네트워크 가위바위보

(2000 년 12 월 마소 Jr.)

from Yongwoo’s Park

목 차

자바 네트워크 가위바위보 게임 프로그래밍

from Yongwoo' s Park

'네트워크 가위바위보(KawuiBawuiBo)' 애플릿 게임 만들기

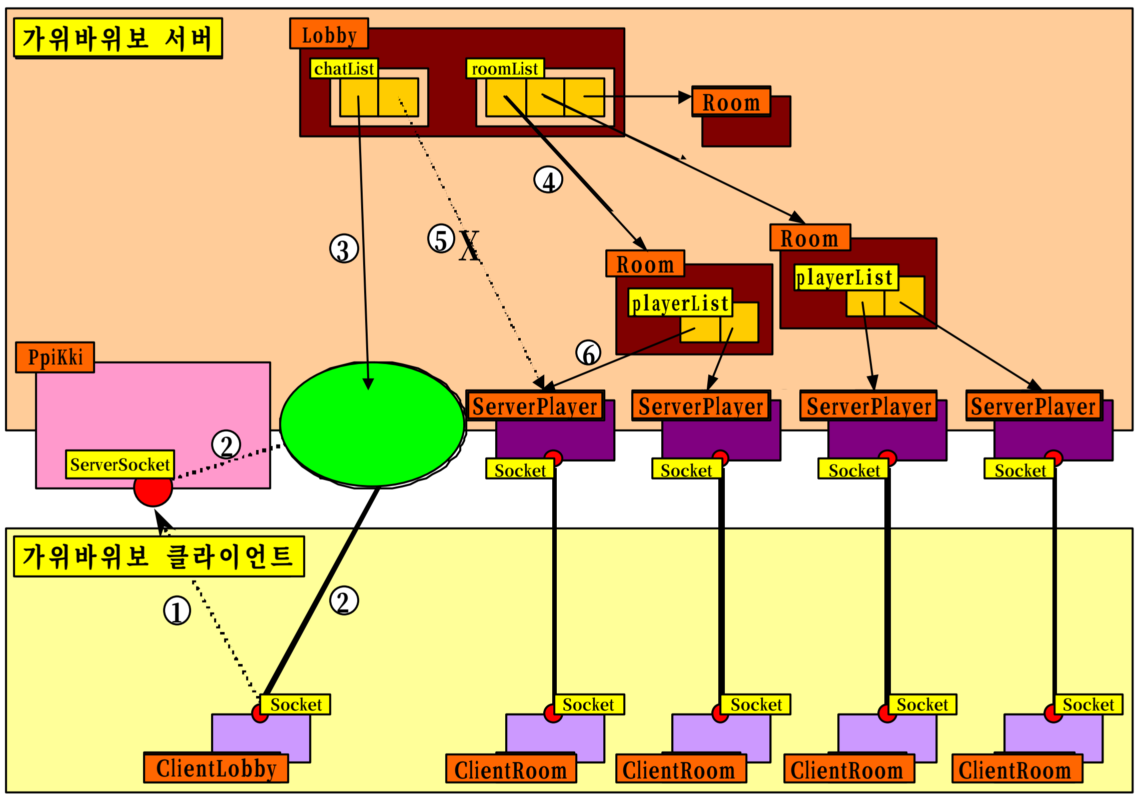
> 자! 자바를 이용한 온라인 게임을 개발하기 위해 사용할 수 있는 여러 가지 기법에 대해서 살펴본 지, 벌써 두 달이 지나고 마지막 원고를 쓰게 되었군요. 지난 두 달의 내용과 이번 달의 내용들이 여러분들에게 많은 도움이 되고, 뭔가 할 수 있는 동기가 되었으면 합니다. 이번 호에서는 다 함께 즐길 수 있는 네트워크 게임을 개발하기 위한 기법들에 대해서 살펴보기로 하겠습니다. 물론, 채팅은 기본이 되겠지요?

벌써 새천년의 마지막 달이 되었군요. 여러분은 새해에 세웠던 계획들을 어느 정도 지켰는지 궁금하군요. 필자 역시 여러 가지 세웠던 계획들이 많았지만, 지키기에 쉽지만은 않았던 한 해였습니다. 그래도, 아직은 남은 한 달이 있기에 다시 한 번 더 계획을 세우고 하나씩 마무리 하려 합니다. 여러분도 새 천년의 첫 해를 잘 마무리 하시기 바랍니다.
이번 호에서는 자바 애플릿 네트워크 게임을 개발하기 위한 다양한 기법에 대해서 살펴보도록 하겠습니다. 먼저, 자바 애플릿 네트워크 게임에서 클라이언트/서버 간의 메시지를 주고 받기 위한 프로토콜을 정의하고, 이렇게 정의된 프로토콜에 따라 클라이언트와 서버가 데이터를 주고 받기 위한 기법에 대해서 살펴보겠습니다. 그리고, 네트워크 게임을 실제로 진행하게 되는 클라이언트 즉, 사용자 화면을 작성하기 위한 기법들에 대해서도 살펴보겠습니다. 지난 호까지 관심있게 살펴본 독자라면, 이번 호에는 어떤 예제를 볼 수 있을지 기다릴지도 모르겠습니다. ^^. 이번 호에서 사용할 게임은 네트워크에서 두 명의 사용자가 가위.바위.보 게임을 하면서, 채팅도 할 수 있는 네트워크 가위바위보 게임입니다. 사용자가 게임 서버에 연결하면 대기실로 가게되고, 대기실에서는 직접 방을 만들거나 또는 이미 만들어진 방에 들어가 게임을 진행할 수 있습니다. 각 방은 두 명의 사용자만이 들어갈 수 있습니다.

1. 네트워크 가위바위보 게임 클라이언트/서버 구성

1.1 네트워크 가위바위보 게임 클라이언트/서버 흐름

그림 1. 네트워크 가위바위보 게임의 구성/흐름도



네트워크 가위바위보 게임은 기본적으로 가위바위보 서버와 가위바위보 클라이언트로 구성되어 있습니다. 그림 1은 네트워크 가위바위보 게임의 클라이언트/서버 구성을 잘 보여주고 있는 그림입니다. 네트워크 가위바위보 게임의 전체적인 흐름을 살펴보면, 다음과 같은 순서로 진행됩니다. (그림에 나와 있는 번호와 다음의 순서에 나와 있는 번호는 같은 순서를 나타내므로 잘 살펴보세요.)

① ClientLobby 에 의해 나타나는 네트워크 가위바위보 클라이언트는 네트워크 가위바위보 서버인 PpiKki 의 ServerSocket 이 accept 메소드를 이용하여 기다리고 있는 4001 포트에 접속하기 위한 소켓(Socket)을 생성함으로써, 네트워크 가위바위보 서버에 대한 연결 설정을 시도합니다.

② PpiKki 는 ServerSocket 을 이용하여 4001 포트에서 기다리고(accept) 있다가, 클라이언트가 연결 설정을 시도하면, 해당 클라이언트와의 소켓을 생성하고, 다음으로 이 클라이언트를 관리하기 위한 ServerPlayer 객체를 생성하면서 객체생성자의 매개변수로 이 소켓을 전달해 줍니다. 이 때, ServerPlayer 와 ClientLobby 사이의 네트워킹이 가능하게 됩니다.

③ PpiKki 는 연결 설정된 클라이언트를 위한 ServerPlayer 객체를 생성한 후, 이 객체를 Lobby 객체의 chatList 에 추가함으로써 클라이언트가 대기실에서 채팅하거나 또는 방을 개설하는 등의 작업을 할 수 있도록 합니다.

④ 클라이언트가 대기실에서 방만들기를 시도하면, Lobby 객체는 생성된 방을 관리하기 위한 Room 객체를 생성합니다.

⑤ Lobby 객체는 클라이언트가 방을 개설한 경우 이 방을 위한 Room 객체를 생성한 후, 이 방을 개설한 클라이언트를 위한 ServerPlayer 객체를 자신의 chatList 에서 제거 합니다. 또는, 대기실에서 클라이언트가 방을 선택하고 방들어가기를 선택한 경우, 이 클라이언트를 위한 ServerPlayer 객체를 자신의 chatList 에서 제거합니다.

⑥ Lobby 객체는 자신의 chatList 에서 제거한 ServerPlayer 객체를 해당 방을 관리하고 있는 Room 객체의 playerList 에 추가합니다. 이 때, 클라이언트의 게임 화면은 ClientRoom 에 의해 관리됩니다.

1.2 네트워크 가위바위보 클라이언트/서버 구성 클래스

네트워크 가위바위보 게임의 전체적인 흐름을 가능하도록 하기 위한 네트워크 가위바위보 클라이언트/서버 구성에서 나타나는 각 클래스들에 대해서 살펴보도록 하겠습니다. 먼저, 네트워크 가위바위보 게임의 클라이언트를 위해 제공되는 클래스들에 대해 살펴보면, 다음과 같습니다.

