App 36. 애플리케이션 시작/종료 상태일 때 알림 받기 :

GoF의 Observer 패턴 적용

```
# 36. 애플리케이션 시작/종료 상태일 때 알림 받기 : GoF의 Observer 패턴 적용

## 학습목표
- GoF의 Observer 설계 패턴을 이해하고 적용할 수 있다.

## 요구사항
- 애플리케이션이 시작되거나 종료될 때 초기화 작업이나 자원 해제 작업을 할 수 있도록 설계를 개선하라.

## 작업
- Observer 객체의 사용 규칙을 정의
  - ApplicationListener 인터페이스 추가
- Observer 객체 구현
  - InitApplicationListener 클래스 추가
- 애플리케이션 환경 정보를 다룰 객체 정의
  - ApplicationContext 클래스 추가
- App 클래스 변경
  - Observer를 등록하고 실행하도록 변경

## 소스 파일

- App.java
- ApplicationListener.java
- InitApplicationListener.java
- ApplicationContext.java
```

[실행 결과]

변경 없음

[Code]

▼ App 36 코드

▼ [App]

```
package bitcamp.myapp;

import bitcamp.context.ApplicationContext;
import bitcamp.listener.ApplicationListener;
import bitcamp.myapp.listener.InitApplicationListener;
import bitcamp.util.Prompt;
import java.util.ArrayList;
import java.util.List;

public class App {

  List<ApplicationListener> listeners = new ArrayList<>();
```

```
ApplicationContext appCtx = new ApplicationContext();

public static void main(String[] args) {
  App app = new App();

  // 애플리케이션이 시작되거나 종료될 때 알림 받을 객체의 연락처를 등록한다.
  app.addApplicationListener(new InitApplicationListener());

  app.execute();
}

private void addApplicationListener(ApplicationListener listener) {
  listeners.add(listener);
}

private void removeApplicationListener(ApplicationListener listener) {
  listeners.remove(listener);
}

void execute() {

// 애플리케이션이 시작될 때 리스너에게 알린다.
for (ApplicationListener listener : listeners) {
  try {
    listener.onStart(appCtx);
  } catch (Exception e) {
    System.out.println("리스너 실행 중 오류 발생!");
  }
}

String appTitle = "[프로젝트 관리 시스템]";
String line = "----------------------------------";

try {
  appCtx.getMainMenu().execute();

} catch (Exception ex) {
  System.out.println("실행 오류!");
  ex.printStackTrace();

} finally {

}

System.out.println("종료합니다.");


    Prompt.close();

    // 애플리케이션이 종료될 때 리스너에게 알린다.
    for (ApplicationListener listener : listeners) {
      try {
        listener.onShutdown(appCtx);
      } catch (Exception e) {
        System.out.println("리스너 실행 중 오류 발생!");
      }
    }
  }
}
```

▼ **[ApplicationListener]**

```
package bitcamp.listener;

import bitcamp.context.ApplicationContext;
```

```
// 애플리케이션의 상태 변경을 알림 받을 객체의 호출 규칙
//
public interface ApplicationListener {

  void onStart(ApplicationContext ctx); // 애플리케이션이 시작될 때 호출됨

  void onShutdown(ApplicationContext ctx); // 애플리케이션이 종료될 때 호출됨
}
```

