31       }
32
33       public void setQuestion(String q) {
34          question = q;
35       }
36
37       public String getQuestion() {
38          return question;
39       }
40
41       public void setAnswer(String a) {
42          answer = a;
43       }
44
45       public String getAnswer() {
46          return answer;
47       }
48
49       public void setHint(String h) {
50          hint = h;
51       }
52
53       public String getHint() {
54          return hint;
55       }
56 }
```

Parsing with String split()

Imagine you have a flashcard like this:



Saved in a question file like this:



Parsing with String split()

How do you separate the question and answer?

In the QuizCardPlayer app, this is what a single line looks like when it's read in from the file.

```
String toTest = "What is blue + yellow?/green";
String[] result = toTest.split("/");
for (String token:result) {
   System.out.println(token);
}
```

The split() method takes the "/" and uses it to break apart the String into (in this case) two pieces. (Note: split() is FAR more powerful than what we're using it for here. It can do extremely complex parsing with filters, wildcards, etc.)

Loop through the array and print each token (piece). In this example, there are only two tokens: "What is blue + yellow?" and "green".

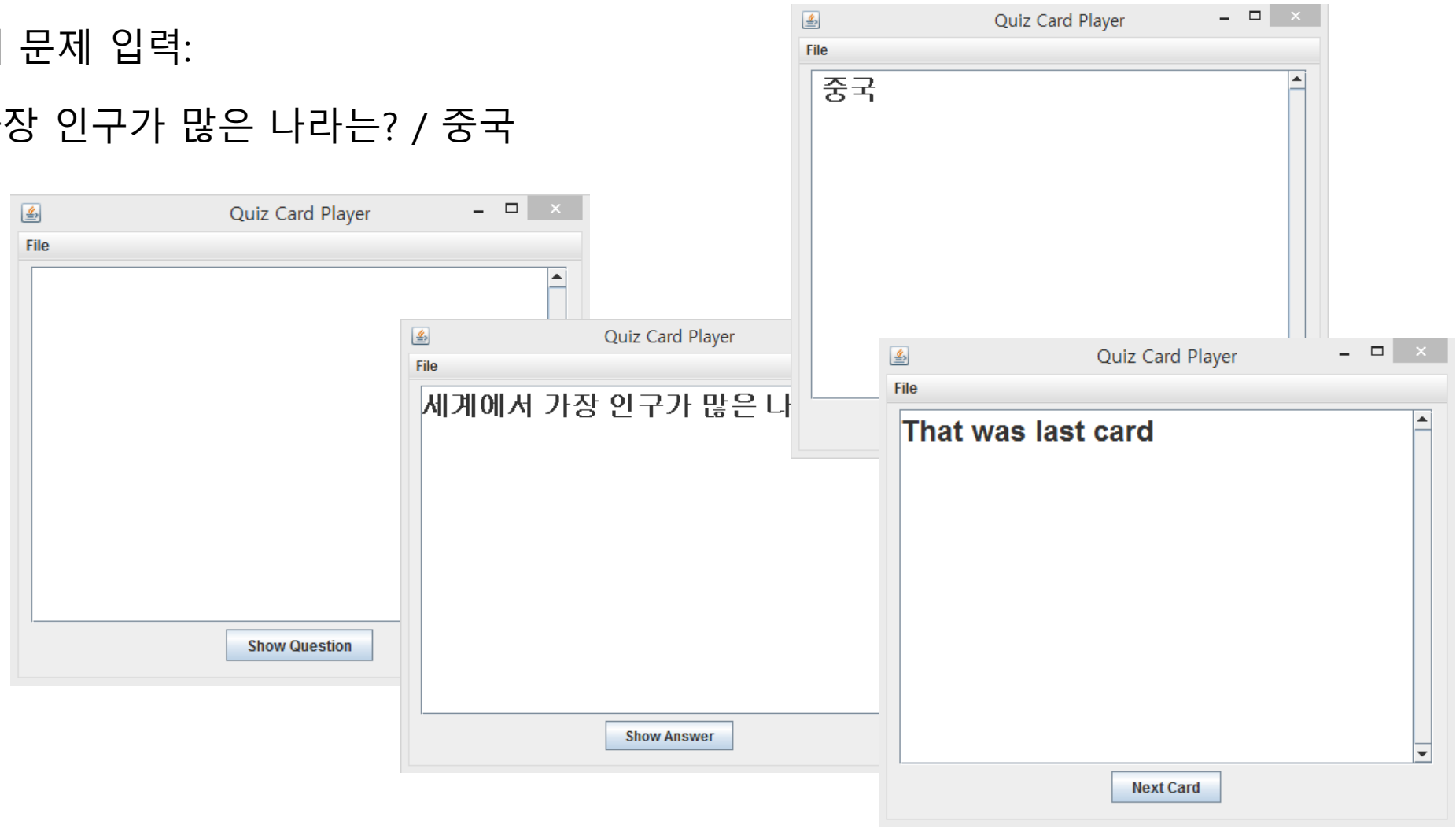
실습과제 14-2 Text File Example: e-Flashcards 구현

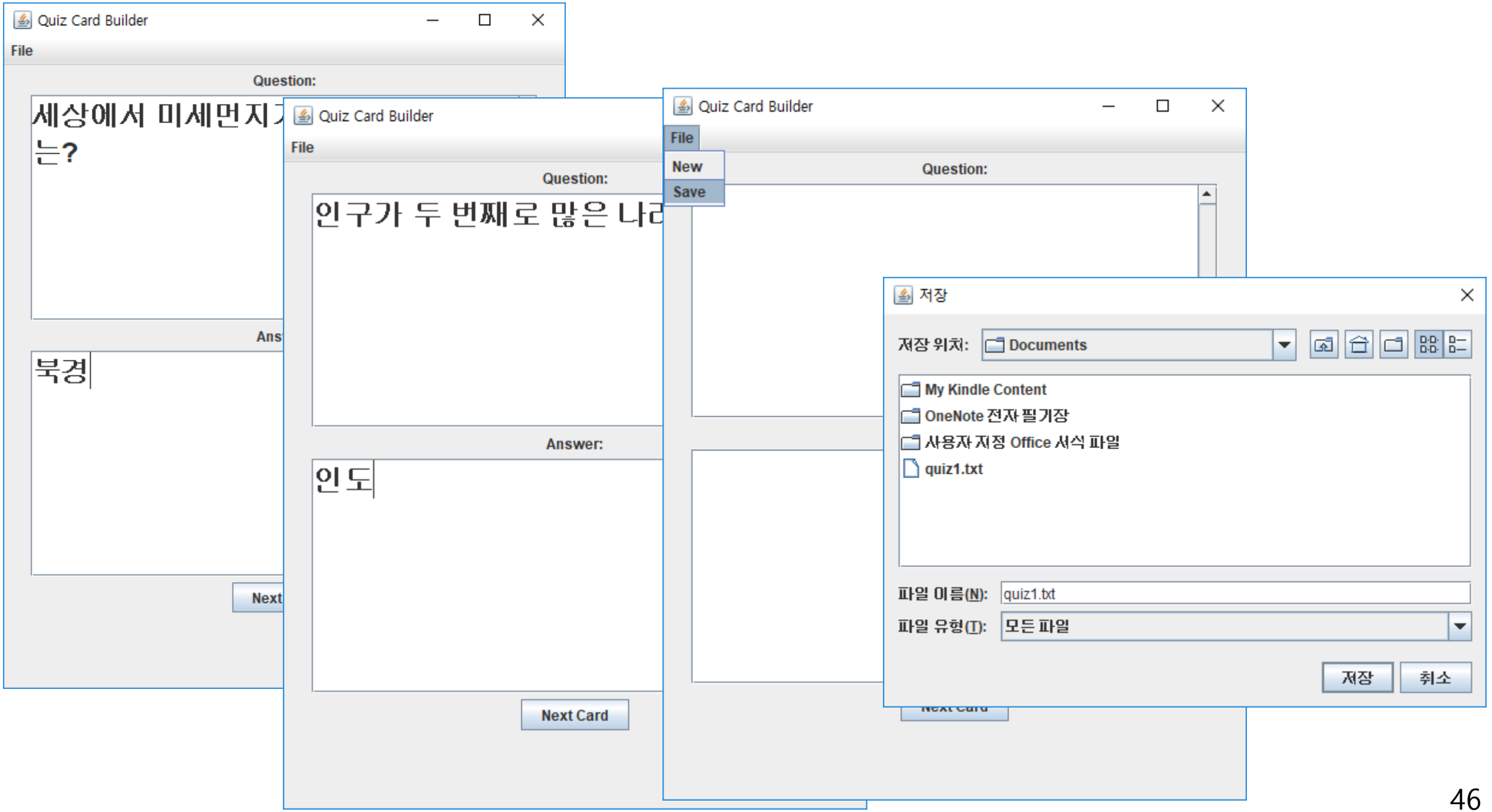
QuizCard.txt

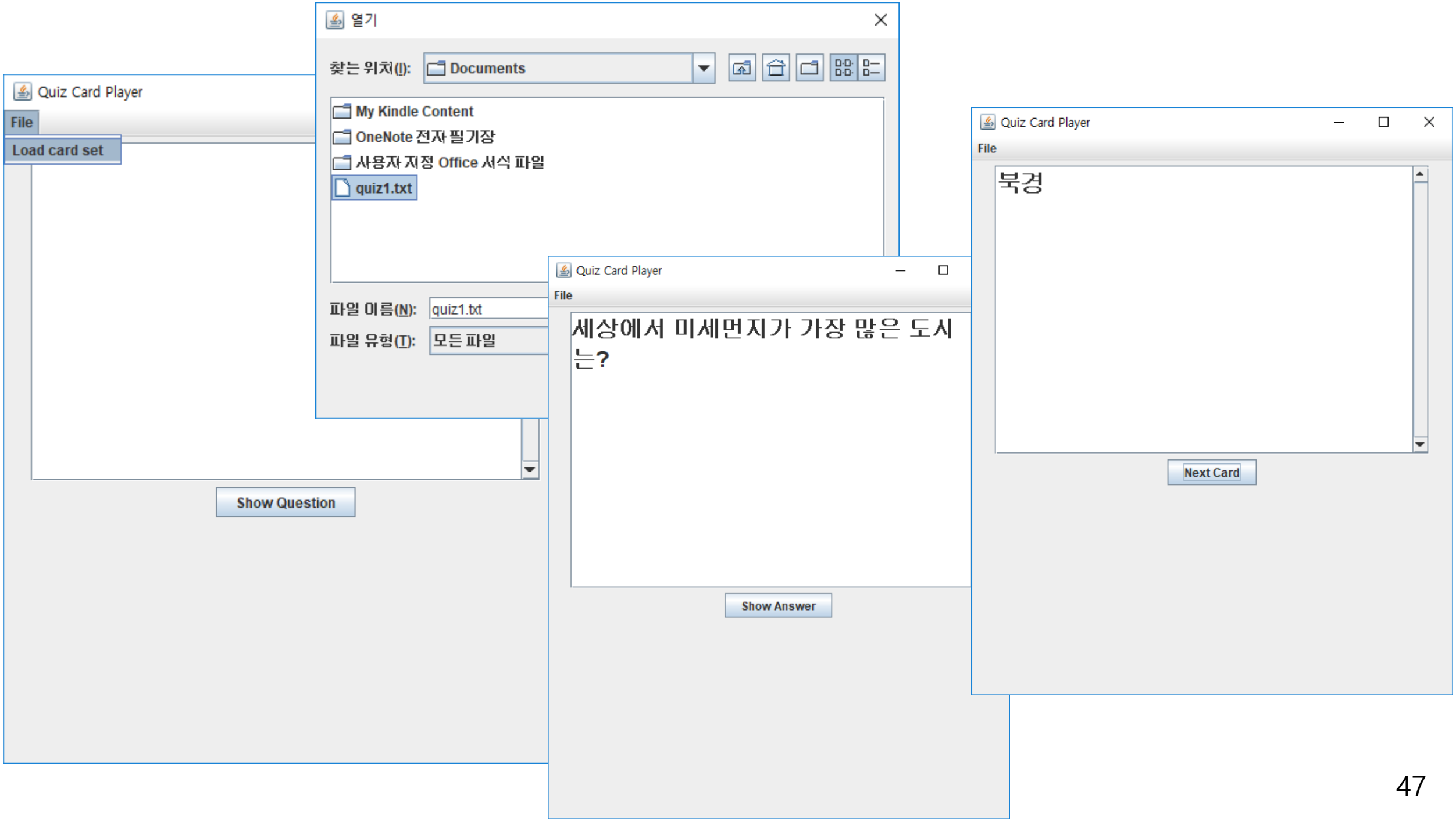
10개 이상의 문제 입력:

세계에서 가장 인구가 많은 나라는? / 중국

. . .

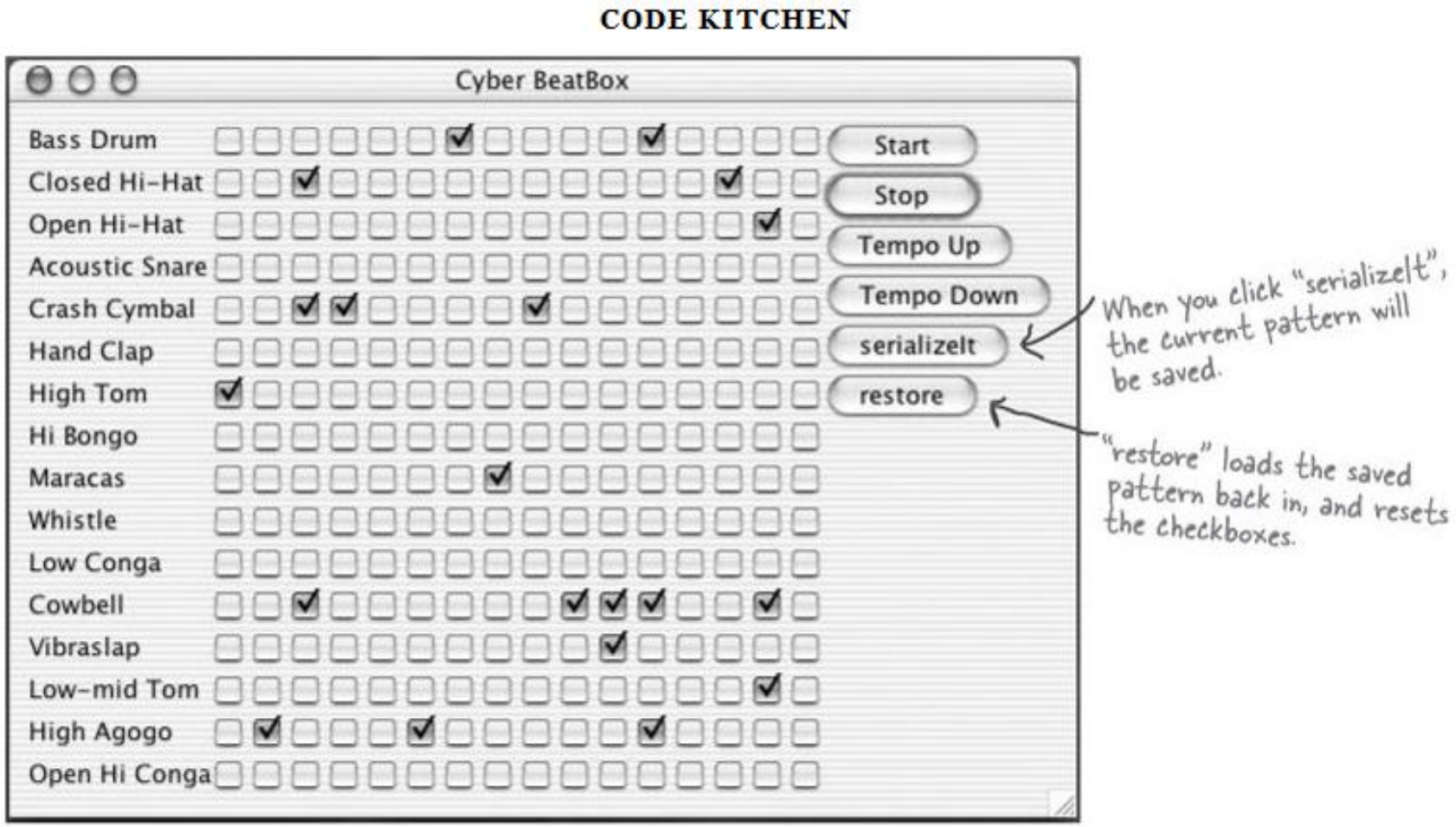



Quiz Card Builder
File
Question:
세상에서 미세먼지
는?
Ans
북경
Next
Quiz Card Builder
File
Question:
인구가 두 번째로 많은 나라
Answer:
인도
Next Card
Quiz Card Builder
File
New
Save
Question:
Next Card
저장
저장 위치:
Documents
My Kindle Content
OneNote 전자 필기장
사용자 지정 Office 서식 파일
quiz1.txt
파일 이름(N):
quiz1.txt
파일 유형(T):
모든 파일
저장
취소


Quiz Card Player
File
Load card set
Show Question
열기
찾는 위치(I):
Documents
My Kindle Content
OneNote 전자 필기장
사용자 지정 Office 서식 파일
quiz1.txt
파일 이름(N):
quiz1.txt
파일 유형(T):
모든 파일
Quiz Card Player
File
세상에서 미세먼지가 가장 많은 도시는?
Show Answer
Quiz Card Player
File
북경
Next Card

Code Kitchen

비트박스 패턴 저장/복구


CODE KITCHEN
Cyber BeatBox
Bass Drum
Closed Hi-Hat
Open Hi-Hat
Acoustic Snare
Crash Cymbal
Hand Clap
High Tom
Hi Bongo
Maracas
Whistle
Low Conga
Cowbell
Vibraslap
Low-mid Tom
High Agogo
Open Hi Conga
Start
Stop
Tempo Up
Tempo Down
serializeIt
restore
When you click "serializeIt", the current pattern will be saved.
"restore" loads the saved pattern back in, and resets the checkboxes.

Saving a BeatBox pattern

Serializing a pattern

BeatBox 코드 안에 있는 내부 클래스