- ClientLobby 는 네트워크 가위바위보 게임 서버에 처음 연결 설정했을 때 대기실로 연결 설정이 이루어지는데, 클라이언트가 대기실에 있을 때를 관리하기 위한 기능을 제공해 줍니다. 클라이언트는 대기실에서 채팅, 방만들기, 방들어가기 등의 기본적인 기능을 사용할 수 있고, 방리스트 보기 및 각 방에 대한 참가자 현황 등을 볼 수 있습니다. 클라이언트가 대기실에서 새로운 방을 개설하거나 또는 특정 방을 선택하여 방들어가기를 하면, 이 때부터는 ClientRoom 클래스에 의해 관리되어집니다.

- ClientRoom 은 특정 방에 들어 있는 클라이언트를 관리하기 위한 클래스입니다. 이 때, 클라이언트는 가위바위보 게임에 참여할 수 있습니다. 여기서 클라이언트가 나가기 버튼을 클릭하게 되면, 클라이언트는 다시 대기실로 가게 되고, 이 때부터는 다시 ClientLobby 에 의해 관리되어집니다.

다음으로, 네트워크 가위바위보 게임의 서버를 구성하고 있는 클래스들을 살펴보면, 다음과 같습니다.

- PpiKki 는 클라이언트가 처음 접속하기를 기다렸다가, 클라이언트가 젒고하면 이를 처리하기 위한 클래스입니다. 이름이 의미하듯이 손님(클라이언트)이 오기를 기다리다가 (ServerSocket 의 포트 4001 에서 accept 메소드를 이용하여 클라이언트가 접속하기를 기다린다.) 클라이언트가 접속을 시도하면, 이 클라이언트와 네트워킹 하기 위한 소켓을 생성하고, 이 소켓을 ServerPlayer 객체의 객체생성자에게 매개변수로 전달하여 ServerPlayer 객체를 이용하여 실제로 클라이언트와 직접 네트워킹 할 수 있도록 합니다. 그리고, 이렇게 생성된 ServerPlayer 객체를 Lobby 객체의 chatList 벡터에 추가함으로써,

Lobby 에게 전달합니다. 이 때부터, 클라이언트는 대기실에서 채팅, 방만들기, 방들어가기 등의 작업을 할 수 있습니다.

- ServerPlayer 는 클라이언트와 네트워킹을 전담하기 위한 클래스입니다. 각 클라이언트와 연결된 소켓을 가지고 있고, 현재 이 클라이언트가 있는 방의 정보를 저장하고 있습니다. 또한, ServerPlayer 객체를 Lobby 의 chatList 또는 Room 의 playerList 에서 유지함으로써 대기실 또는 각 방에 있는 클라이언트를 쉽게 관리하고, 대기실과 방을 옮겨다니는 것을 쉽게 처리할 수 있습니다.

- Lobby 는 대기실에 있는 클라이언트와 방(Room)을 관리하기 위한 클래스입니다. 클라이언트가 처음 접속하게 되면, PpiKki 가 이 클라이언트를 위한 ServerPlayer 객체를 생성하고, 이 객체를 Lobby 의 chatList 에 추가함으로써, Lobby 는 chatList 에 있는 클라이언트만 관리하도록 하는 것입니다. 그리고, 새로운 방이 생성되면 이 방을 관리하기 위한 Room 객체를 생성하고, 방을 개설한 클라이언트를 위한 ServerPlayer 객체를 자신의 chatList 에서 제거하고 해당 방을 관리하기 위해 생성된 Room 객체의 playerList 에 추가함으로써, 해당 클라이언트가 대기실에서 방으로 이동할 수 있도록 합니다. 물론, 클라이언트가 대기실에서 방을 선택하여 들어가기를 하면, Lobby 는 해당 클라이언트를 위한 ServerPlayer 객체를 자신의 chatList 에서 제거하고 해당 Room 객체의 playerList에 추가해 줌으로써 클라이언트가 대기실에서 방으로 이동할 수 있도록 합니다.

- Room 은 각 방을 관리하기 위한 클래스입니다. Room 은 방이 개설될 때마다 새로운 객체가 생성됩니다. 그리고, Room 은 Lobby 로부터 넘겨받은 ServerPlayer 객체를 playerList 에서 유지하여 관리합니다. 이 때, 두 명의 클라이언트가 방에 들어왔다면, 가위바위보 게임을 진행하게 됩니다.

1.3 네트워크 가위바위보 게임 프로토콜

네트워크 가위바위보 게임을 만들기 위해서는 먼저 클라이언트와 서버가 주고 받을 데이터에 대한 프로토콜을 정의해야 합니다. 실제로 네트워크 가위바위보의 서버는 대기실 및 방으로 각각 구성되고, 클라이언트 역시 대기실 및 방이 각각 관리됩니다. 따라서, 네트워크 가위바위보 게임에서 클라이언트와 서버가 주고 받기 위한 데이터의 프로토콜을 다음과 같이 나누어 정의할 수 있습니다.

- 대기실 클라이언트(ClientLobby) → 대기실 서버(Lobby)

- 대기실 서버(Lobby) → 대기실 클라이언트(ClientLobby)
- 방 클라이언트(ClientRoom) → 방 서버(Room)
- 방 서버(Room) → 방 클라이언트(ClientRoom)

1.3.1 ClientLobby/Lobby 간의 메시지 프로토콜

먼저, 클라이언트(ClientLobby)가 가위바위보 게임 서버에 연결하면, 대기실(Lobby)로 들어가게 됩니다. 이 때, ClientLobby 와 Lobby 사이에 데이터를 주고 받게 되는데, 먼저 ClientLobby 와 Lobby 간에 주고 받는 데이터에 대한 프로토콜을 정의하면, 다음과 같습니다.

그림 2. ClientLobby 와 Lobby 간에 사용되는 프로토콜

로비에서 사용하는 메시지 프로토콜			
프로토콜 관계	KawuiBawuiBo ClientLobby -> KawuiBawuiBo Lobby(Server)		
프로토콜	코드	값	
브로드캐스팅 채팅	100	BROADCAST|playerId|*|chatStr	
그룹캐스팅 채팅	102	GROUPCAST|playerId|groupId|chatStr	
유니캐스팅 채팅	104	[UNICAST|PRIVATE]|playerId|toPlayerId|chatStr	
룸리스트요청	110	playerId	
사용자리스트요청	112	playerId|roomId	
방만들기	120	playerId|roomId	
방참여가기	122	playerId|roomId	
게임참가	180	JOIN|playerId	
게임종료	182	LEAVE|playerId	
프로토콜 관계	KawuiBawuiBo Lobby(Server) -> KawuiBawuiBo ClientLobby		
프로토콜	코드	값	
브로드캐스팅 채팅	300	BROADCAST|playerId|*|chatStr	
그룹캐스팅 채팅	302	GROUPCAST|playerId|groupId|chatStr	
유니캐스팅 채팅	304	[UNICAST|PRIVATE]|playerId|toPlayerId|chatStr	
룸리스트	310	playerId|roomId#1,roomId#2,…,roomId#n	
사용자리스트	312	playerId|roomId|playerId#1,playerId#2,…,playerId#n	
방만들기 결과	320	playerId|roomId|[SUCCESS|FAIL]	
방참여 결과	322	playerId|roomId|[SUCCESS|FULL|NOROOM]	
게임참가	380	JOIN|playerId	
게임종료	382	LEAVE|playerId	
메시지 출력	399	playerId|message	

1.3.2 ClientRoom/Room 간의 메시지 프로토콜

다음으로, ClientRoom과 Room 간에 데이터를 주고받을 때 사용하기 위해 정의된 프로토콜은 다음과 같습니다.

그림 3. ClientRoom와 Room 간에 사용되는 프로토콜

방에서 사용하는 메시지 프로토콜			
프로토콜 관계	KawuiBawuiBo ClientRoom -> KawuiBawuiBo Room(Server)		
프로토콜	코드	값	
브로드캐스팅 채팅	500	BROADCAST|playerId|*|chatStr	
그룹캐스팅 채팅	502	GROUPCAST|playerId|groupId|chatStr	
유니캐스팅 채팅	504	[UNICAST|PRIVATE]|playerId|toPlayerId|chatStr	
사용자리스트요청	512	playerId|roomId	
가위바위보 시작	520	START|playerId	
가위바위보 내기	522	PUT|playerId|[KAWUI|BAWUI|BO]	
점수	524	POINT|playerId|point	
방들어가기	580	JOIN_ROOM|playerId	
방나가기	582	EXIT_ROOM|playerId	
프로토콜 관계	KawuiBawuiBo Room(Server) -> KawuiBawuiBo ClientRoom		
프로토콜	코드	값	
브로드캐스팅 채팅	700	BROADCAST|playerId|*|chatStr	
그룹캐스팅 채팅	702	GROUPCAST|playerId|groupId|chatStr	
유니캐스팅 채팅	704	[UNICAST|PRIVATE]|playerId|toPlayerId|chatStr	
사용자리스트	712	playerId|roomId|playerId#1,playerId#2,…,playerId#n	
가위바위보 시작	720	START|playerId	
가위바위보 내기	722	PUT|playerId#1|[KAWUI|BAWUI|BO]|playerId#1|[KAWUI|BAWUI|BO]	
점수	724	POINT|playerId|point	
게임참가	780	JOIN_ROOM|playerId	
게임종료	782	EXIT_ROOM|playerId	
메시지 출력	799	playerId|message	