▼ **[InitApplicationListener]**

```
package bitcamp.myapp.listener;

import bitcamp.context.ApplicationContext;
import bitcamp.listener.ApplicationListener;
import bitcamp.menu.MenuGroup;
import bitcamp.menu.MenuItem;
import bitcamp.myapp.command.HelpCommand;
import bitcamp.myapp.command.HistoryCommand;
import bitcamp.myapp.command.board.BoardAddCommand;
import bitcamp.myapp.command.board.BoardDeleteCommand;
import bitcamp.myapp.command.board.BoardListCommand;
import bitcamp.myapp.command.board.BoardUpdateCommand;
import bitcamp.myapp.command.board.BoardViewCommand;
import bitcamp.myapp.command.project.ProjectAddCommand;
import bitcamp.myapp.command.project.ProjectDeleteCommand;
import bitcamp.myapp.command.project.ProjectListCommand;
import bitcamp.myapp.command.project.ProjectMemberHandler;
import bitcamp.myapp.command.project.ProjectUpdateCommand;
import bitcamp.myapp.command.project.ProjectViewCommand;
import bitcamp.myapp.command.user.UserAddCommand;
import bitcamp.myapp.command.user.UserDeleteCommand;
import bitcamp.myapp.command.user.UserListCommand;
import bitcamp.myapp.command.user.UserUpdateCommand;
import bitcamp.myapp.command.user.UserViewCommand;
import bitcamp.myapp.dao.BoardDao;
import bitcamp.myapp.dao.ListBoardDao;
import bitcamp.myapp.dao.ListProjectDao;
import bitcamp.myapp.dao.ListUserDao;
import bitcamp.myapp.dao.ProjectDao;
import bitcamp.myapp.dao.UserDao;

public class InitApplicationListener implements ApplicationListener {

  UserDao userDao;
  BoardDao boardDao;
  ProjectDao projectDao;

  @Override
  public void onStart(ApplicationContext ctx) {
    userDao = new ListUserDao("data.xlsx");
    boardDao = new ListBoardDao("data.xlsx");
    projectDao = new ListProjectDao("data.xlsx", userDao);

    MenuGroup mainMenu = ctx.getMainMenu();

    MenuGroup userMenu = new MenuGroup("회원");
    userMenu.add(new MenuItem("등록", new UserAddCommand(userDao)));
    userMenu.add(new MenuItem("목록", new UserListCommand(userDao)));
    userMenu.add(new MenuItem("조회", new UserViewCommand(userDao)));
    userMenu.add(new MenuItem("변경", new UserUpdateCommand(userDao)));
    userMenu.add(new MenuItem("삭제", new UserDeleteCommand(userDao)));
    mainMenu.add(userMenu);
```

```
MenuGroup projectMenu = new MenuGroup("프로젝트");
ProjectMemberHandler memberHandler = new ProjectMemberHandler(userDao);
projectMenu.add(
    new MenuItem("등록", new ProjectAddCommand(projectDao, memberHandler)));
projectMenu.add(new MenuItem("목록", new ProjectListCommand(projectDao)));
projectMenu.add(new MenuItem("조회", new ProjectViewCommand(projectDao)));
projectMenu.add(new MenuItem("변경", new ProjectUpdateCommand(projectDao, memberHandler)));
projectMenu.add(new MenuItem("삭제", new ProjectDeleteCommand(projectDao)));
mainMenu.add(projectMenu);

MenuGroup boardMenu = new MenuGroup("게시판");
boardMenu.add(new MenuItem("등록", new BoardAddCommand(boardDao)));
boardMenu.add(new MenuItem("목록", new BoardListCommand(boardDao)));
boardMenu.add(new MenuItem("조회", new BoardViewCommand(boardDao)));
boardMenu.add(new MenuItem("변경", new BoardUpdateCommand(boardDao)));
boardMenu.add(new MenuItem("삭제", new BoardDeleteCommand(boardDao)));
mainMenu.add(boardMenu);

mainMenu.add(new MenuItem("도움말", new HelpCommand()));
  mainMenu.add(new MenuItem("명령내역", new HistoryCommand()));

  mainMenu.setExitMenuTitle("종료");
}

@Override
public void onShutdown(ApplicationContext ctx) {
  try {
    ((ListUserDao) userDao).save();
  } catch (Exception e) {
    System.out.println("회원 데이터 저장 중 오류 발생!");
    e.printStackTrace();
    System.out.println();
  }

  try {
    ((ListBoardDao) boardDao).save();
  } catch (Exception e) {
    System.out.println("게시글 데이터 저장 중 오류 발생!");
    e.printStackTrace();
    System.out.println();
  }

  try {
    ((ListProjectDao) projectDao).save();
    } catch (Exception e) {
      System.out.println("프로젝트 데이터 저장 중 오류 발생!");
      e.printStackTrace();
      System.out.println();
    }
  }
}
```

▼ **[ApplicationContext]**

```
package bitcamp.context;

import bitcamp.menu.MenuGroup;
import java.util.HashMap;
import java.util.Map;

public class ApplicationContext {

  MenuGroup mainMenu = new MenuGroup("메인");
```

```
  Map<String, Object> objContainer = new HashMap<>();

  public MenuGroup getMainMenu() {
    return mainMenu;
  }

  public void addAttribute(String name, Object value) {
    objContainer.put(name, value);
  }

  public Object getAttribute(String name) {
    return objContainer.get(name);
  }


}
```

[과정 탐구]

- Observer 객체의 사용 규칙을 정의

▼ [ApplicationListener] - 규칙 정의

```
package bitcamp.myapp.listener;

// 애플리케이션의 상태 변경을 알림 받을 객체의 호출 규칙

public interface ApplicationListener {

  void onStart();  // 애플리케이션이 시작될 때 호출됨
  void onShutdown();  // 애플리케이션이 종료될 때 호출됨

}
```