```
public class MySendListener implements ActionListener {

    public void actionPerformed(ActionEvent a) {

        boolean[] checkboxState = new boolean[256];

        for (int i = 0; i < 256; i++) {

            JCheckBox check = (JCheckBox) checkboxList.get(i);
            if (check.isSelected()) {
               checkboxState[i] = true;
            }
        }

        try {
           FileOutputStream fileStream = new FileOutputStream(new File("Checkbox.ser"));
           ObjectOutputStream os = new ObjectOutputStream(fileStream);
           os.writeObject(checkboxState);
        } catch(Exception ex) {
            ex.printStackTrace();
        }

    } // close method
} // close inner class
```

사용자가 버튼을 클릭할 때 모든 일이 발생하며 ActionEvent가 점화된다.

각 체크박스의 상태를 저장할 이진 배열을 생성한다.

Walk through the checkboxList (ArrayList of checkboxes), and get the state of each one, and add it to the boolean array.

This part's a piece of cake. Just write/serialize the one boolean array!

Restoring a BeatBox pattern

Restoring a pattern

BeatBox 코드 안에 있는 내부 클래스

```
public class MyReadInListener implements ActionListener {

    public void actionPerformed(ActionEvent a) {
        boolean[] checkboxState = null;
        try {
            FileInputStream fileIn = new FileInputStream(new File("Checkbox.ser"));
            ObjectInputStream is = new ObjectInputStream(fileIn);
            checkboxState = (boolean[]) is.readObject();

        } catch(Exception ex) {ex.printStackTrace();}


        for (int i = 0; i < 256; i++) {
            JCheckBox check = (JCheckBox) checkboxList.get(i);
            if (checkboxState[i]) {
                check.setSelected(true);
            } else {
                check.setSelected(false);
            }
        }

        sequencer.stop();
        buildTrackAndStart();

    } // close method
} // close inner class
```

파일에 있는 싱글 객체를 읽어서 다시 이진 배열에 돌려준다. (readObject() 메소드는 Object 타입의 레퍼런스임을 기억하자)

실제 JCheckBox 객체의 ArrayList (checkboxList)에 있는 체크박스들의 각각의 상태를 복구한다.

현재 수행되고 있는 것이 무엇이든 간에 스톱시키고 ArrayList에 있는 체크박스들의 새로운 상태를 다시 구성한다.

실습과제 14-3 비트박스 패턴 저장/복구