2. 클라이언트를 위한 사용자 인터페이스 작성 기법

2.1 애플릿에서 대화상자(Dialogbox) 컴포넌트 사용하기

자바에서는 대화상자(Dialogbox)를 사용할 수 있도록 하기 위해 Dialog 컴포넌트를 제공해 주고 있습니다. 따라서, Dialog 클래스를 상속하여 자신의 새로운 대화상자 컴포넌트를 정의할 수 있습니다. 이 때, 대화상자는 모달(modal) 또는 모달이 아닐(non-modal 또는 modal-less) 수 있습니다. 모달 대화상자는 대화상자가 닫힐 때까지 대화상자 이외의 다른 작업을 할 수 없게 합니다. 가장 흔히 볼 수 있는 모달 대화상자의 예를 들면, 파일을 열기 위해 또는 파일을 저장하기 위해 파일을 선

택할 수 있는 기능을 제공해주는 파일 대화상자, 또는 폰트나 색상을 바꾸도록 해 주는 대화상자 등을 들 수 있는데, 이러한 대화상자들은 작업을 마칠 때까지 부모 윈도우에서 다른 작업을 할 수 없습니다. 모달이 아닌(modal-less or non-modal) 대화상자의 예를 들면, 텍스트 찾기 또는 바꾸기 대화상자를 들 수 있습니다. 이러한 대화상자들은 대화상자가 떠 있는 동안에도 부모 윈도우에서 다른 작업을 할 수 있도록 해 줍니다. 네트워크 가위바위보 게임에서 방만들기 작업에서 방이름을 입력받기 위한 TextDialog 클래스와 방이름이 중복되었을 때 중복 메시지를 보여주기 위한 WarningDialog 클래스를 살펴보면, 다음과 같습니다.

리스트 1. TextDialog.java, WarningDialog.java

```
class TextDialog extends Dialog {
   Label msgLabel = null;
   TextField field = null;
    String strTitle=null;

   public TextDialog(Frame owner, String title, String msg) {
        super(owner, title);  // 상위 클래스의 객체생성자 호출
        setModal(true);       // 모달 대화상자 설정
        /* 컴포넌트의 생성 및 레이아웃 */
        addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
               field.setText("");
               dispose();                // 대화상자 닫기
          }
        });
        field.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent ae) {
             if(ae.getSource() == field) {
                 dispose();
             }
          }
        });
    }
}

class WarningDialog extends Dialog {
    Button okButton=null;
    Label msgLabel=null;

   public WarningDialog(Frame owner, String title, String message) {
        super(owner, title, true);  // 상위 클래스의 객체생성자 호출, modal
        /* 컴포넌트의 생성 및 레이아웃 */
        okButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
               dispose();
          }
        });
        addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
               dispose();
          }
        });
        pack();
        autoAlign();
    }
    public void autoAlign() {    // 스크린의 중앙으로 대화상자를 위치시키기 위해
       int sw = Toolkit.getDefaultToolkit().getScreenSize().width;
       int sh = Toolkit.getDefaultToolkit().getScreenSize().height;
       int dw = getSize().width;
       int dh = getSize().height;
       int x = (((sw - dw) / 2) < 0) ? 0 : ((sw - dw) / 2);
       int y = (((sh - dh) / 2) < 0) ? 0 : ((sh - dh) / 2);

         setLocation(x, y);
    }
}
```

네트워크 가위바위보 게임 클라이언트에서 위에 정의되어 있는 대화상자 컴포넌트를 띄우기 위해서는, 다음과 같이 하면 됩니다.

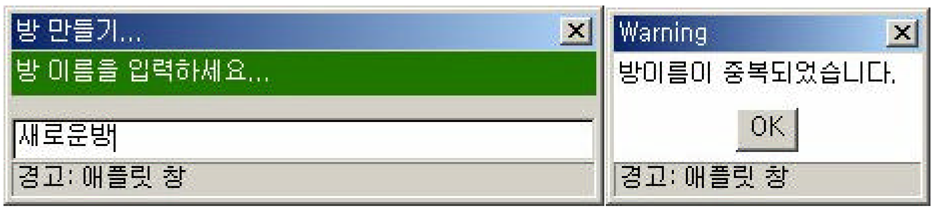
리스트 2. ClientLobby.java

```
TextDialog s = new TextDialog((Frame)(ClientLobby.this.getParent()),
                              "방 만들기...",
                              "방 이름을 입력하세요...");
s.show();
roomId = s.field.getText().trim();
if(!"".equals(roomId)) {
    /* ...*/
}
```

```
WarningDialog w = new WarningDialog((Frame)(ClientLobby.this.getParent()),
                                    "Warning",
                                    "방이름이 중복되었습니다.");
w.show();
```

다음에 나오는 그림은 네트워크 가위바위보 게임 클라이언트에서 위에 정의되어 있는 대화상자 컴포넌트를 실제로 사용할 때, 나타나는 화면입니다.

그림 4. 사용자 정의 TextDialog 컴포넌트와 WarningDialog 컴포넌트의 사용예


방 만들기...
방 이름을 입력하세요...
새로운방
경고: 애플릿 창
Warning
방이름이 중복되었습니다.
OK
경고: 애플릿 창

2.1.1 사용자 컴포넌트의 생성

자바에서 제공해주는 컴포넌트를 상속하여 자신만의 컴포넌트를 새롭게 생성할 수 있습니다. 가장 대표적으로 새로 생성할 수 있는 컴포넌트가 바로 이미지 컴포넌트 또는 이미지 버튼 컴포넌트입니다. 이렇게 자바에서 제공해 주는 컴포넌트 클래스를 상속하여 새로운 컴포넌트를 생성할 때, 다음과 같은 몇 가지 사항을 고려해 주어야 한다.

- 컴포넌트의 크기 설정: setLocation(x, y) 메소드와 setSize(width, height) 메소드를 통하여 각각의 위치 좌표와 크기 값을 설정해 주어야 합니다.
- 컴포넌트의 그리기: 새로운 컴포넌트를 생성하고, 그 컴포넌트의 paint 메소드를 재정의하여 디스플레이 되기를 원하는 형태로 디스플레이 될 수 있도록 합니다.

리스트 3. ImageComponent.java, ImageButton.java

```
class ImageComponent extends Canvas {
    Image image=null;

    public ImageComponent(Image image) {
         this.image = image;
         if(this.image != null) {
              setSize(image.getWidth(this), image.getHeight(this));
         }
     }
     public void paint(Graphics g) {
        if(image != null) {
              g.drawImage(image, 0, 0, this);
         }
     }
}

class ImageButton extends Canvas {
    private Image currentImage=null;
    private Image pressedImage=null;
    private Image releasedOffImage=null;
    private Image releasedOnImage=null;

     private ActionListener al=null;

     public ImageButton(Image releasedOff, Image releasedOn, Image pressed) {
          this.pressedImage = pressed;
         this.releasedOffImage  =  releasedOff;
         this.releasedOnImage  =  releasedOn;
         this.currentImage  =  releasedOff;
          setSize(releasedOff.getWidth(this),  releasedOff.getHeight(this));

          addMouseListener(new MouseAdapter () {
              public void mousePressed(MouseEvent e) {
                currentImage = pressedImage;
                repaint();
            }
            public void mouseReleased(MouseEvent e) {
                currentImage = releasedOffImage;
                repaint();

                ActionEvent ae = new ActionEvent(ImageButton.this,
                                        ActionEvent.ACTION_PERFORMED,
                                        "ImageButton");
                al.actionPerformed(ae);    // 등록된 ActionListener 호출
            }
            public void mouseEntered(MouseEvent e) {
                currentImage = releasedOnImage;
                repaint();
            }
            public void mouseExited(MouseEvent e) {
                currentImage = releasedOffImage;
                repaint();
            }
        });
    }

    public void addActionListener(ActionListener al) {
        this.al = al;    // ActionEvent 를 처리하기 위한 ActionListener 등록
    }

    public void paint(Graphics g) {
        if(currentImage != null) {
            g.drawImage(currentImage, 0, 0, getSize().width, getSize().height, this);
        }
    }
}
```

위의 예제에서는 네트워크 가위바위보 게임에서 이미지를 다루기 위해 정의한 ImageCanvas 클래스와 이미지 버튼을 위해 정의된 ImageButton 클래스입니다. 이미지 컴포넌트의 객체생성자에 주어진 이미지 파일 또는 이미지에 해당하는 크기로 새로 생성하는 컴포넌트의 크기를 재설정하고, paint 메소드에서는 이미지 컴포넌트가 가지고 있는 이미지를 그려주도록 하고 있습니다. 그리고, ImageButton 컴포넌트를 살펴보면, AWT 에서 제공해주는 Button 컴포넌트와 같이 ActionListener 를 등록하고, 실제로 ActionEvent 를 처리할 수 있도록 하고 있습니다. 이를 위해, addActionListener() 메소드를 이용하여 ActionListener 를 등록할 수 있고, 마우스의 버튼이 눌렸다가 놓이면 ActionEvent 객체를 생성하여, 등록된 ActionListener 를 호출하도록 되어 있습니다.

2.1.2 카드 레이아웃 관리자(CardLayout)

카드 레이아웃 관리자는 여러 장의 카드를 겹쳐 배치하는 식으로 관리하고, 컨테이너가 포함하고 있는 컴포넌트들을 한 장의 카드처럼 다루며, 한 번에 한 장씩 사용자가 보기를 원하는 카드를 보여주고, 여러 컴포넌트가 카드 레이아웃 관리자에 등록이 될 경우 가장 처음 등록된 컴포넌트를 디폴트로 보여줍니다. 다음에 나와있는 코드를 살펴보면, kawuiBawuiBo CardLayout 라는 이름의 카

드 레이아웃 관리자에 “LobbyPanel”이라는 이름으로 lobbyPanel 컴포넌트를 등록하고, “BoardPanel” 이라는 이름으로 boardPanel 컴포넌트를 cardPanel 에 등록하고 있습니다. 그리고, “방만들기(makeButton)” 또는 “방들어가기(joinButton)” 등의 버튼에서 액션 이벤트가 발생했을 경우, lobbyPanel 에서 boardPanel 로 카드를 전환하기 위해 다음과 같이 하고 있습니다.

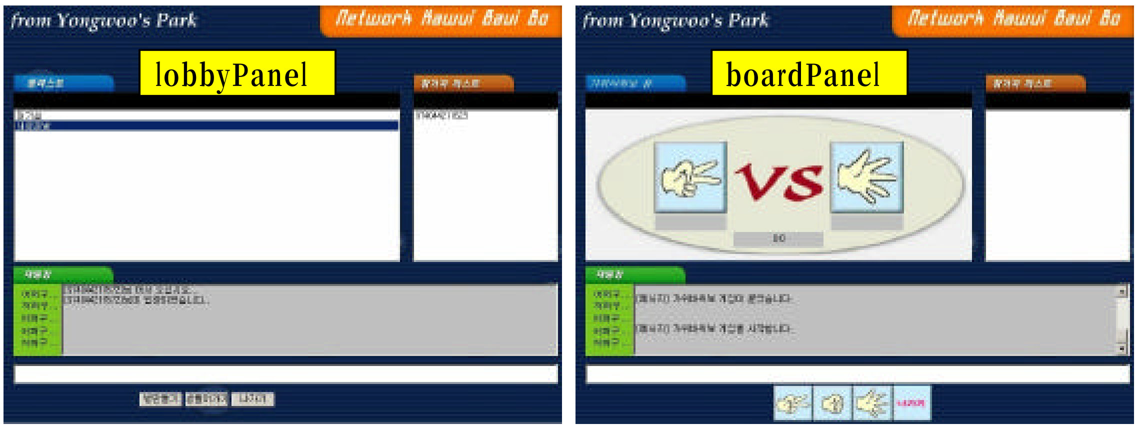
```
kawuiBawuiBoCardLayout.show(cardPanel, "BoardPanel");
```

마찬가지로, Client 의 “나가기(exitButton)”에서 액션 이벤트가 발생한 경우, boardPanel 에서 lobbyPanel 카드로 전환하기 위해 Client 객체 생성시에 객체생성자를 통해 전달받은 parentApplet 객체를 이용하여 다음과 같이 하고 있다.

```
parentApplet. kawuiBawuiBoCardLayout.show(parentApplet.cardPanel, "LobbyPanel");
```

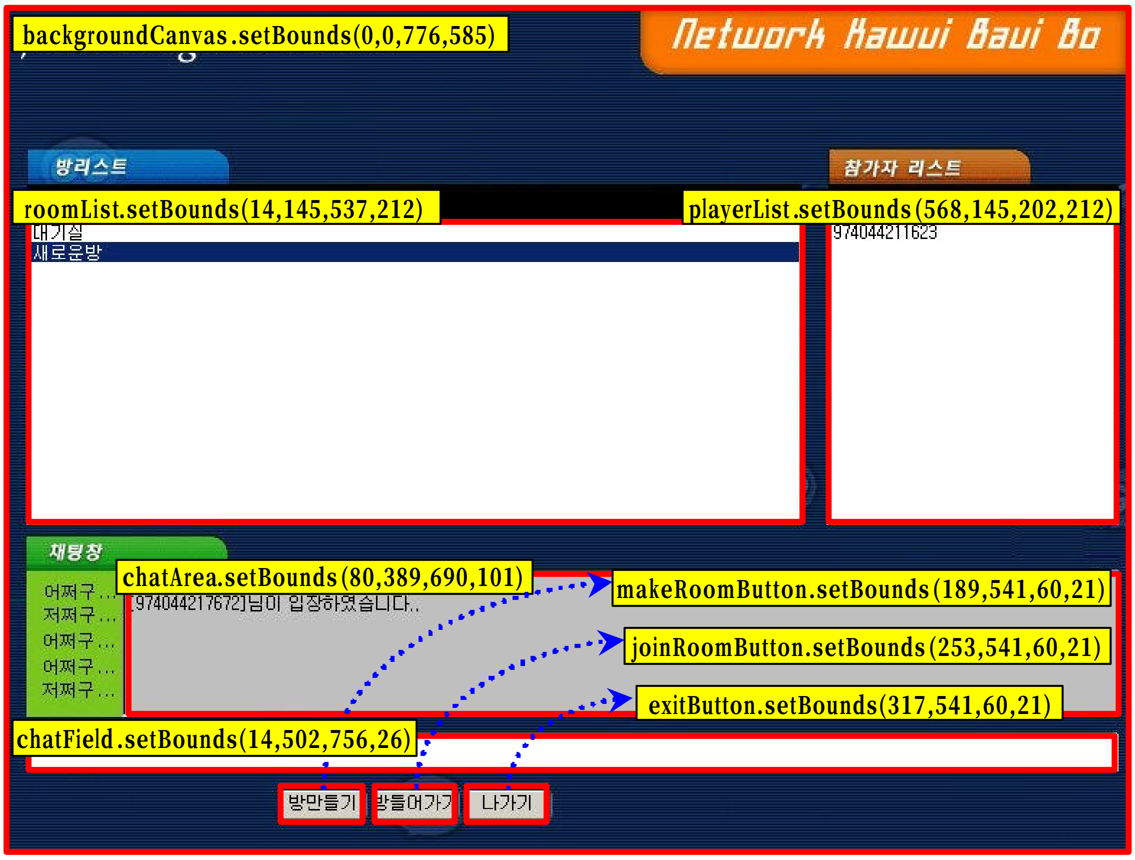
네트워크 가위바위보 게임에서 카드 레이아웃 관리자에 의해 관리되는 두 장의 카드 각각에 대한 레이아웃을 살펴보면, 다음에 나오는 그림과 같습니다.

그림 5. 네트워크 가위바위보 게임의 클라이언트에서 사용되는 두 개의 화면을 위한 Panel



2.1.3 Absolute Positioning

사용자가 원하는 위치에 직접 컴포넌트를 위치시키고 원하는 크기로 설정할 수 있어야 합니다. 그러나, 이러한 방법은 시스템의 폰트가 바뀐다거나 자바 프로그램을 실행하는 플랫폼이 바뀐다거나 할 때, 컴포넌트의 레이아웃이 처음에 의도한 것과는 다르게 나타날 수 있으므로 주의해야 합니다. 다음에 나오는 그림은 실제로 네트워크 가위바위보 게임을 위한 화면구성중 ClientLobby 를 위한 화면 구성입니다.


backgroundCanvas.setBounds(0,0,776,585)
Network Kawui Baui Bo
방리스트
참가자 리스트
roomList.setBounds(14,145,537,212)
playerList.setBounds(568,145,202,212)
대기실
새로운방
974044211623
채팅창
chatArea.setBounds(80,389,690,101)
974044217672]님이 입장하였습니다..
어쩌구...
저쩌구...
어쩌구...
어쩌구...
저쩌구...
makeRoomButton.setBounds(189,541,60,21)
joinRoomButton.setBounds(253,541,60,21)
exitButton.setBounds(317,541,60,21)
chatField.setBounds(14,502,756,26)
방만들기
방들어가기
나가기

위의 그림을 살펴보면, 배경 이미지를 위한 ImageCanvas 컴포넌트와 그 위에 여러가지 리스트 및 버튼 컴포넌트들이 레이아웃되어 있습니다. 이 때, 배경 이미지는 화면 전체에 걸쳐 있고, 나머지 컴포넌트들은 그 위에 해당 위치에 각각 레이아웃되어 있습니다. 이렇게 레이아웃 하기 위해서는 레이아웃 관리자를 null 로 설정해 줌으로써, 사용자가 직접 각 컴포넌트의 위치와 크기를 설정해 줄 수 있도록 해야 합니다. 그리고, 배경 이미지를 나타내기 위한 ImageCanvas 객체를 lobbyPanel 에 제일 나중에 추가(add)해 줌으로써, 제일 먼저 그려지도록 해야 합니다. 그렇지 않고 ImageCanvas 객체를 가장 먼저 lobbyPanel 에 추가한다면, 배경 이미지가 가장 나중에 그려짐으로써 다른 컴포넌트들이 보이지 않게 되므로 주의해야 합니다. 위의 그림에서 각 컴포넌트에 있는 4 개의 숫자는 각각 (x, y, width, height) 등에 대한 정보이다. 실제, 위와 같이 레이아웃 하기 위한 코드를 살펴보면, 다음과 같다.

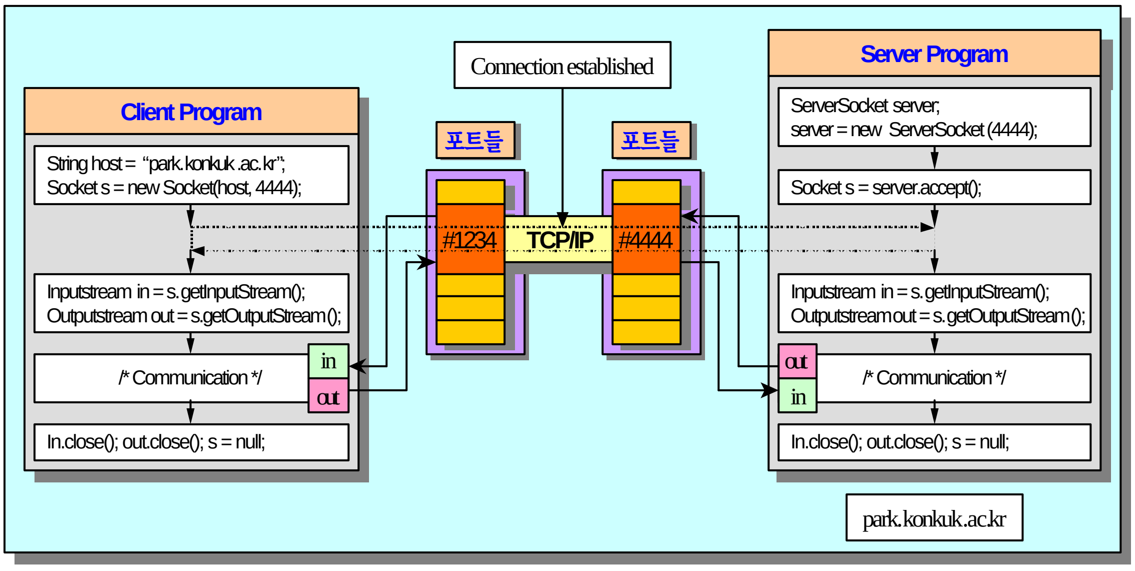
```
public class ClientLobby extends Applet implements Runnable {
   /* ...*/
   public Panel initLobbyPanel() {
       /* 레이아웃을 위한 컴포넌트 생성 */
        lobbyPanel = new Panel(null);
        backgroundCanvas.setBounds(0, 0, 776, 585);
        roomList.setBounds(14, 145, 537, 212);
        playerList.setBounds(568, 145, 202, 212);
       chatArea.setBounds(80, 389, 690, 101);
       chatField.setBounds(14, 502, 756, 26);

        lobbyPanel.add(roomList);
        lobbyPanel.add(playerList);
        lobbyPanel.add(chatArea);
        lobbyPanel.add(chatField);
        lobbyPanel.add(makeRoomButton);
        lobbyPanel.add(joinRoomButton);
        lobbyPanel.add(exitButton);
        lobbyPanel.add(backgroundCanvas);  // 배경이미지이므로 가장 마지막에 add 해야 함
    }
}
```

3. 네트워크 가위바위보 게임 클라이언트/서버 네트워킹

3.1 Socket 클래스와 ServerSocket 클래스

소켓 또는 포트는 두 개의 프로그램이 하부 네트워크에 신경쓰지 않고 통신하기 위한 소프트웨어적인 메커니즘(mechanism)입니다. 데이터를 주고 받기 위해 통신하려는 두 개의 프로그램 사이에 채널이 형성되고, 이 채널을 통해서 양방향 통신이 이루어집니다. ServerSocket 클래스는 서버 소켓(server socket)을 위한 기능을 제공해 줍니다. 일반적으로, 자바에서 제공해 주는 ServerSocket 클래스와 Socket 클래스를 이용하여 클라이언트/서버 프로그램을 작성할 경우, 대부분의 클라이언트 프로그램고 서버 프로그램은 다음에 나오는 그림과 같은 구조를 따르게 됩니다.

그림 6. 네트워킹 프로그램의 클라이언트/서버 구성도



```
String host = "park.konkuk.ac.kr";
Socket s = new Socket(host, 4444);

Inputstream in = s.getInputStream();
Outputstream out = s.getOutputStream();

/* Communication */

In.close(); out.close(); s = null;
```

```
ServerSocket server;
server = new ServerSocket(4444);

Socket s = server.accept();

Inputstream in = s.getInputStream();
Outputstream out = s.getOutputStream();

/* Communication */

In.close(); out.close(); s = null;
```

서버 프로그램은 자신의 포트 번호를 통해 들어오는 요청을 계속 듣게(listen)되며, 요청이 이 포트 번호로 들어왔을 때 이를 받아 해당 서비스를 수행하게 됩니다. 그림 6은 이러한 과정을 자세히 보여주고 있습니다. 먼저, 서버는 4444 번 포트에 대해 요청이 들어오는지를 계속 듣게(listen)되며, 이 때 클라이언트는 4444 번 포트를 통하여 서버에게 서비스를 요청하게 됩니다. 마찬가지로, 서버도 서비스를 수행한 후, 그 결과를 클라이언트가 사용하는 포트(예에서는 1234)를 통해 전달하게

됩니다. 이렇게 서버 프로그램과 클라이언트 프로그램은 각각이 사용하는 포트(소켓)를 연결함으로써 하나의 논리적인 채널을 생성하게 됩니다.

3.2 네트워크 가위바위보 게임 클라이언트/서버 네트워킹

앞에서 살펴본 것과 마찬가지로, 네트워크 가위바위보 게임은 PpiKki, ServerPlayer, Lobby, Room, ClientLobby, Client 등 크게 여섯 가지 클래스로 구성되어 있습니다. 이 때, 실제 TCP/IP 네트워킹은 PpiKki, ServerPlayer, ClientLobby 등 세 개의 클래스가 전담하고 있습니다. 다시 말해서, PpiKki 클래스는 ServerSocket 객체를 이용하여 클라이언트로부터의 요청이 들어오기를 4001 포트에서 기다리고 있습니다. 이 때, 클라이언트의 요청이 4001 포트를 통해서 들어오게 되면, 해당 클라이언트와 네트워킹을 위한 소켓을 생성하여 ServerPlayer 의 객체생성자를 통해서 전달함으로써, ServerPlayer 객체가 하나의 클라이언트와 네트워킹 할 수 있도록 합니다. 마찬가지로, ClientLobby 객체는 네트워크 가위바위보 서버의 4001 포트에 대한 서비스를 요청하기 위한 Socket 객체를 생성하게 되고, 이 Socket 객체를 이용하여 네트워크 가위바위보가 진행되는 내내 네트워크 가위바위보 서버(실제로는 담당 ServerPlayer)와 네트워크 가위바위보 프로토콜에 따라 데이터를 주고 받게 됩니다.

3.2.1 네트워크 가위바위보 게임 서버 - PpiKki 클래스

ServerSocket 을 이용한 네트워크 가위바위보 게임 서버를 구현하고 있는 PpiKki 클래스는 네트워크 가위바위보 게임 서버 프로그램으로서, main 메소드를 갖고 있습니다. main 메소드에서는 단순히 4001 포트에서 서비스를 수행하도록 하는 PpiKki 스레드 객체를 생성하고, 이 PpiKki 스레드를 시작시켜 주기만 합니다. 이 때, PpiKki 객체생성자에서 클라이언트로부터의 요청을 받아들이기 위해 4001 포트에 대해 ServerSocket 객체를 생성하고, 동시에 Lobby 스레드 객체를 생성하여 Lobby 스레드 객체를 시작시켜 줍니다. 다음으로, PpiKki 스레드의 run 메소드에서는 클라이언트의 요청을 기다리고(accept), 클라이언트로부터의 요청에 의해 생성된 소켓을 ServerPlayer 의 객체생성자의 매개변수로서 전달함으로써, ServerPlayer 가 해당 클라이언트와 네트워킹을 전담할 수 있도록 합니다. 이렇게 생성된 ServerPlayer 객체를 Lobby 스레드 객체에 addPlayer 메소드를 통해서 추가해줌으로써, Lobby 스레드 객체가 이 클라이언트를 관리할 수 있도록 해 준다. 마지막으로, 다음과 같은 코드를 통해 소켓에 대한 SO_TIMEOUT 을 지정해 줌으로써, 소켓에서 데이터를 읽어들이기 위해 블록킹 되지 않도록 함으로써, 하나의 스레드를 이용하여 여러 개의 클라이언트를 관리할 수 있도록 해 줍니다.

```
player.socket.setSoTimeout(10);   // 소켓에서 읽을 때, 10msec 로 타임아웃을 설정
```

위와 같이 SO_TIMEOUT 을 설정하는 것은 하나의 Lobby 스레드 객체가 여러 개의 클라이언트를 다루기 위한 것입니다. 다시 말해서, Lobby 스레드 객체는 클라이언트로부터 데이터가 전송되었는

지를 알아보기 위해 해당 소켓을 읽으려고 할 것이고, 이 때 데이터가 전송되지 않았다면, 데이터가 전송될 때까지 블록킹(Blocking) 되어 버립니다. 이렇게 되면 현재 소켓에 데이터가 들어온 후에만 다음 클라이언트로부터 전송되어 온 데이터를 처리할 수 있습니다. 따라서, 소켓에서 데이터를 읽어들이기 위해 블록킹 되지 않도록 TIMEOUT 을 지정하게 되는 것이고, 이를 SO_TIMEOUT 이라 합니다. 이렇게 함으로써, Lobby 스레드 객체 및 GRoom 스레드 객체는 여러 개의 클라이언트를 관리할 수 있는 것입니다.

```
public class PpiKki extends Thread {
   final static int KAWUI_BAWUI_BO_GAME_PORT=4001;
    Lobby lobby=null;
    ServerSocket  ppikkiSocket=null;

   public static void main(String args[]) {
       System.out.println("MW Game Server Running...");
        PpiKki ppikki = new PpiKki();
        ppikki.start();
    }

   public PpiKki() {
       try {
            ppikkiSocket = new ServerSocket(KAWUI_BAWUI_BO_GAME_PORT);
       } catch(IOException e) {
            System.out.println("IOException:  "+e);
            System.exit(1);
        }
        lobby = new Lobby();
        lobby.start();
    }

   public void run() {
        Socket socket=null;
       ServerPlayer  player=null;

       System.out.println("PpiKki thread is started...");
        while(true) {
           try {
              socket = ppikkiSocket.accept();
             if(socket == null) {
                 System.out.println("Can't create a socket...");
                 continue;
             } else {
                 player = new ServerPlayer(socket, lobby, null);
                if(player == null) {
                   System.out.println("Can't create a player...");
                     continue;
                 }
              }
              lobby.addPlayer(player);
          } catch(Exception e) {
              e.printStackTrace();
           }
        }
    }
}
```

3.2.2 클라이언트와의 네트워킹 모듈 - ServerPlayer 클래스

ServerPlayer 클래스는 네트워크 가위바위보 게임 클라이언트와 데이터를 주고 받기 위한 서버측 네트워크 모듈이라 할 수 있습니다. ServerPlayer 객체는 하나의 클라이언트(ClientLobby)와의 네트워크를 위한 소켓을 가지고 있는데, 이 소켓은 PpiKki 스레드 객체가 클라이언트로부터의 요청에 의해 생성된 소켓을 ServerPlayer 객체 생성시에 객체생성자의 매개변수로서 PpiKki 스레드 객체로부터 전달받은 것입니다. 이렇게 전달받은 소켓으로부터 데이터를 읽어들이기 위한 BufferedReader 객체를 얻고, 마찬가지로 소켓에 데이터를 쓰기 위해 사용할 PrintWriter 객체를 얻습니다. 이 두 입출력 스트림 객체를 이용하여 각각 receiveMessage 메소드와 sendMessage 메소드를 제공해 주는 것입니다. 이 때, receiveMessage 메소드에서는 데이터를 읽어들일 때 발생하는 예외 중에서, SocketException 객체와 IOException 객체를 자신을 호출한 쪽으로 throw 한다는 것에 주의하기 바랍니다. 이 두 가지 예외는 클라이언트와 네트워킹을 하는데 있어서 문제가 발생했다는 것을 의미하며, 이는 클라이언트가 연결을 끊고 나갔다는 것을 의미하기도 합니다. 따라서, ServerPlayer 의 receiveMessage 메소드를 이용하여 클라이언트로부터 오는 데이터를 처리하는 Lobby 객체 또는 Room 객체에서는 이에 대한 적절한 처리를 해 주어야 하며, 이는 일반적으로 removePlayer 메소드 호출함으로써 자신이 현재 관리하고 있는 클라이언트 리스트에서 해당 클라이언트를 제거하는 것과 같은 작업이 됩니다.

```
class ServerPlayer {
    public Socket socket=null;
    public BufferedReader is=null;
    public PrintWriter os=null;
    /* ...*/
     ServerPlayer(Socket socket, Lobby lobby, Room room) throws IOException {
        is = new BufferedReader(new InputStreamReader(socket.getInputStream()));
         os = new PrintWriter(socket.getOutputStream(), true);
        if(is == null) {            throw new IOException();        }
        if(os == null) {            throw new IOException();        }
         this.socket = socket;
        this.lobby = lobby;
        this.room = room;
         getInitialInfo();
         socket.setSoTimeout(10);
     }

    public void getInitialInfo() {
        if(is != null) {
            try {
                sendMessage("UserName");    // 처음 네트워크 연결 설정 시
                playerId = receiveMessage();    // 클라이언트의 아이디 읽어옴
            } catch(SocketException e) {
                clear();
            } catch(IOException e) {
                 clear();
            }
        }
    }

   public void sendMessage(String message) {
       if(message != null) {
            os.println(message);    // 클라이언트에세 메시지 보낸다(출력).
            os.flush();                 // 버퍼의 내용을 실제로 출력
            if(os.checkError()) {    // 네트워크 에러 체크
                lobby.removePlayer(this);
            }
            System.out.println("Send: "+message);
        }
    }

   public String receiveMessage() throws SocketException, IOException {
        String message=null;
        try {
            message = is.readLine();    // 클라이언트로부터 데이터를 읽어들임
          if(message == null) {        // 예외를 이용하여 통신 상태 체크 가능
              throw(new IOException("Null pointer received...."));
            }
            System.out.println("Recv:  "+message);
       } catch(SocketException e) {
            throw(e);
       } catch(InterruptedIOException e) {
            message  = "";
       } catch(IOException e) {
            throw(e);
       } catch(Exception e) {
           message  =  "";
        }
        return(message);
    }
    /* ...*/
}
```

3.2.3 대기실 관리를 위한 Lobby 클래스

Lobby 클래스는 스레드 클래스로서 대기실을 관리하는 작업을 위한 클래스입니다. Lobby 스레드 객체는 PpiKki 객체가 생성되면서 같이 생성됩니다. PpiKki 스레드 객체는 자신이 서비스하고 있는 4001 포트에 클라이언트의 연결 설정이 이루어지면, 이 클라이언트를 위한 ServerPlayer 객체를 생성하고, 이 객체를 Lobby 스레드 객체의 chatList 에 추가시킵니다. 이렇게 함으로써, Lobby 스레드

객체는 자신이 chatList 를 통해서 관리하고 있는 각 클라이언트들로부터 들어온 데이터를 각 클라이언트를 위한 ServerPlayer 객체의 receiveMessage 메소드를 통해서 읽게 되고, 이 데이터를 해당 프로토콜에 맞게 해석하여 적당한 작업을 하게 되는 것입니다.

```
public class Lobby extends Thread {
    Hashtable chatList=null;    // 대기실에서 대기중인 클라이언트 리스트
    Hashtable roomList=null;    // 만들어진 방 리스트

   public Lobby() {
       chatList = new Hashtable();
       roomList = new Hashtable();
    }

   // 클라이언트가 게임 서버에 처음 접속할 경우, 대기실에 추가
   public void addPlayer(ServerPlayer player) {
      if(player.playerId != null && chatList.get(player.playerId) == null){
           String message=null;
            chatList.put(player.playerId, player);

           message = "300|UNICAST|"+player.playerId+"|"
                      +player.playerId+"|["+player.playerId+"]님 어서 오십시요...";
            player.sendMessage(message);

           message = "300|BROADCAST|"+player.playerId
                      +"|*|["+player.playerId+"]님이 입장하였습니다..";
            broadcastMessage(message);

            sendRoomList(player.playerId);
            broadcastUserList("대기실");
       }
    }
   // 방에 들어가 있는 클라이언트가 다시 대기실로 이동
   public void takePlayerFromRoom(ServerPlayer player) {
       chatList.put(player.playerId, player);
    }
   // 게임을 마치고 또는 네트워크 연결 불량으로 빠져나감
   public void removePlayer(ServerPlayer player) {
        chatList.remove(player.playerId);
        broadcastUserList("대기실");
    }
   // ServerPlayer 객체를 이용하여 메소지 보내기
   public void sendMessage(String playerId, String message) {
       ServerPlayer player = (ServerPlayer)chatList.get(playerId);
       if(player != null) {
            player.sendMessage(message);
       }
```

```
  }
  // 방만들기
  public void makeRoom(String playerId, String roomId) {
      ServerPlayer player = (ServerPlayer)chatList.get(playerId);
      Room room = new Room(roomId, player);
      roomList.put(roomId, room);
      room.start();
  }
  // 방 삭제하기
  public void removeRoom(String roomId) {
      Room room = (Room)roomList.get(roomId);
      if(room != null) {
          roomList.remove(roomId);
          room = null;
          broadcastRoomList();
      }
  }
  public void broadcastMessage(String message) {  /* 소스 참조 */    }
  public void broadcastRoomList() {  /* 소스 참조 */    }
  public void sendRoomList(String playerId) {  /* 소스 참조 */    }
  public void sendUserList(String playerId, String roomId) {  /* 소스 참조 */    }
  public void broadcastUserList(String roomId) {  /* 소스 참조 */    }
  public boolean enterRoom(String playerId, String roomId) {  /* 소스 참조 */  }
  // 클라이언트로부터 온 메시지를 스레드로 처리..
  public void run() {
      while(true) {
          // Read Data
          try {
              sleep(100);
          } catch(InterruptedException ie) {
          }
          // 대기실에 있는 모든 클라이언트에서 메시지가 있는지 체크
          Enumeration e = chatList.elements();
          while(e.hasMoreElements()) {
              ServerPlayer player = (ServerPlayer)e.nextElement();
              try {
                  String receive = null;
                  receive = player.receiveMessage();
                  if(receive == null || "".equals(receive)) {
                      continue;
                  }
                  StringTokenizer st = new StringTokenizer(receive, "|");
                  String command = st.nextToken();
                  // 정해 놓은 프로토콜에 따라, 메시지 처리
                  if("100".equals(command)) {
                      // 브로드캐스팅 채팅 100|BROADCAST|playerId|*|chatStr
                      String type = st.nextToken();
                      String playerId = st.nextToken();
```

```
                        String target = st.nextToken();
                        String chatStr = st.nextToken();
                        String message = "300|BROADCAST|"
                                    +playerId+"|*|["+player.playerId+"] "+chatStr;
                        broadcastMessage(message);
                    } else if("102".equals(command)) {
                    } else if("104".equals(command)) {
                    } else if("110".equals(command)) {
                        String playerId = st.nextToken();
                        sendRoomList(playerId);
                    } else if("112".equals(command)) {
                        // 사용자리스트요청  112|playerId|roomId
                        String playerId = st.nextToken();
                        String roomId = st.nextToken();
                        sendUserList(playerId, roomId);
                    } else if("120".equals(command)) {
                        // 방만들기          120|playerId|roomId
                        String playerId = st.nextToken();
                        String roomId = st.nextToken();
                        makeRoom(playerId, roomId);
                        broadcastRoomList();
                    } else if("122".equals(command)) {
                        // 방참여가기        122|playerId|roomId
                    } else if("180".equals(command)) {
                        // 게임참가          180|JOIN|playerId
                    } else if("182".equals(command)) {
                        // 게임종료          182|LEAVE|playerId
                    }
                } catch(SocketException ne) {
                    removePlayer(player);
                } catch(IOException ne) {
                    removePlayer(player);
                }
            }
        }
    }
    /* ...*/
}
```

3.2.4 방 관리를 위한 Room 클래스

Room 클래스는 클라이언트가 생성한 하나 하나의 방을 관리하기 위한 스레드 클래스입니다. Room 스레드 객체는 클라이언트가 방을 생성할 경우에 생성되어 시작되게 됩니다. 클라이언트가 대기실에서 방만들기를 요청하게 되면, Lobby 스레드 객체는 해당 메시지를 받아, 새로운 Room 스레드 객체를 생성합니다. 그리고, 자신의 chatList 에서 요청한 클라이언트를 위한 ServerPlayer 객체

를 제거한 후, 그 객체를 Room 스레드 객체의 playerList 에 추가해 줍니다. 이렇게 함으로써, Lobby 스레드 객체가 관리하던 클라이언트를 Room 스레드 객체가 관리하게 할 수 있고, 이는 클라이언트로 하여금 방에 들어온 것과 같이 느낄 수 있도록 하는 것입니다. 여러 개의 클라이언트를 하나의 Room 스레드 객체가 관리하는 것은 대기실을 관리하기 위해 제공되는 Lobby 스레드 객체와 같습니다. 왜냐하면, 대기실이라는 것은 특수한 형태의 하나의 방이기 때문입니다. Room 스레드 객체 역시 자신이 관리하고 있는 각 클라이언트들로부터 들어온 데이터를 각 클라이언트를 위한 ServerPlayer 객체의 receiveMessage 메소드를 통해서 읽게 되고, 이 데이터를 해당 프로토콜에 맞게 해석하여 적당한 작업을 하게 되는 것입니다.

```
class Room extends Thread {
    String  roomId=null;
     ServerPlayer hostPlayer=null;
    Vector  playerList=null;
    /* …*/
   public Room(String roomId, ServerPlayer player) {
        this.roomId = roomId;
        this.hostPlayer = player;
        playerList = new Vector();
         addPlayer(player);
    }

    synchronized  public void addPlayer(ServerPlayer player) {
        if(!playerList.contains(player))  {
             playerList.addElement(player);
            player.room = this;
             broadcastUserList();
            if(playerList.size() == 2) {
                 startGame();
             }
         }
    }

   synchronized public void removePlayer(ServerPlayer player) {
        if(player == null) {
             return;
         }
        player.room = null;
        if(playerList.contains(player))  {
             playerList.removeElement(player);
             player.lobby.takePlayerFromRoom(player);
            if(playerList.size() == 0) {
                 player.lobby.removeRoom(roomId);
             }
             broadcastUserList();
         }
    }
```

```
public void run() {
    String receive=null, message=null;
    Enumeration enum=null;
    ServerPlayer player=null;

    while(true) {
      // Read Data
       try {
           Thread.sleep(50);
      } catch(InterruptedException ie) {
       }
      enum = playerList.elements();
       while(enum.hasMoreElements()) {
         player = (ServerPlayer)enum.nextElement();
          try {
             receive = player.receiveMessage();
            if(receive == null || "".equals(receive)) {
                 continue;
              }
             // MW Game Protocol
             // nnn|arg1|arg2|arg3|...
            StringTokenizer st = new StringTokenizer(receive, "|");
             String command = st.nextToken();
             if("500".equals(command)) {
                // 브로드캐스팅 채팅 500|BROADCAST|playerId|*|chatStr
             } else if("502".equals(command)) {
             } else if("504".equals(command)) {
             } else if("512".equals(command)) {
                // 사용자리스트        512|playerId|roomId
             } else if("520".equals(command)) {
                // 게임시작         520|START|playerId
             } else if("522".equals(command)) {
                // 게임결과        522|PUT|playerId|[KAWUI|BAWUI|BO]
             } else if("524".equals(command)) {
                // 점수          524|POINT|playerId|point1
             } else if("580".equals(command)) {
                // 게임참가              580|JOIN_ROOM|playerId
             } else if("582".equals(command)) {
                // 게임종료              582|EXIT_ROOM|playerId
              }
        } catch(SocketException e) { //Stream doesn't exist.
              removePlayer(player);
        } catch(IOException e) { //Stream doesn't exist.
              removePlayer(player);
         } catch(Exception e) {
           }
       }
    }
    }
}
```

3.2.5 클라이언트 대기실을 위한 ClientLobby 클래스

ClientLobby 클래스는 Runnable 인터페이스를 구현하고 Applet 클래스를 상속하고 있는 애플릿 스레드입니다. ClientLobby 클래스는 가위바위보 서버와 데이터를 주고 받기 위한 네트워크 모듈도 제공해 주고 있습니다. 또한, 대기실을 위한 사용자 인터페이스를 제공해 주고 있습니다. 그리고, ClientLobby 스레드는 Client 의 processMessage 메소드를 호출함으로써, 실제로 클라이언트는 대기실 및 가위바위보 진행을 위한 두 개의 사용자 인터페이스를 가지지만 하나의 스레드로 메시지를 처리하도록 하고 있습니다. 이렇게 함으로써 클라이언트의 부담을 크게 줄일 수 있습니다. 먼저, ClientLobby 스레드 애플릿은 init 메소드에서 네트워크 가위바위보 서버에 연결을 시도하게 되고, 데이터를 주고 받기 위한 BufferedReader 객체와 PrintWriter 객체를 연결 설정된 소켓에서 얻음으로써 네트워크 I/O 를 할 수 있도록 합니다. 또한, 자신 및 Client 를 위해 sendToServer 메소드와 receiveMessage 메소드를 제공해 주고 있습니다. ClientLobby 애플릿은 스레드로 동작하므로, start 와 stop 메소드에서 각각 자신의 스레드를 시작하고 멈추도록 하고 있습니다. 마지막으로, 스레드의 몸체인 run 메소드에서는 네트워크 가위바위보 서버로부터 온 메시지를 읽어, 해당 메시지를 ClientLobby 객체와 Client 객체가 적절하게 처리할 수 있도록 하기 위해 각각의 processMessage 메소드를 호출하고 있습니다. 물론, ClientLobby 의 processMessage 메소드 내에서는 자신이 처리해야 할 메시지만 처리하게 됩니다.

```
public class ClientLobby extends Applet implements Runnable {
    static String KAWUI_BAWUI_BO_GAME_SERVER;
    final static int KAWUI_BAWUI_BO_GAME_PORT=4001;
    public Socket socket=null;
    public BufferedReader is=null;
    public PrintWriter os=null;

    public void init() {
        KAWUI_BAWUI_BO_GAME_SERVER = getCodeBase().getHost();
        kawuiBawuiBoCardLayout = new CardLayout();
        cardPanel = new Panel(kawuiBawuiBoCardLayout);

        lobbyPanel = initLobbyPanel();
        boardPanel = new ClientRoom(ClientLobby.this);

         connect();

         cardPanel.add(lobbyPanel, "LobbyPanel");
         cardPanel.add(boardPanel, "BoardPanel");

         setLayout(new BorderLayout());
```

```
      add("Center", cardPanel);

      kawuiBawuiBoCardLayout.show(cardPanel, "LobbyPanel");
  }

public Panel initLobbyPanel() {
      /* ...*/
       return(lobbyPanel);
  }

private void connect() {
       playerId = ""+new Date().getTime();     //  아이디 자동 생성

      try {
          socket = new Socket(KAWUI_BAWUI_BO_GAME_SERVER,
                                    KAWUI_BAWUI_BO_GAME_PORT);
         is = new BufferedReader(new InputStreamReader(socket.getInputStream()));
          os = new PrintWriter(socket.getOutputStream(), true);

           String receive=null;
          StringTokenizer st = null;
          String command = null;
           try {
              receive = receiveMessage();   // "UserName" is read
                sendToServer(playerId);
                socket.setSoTimeout(10); // 10ms 만 기다림
         } catch(Exception e) { //Unknown error. Throw tantrum.
                chatArea.append("예외발생: "+e+"\n");
             }
      } catch(UnknownHostException e) {
            chatArea.append("예외발생: "+e+"\n");
            stop();
            return;
       }  catch(SocketException e) {
            chatArea.append("예외발생: "+e+"\n");
            stop();
            return;
      } catch(IOException e) {
            chatArea.append("예외발생: "+e+"\n");
            stop();
            return;
       }
  }
public synchronized void start() {
      /* ...*/
  }
public synchronized void stop() {
      /* ...*/
```

```
 }
public void sendToServer(String message) {
    if(message != null) {
          os.println(message);
          os.flush();
         if(os.checkError()) {
          }
     }
 }

public String receiveMessage() throws SocketException {
    String message = null;
     try {
         message = is.readLine();
    } catch(SocketException e) {
          throw(e);
    } catch(Exception e) {
         return "";
     }
     return message;
 }

public void run() {
     String receive=null;
    while(Thread.currentThread() == clientLobbyThread) {
         try {
              Thread.sleep(100);
        } catch(InterruptedException e) {
          }
         try {
              receive = receiveMessage();
            if((receive == null)||("".equals(receive))) {
                  continue;
              }
              processMessage(receive);
              boardPanel.processMessage(receive);
        } catch(SocketException e) { //Stream doesn't exist.
              stop();
              return;
       } catch(Exception e) { //Unknown error. Throw tantrum.
          }
     }
 }

public void processMessage(String receive) {
   StringTokenizer st = new StringTokenizer(receive, "|");
    String command = st.nextToken();
     if("300".equals(command)) {            // 브로드캐스팅 채팅
         // 브로드캐스팅 채팅  300|BROADCAST|playerId|*|chatStr
```

```
            String type = st.nextToken();
            String playerId = st.nextToken();
            String target = st.nextToken();
             String chatStr = st.nextToken();

             chatArea.append(chatStr+"\n");
        } else if("302".equals(command)) {          // 브로드캐스팅 채팅
        /* ...*/
         }
    }
}
```

3.2.6 가위바위보 진행을 위한 Client 클래스

Client 클래스는 가위바위보 진행을 위한 사용자 인터페이스를 제공하고, 가위바위보 진행시 주고받는 데이터를 처리하기 위한 모듈을 제공해 주는 클래스입니다. Client 객체는 Panel 클래스를 상속함으로써 카드 레이아웃 관리자에 의해 관리되는 한 장의 카드 패널을 위한 레이아웃을 정의하고 있습니다. 이는 물론 가위바위보 진행을 위한 사용자 인터페이스를 제공하기 위한 것입니다. 그리고, Client 는 객체생성시에 객체생성자의 매개변수로서 ClientLobby 객체를 전달받게 됩니다. 이는 애플릿 컨텍스트를 얻어 대화상자를 띄우거나 브라우저와의 통신 등의 작업을 하기 위한 것입니다. 또한, 네트워크 가위바위보 서버에 데이터를 보내거나 읽기 위한 네트워크 모듈로서 사용할 수 있습니다. 마지막으로, Client 의 processMessage 메소드는 ClientLobby 의 run 으로부터 호출됨으로써 네트워크 가위바위보 서버로부터 전달되는 가위바위보 진행 관련 메시지를 처리할 수 있습니다.

```
public class ClientRoom extends Panel {
    ClientLobby parentApplet=null;
    /* ...*/
   public ClientRoom(ClientLobby parentApplet) {
        this.parentApplet = parentApplet;
         init();
    }
   public void init() {
        /* ...*/
       ActionListener al = new ActionListener() {
           public void actionPerformed(ActionEvent e) {
                String message=null;
               if(e.getSource() == chatField) {
                    String chatStr = chatField.getText();
                     if(!"".equals(chatStr)) {
                          message = "500|BROADCAST|"
                                   +parentApplet.playerId+"|*|"+chatStr;
                          sendToServer(message);
                }
                chatField.setText("");
            } else if(e.getSource() == kawuiButton) {
                message = "522|PUT|"+parentApplet.playerId+"|KAWUI";
                sendToServer(message);
                messageBoard("가위를 냈습니다.");
                setButtonEvent(false);
            } else if(e.getSource() == bawuiButton) {
                message = "522|PUT|"+parentApplet.playerId+"|BAWUI";
                sendToServer(message);
                messageBoard("바위를 냈습니다.");
                setButtonEvent(false);
            } else if(e.getSource() == boButton) {
                message = "522|PUT|"+parentApplet.playerId+"|BO";
                sendToServer(message);
                messageBoard("보를 냈습니다.");
                setButtonEvent(false);
            } else if(e.getSource() == exitButton) {
                sendToServer("582|EXIT_ROOM|"+parentApplet.playerId);
                parentApplet.kawuiBawuiBoCardLayout.show(parentApplet.cardPanel,
                                                      "LobbyPanel");
                parentApplet.buttonDefaultSetting();
             }
          }
       };

       chatField.addActionListener(al);
       kawuiButton.addActionListener(al);
       bawuiButton.addActionListener(al);
       boButton.addActionListener(al);
       exitButton.addActionListener(al);
       /* ...*/
      // 더블버퍼링을 위한 이미지
      offImage = parentApplet.createImage(parentApplet.getSize().width,
                                   parentApplet.getSize().height);
      offGraphics = offImage.getGraphics();
   }
   // 그래픽 기법
   public void update(Graphics g) {
       paint(g);
   }
   public void paint(Graphics g) {
      offGraphics.drawImage(backgroundImage, 0, 0, this);
       offGraphics.drawImage(kawuiImage, 110, 190, 100, 100, this);
      offGraphics.drawImage(bawuiImage, 355, 190, 100, 100, this);
       switch(myX) {
         case 0: offGraphics.drawImage(kawuiImage, 110, 190, 100, 100, this); break;
          case 1: offGraphics.drawImage(bawuiImage, 110, 190, 100, 100, this); break;
         case 2: offGraphics.drawImage(boImage, 110, 190, 100, 100, this); break;
```

```
            default:
                offGraphics.setColor(Color.lightGray);
              offGraphics.fillRect(110, 190, 100, 100);
            break;
        }
        switch(yourX) {
          case 0: offGraphics.drawImage(kawuiImage, 355, 190, 100, 100, this); break;
          case 1: offGraphics.drawImage(bawuiImage, 355, 190, 100, 100, this); break;
          case 2: offGraphics.drawImage(boImage, 355, 190, 100, 100, this); break;
            default:
                offGraphics.setColor(Color.lightGray);
              offGraphics.fillRect(355, 190, 100, 100);
            break;
        }
       g.drawImage(offImage, 0, 0, this);
    }

  public void messageBoard(String message) {
       chatArea.append("\n[메시지] "+message+"\n\n");
    }

  public String receiveMessage() throws SocketException {
       try {
            return(parentApplet.receiveMessage());
       } catch(SocketException e) {
            throw(e);
        }
    }

   public void sendToServer(String message) {
       parentApplet.sendToServer(message);
    }
   /* ...*/
  public void processMessage(String receive) {
      StringTokenizer st = new StringTokenizer(receive, "|");
      String command = st.nextToken();
       if("700".equals(command)) {          // 브로드캐스팅 채팅
          // 브로드캐스팅 채팅  700|BROADCAST|playerId|*|chatStr
         String type = st.nextToken();
         String playerId = st.nextToken();
         String target = st.nextToken();
         String chatStr = st.next Token();

          chatArea.append(chatStr+"\n");
      } else if("702".equals(command)) {         // 브로드캐스팅 채팅
      /* ...*/
       }
      /* ...*/
    }
```

```
}
```

4. 3 개월 간의 연재를 마치며..

마소 주니어 독자들과 함께 자바 애플릿 온라인 게임 개발을 위한 기법들에 대해서 알아본지, 벌써 3 개월이란 시간이 흘렀습니다. 나름대로 많은 얘기들을 하고 싶었고, 또 이를 위해 많은 예제들을 보여 주고자 했습니다. 솔직히, 한정된 지면을 통해서 자바 애플릿 온라인 게임 개발과 관련된 많은 기법들을 소개하려다 보니, 소스에 대한 설명이 많이 부족한 것 같다는 느낌이 듭니다. 그래도, 많은 얘기들을 했다는 생각과 또한 많은 소스 코드들을 통해서 보다 실질적인 것들을 보여주었다고 자부합니다. 길면서도 어쩌면 훌쩍 지나간 3 개월을 마무리 하고자 합니다.