▼ [App]

✔

ApplicationListener 객체를 저장할 List 생성

✔

addApplicationListener( ), removeApplicationListener( ) 생성

✔ App의 시작 부분과 종료 부분에서 List의 Listener들을 하나씩 뽑아서 실행시킨다.

```
  List<ApplicationListener> listeners = new ArrayList<>();

  private void addApplicationListener(ApplicationListener listener) {
    listeners.add(listener);
  }

  private void removeApplicationListener(ApplicationListener listener) {
    listeners.remove(listener);
  }

──────────────────────────────────────────────────────────────────────────────


  public App() {

    // 애플리케이션이 시작될 때 리스너에게 알린다.
    for (ApplicationListener listener : listeners) {
```

```
      try {
        listener.onStart();
      } catch (Exception e) {
        System.out.println("리스너 실행 중 오류 발생!");
      }
    }


  void execute() {
    String appTitle = "[프로젝트 관리 시스템]";
    String line = "-----------------------------------";

    try {
      mainMenu.execute();

    } catch (Exception ex) {
      System.out.println("실행 오류!");
      ex.printStackTrace();

    } finally {
      try {
        //((MapUserDao) userDao).save();
        ((ListUserDao) userDao).save();
      } catch (Exception e) {
        System.out.println("회원 데이터 저장 중 오류 발생!");
        e.printStackTrace();
        System.out.println();
      }

      try {
        //((MapBoardDao) boardDao).save();
        ((ListBoardDao) boardDao).save();
      } catch (Exception e) {
        System.out.println("게시글 데이터 저장 중 오류 발생!");
        e.printStackTrace();
        System.out.println();
      }

      try {
        //((MapProjectDao) projectDao).save();
        ((ListProjectDao) projectDao).save();
      } catch (Exception e) {
        System.out.println("프로젝트 데이터 저장 중 오류 발생!");
        e.printStackTrace();
        System.out.println();
      }
    }

    System.out.println("종료합니다.");

    Prompt.close();

    // 애플리케이션이 종료될 때 리스너에게 알린다.
    for (ApplicationListener listener : listeners) {
      try {
        listener.onShutdown();
      } catch (Exception e) {
        System.out.println("리스너 실행 중 오류 발생!");
      }
    }

  }
```

▼ **[App] - 생성자의 이름을 init( )로 변경 및 execute( )의 시작 부분에서 init( )를 호출한다.**

```
package bitcamp.myapp;

public class App {
  MenuGroup mainMenu = new MenuGroup("메인");

  List<ApplicationListener> listeners = new ArrayList<>();

  Map<Integer, Project> projectMap = new HashMap<>();
  List<Integer> projectNoList = new ArrayList<>();

  UserDao userDao;
  BoardDao boardDao;
  ProjectDao projectDao;

  public void init() {

    // 애플리케이션이 시작될 때 리스너에게 알린다.
    for (ApplicationListener listener : listeners) {
      try {
        listener.onStart();
      } catch (Exception e) {
        System.out.println("리스너 실행 중 오류 발생!");
      }
    }

//    userDao = new MapUserDao("data.xlsx");
//    boardDao = new MapBoardDao("data.xlsx");
//    projectDao = new MapProjectDao("data.xlsx", userDao);

    userDao = new ListUserDao("data.xlsx");
    boardDao = new ListBoardDao("data.xlsx");
    projectDao = new ListProjectDao("data.xlsx", userDao);

    MenuGroup userMenu = new MenuGroup("회원");
    userMenu.add(new MenuItem("등록", new UserAddCommand(userDao)));
    userMenu.add(new MenuItem("목록", new UserListCommand(userDao)));
    userMenu.add(new MenuItem("조회", new UserViewCommand(userDao)));
    userMenu.add(new MenuItem("변경", new UserUpdateCommand(userDao)));
    userMenu.add(new MenuItem("삭제", new UserDeleteCommand(userDao)));
    mainMenu.add(userMenu);

    MenuGroup projectMenu = new MenuGroup("프로젝트");
    ProjectMemberHandler memberHandler = new ProjectMemberHandler(userDao);
    projectMenu.add(
        new MenuItem("등록", new ProjectAddCommand(projectDao, memberHandler)));
    projectMenu.add(new MenuItem("목록", new ProjectListCommand(projectDao)));
    projectMenu.add(new MenuItem("조회", new ProjectViewCommand(projectDao)));
    projectMenu.add(new MenuItem("변경", new ProjectUpdateCommand(projectDao, memberHandler)));
    projectMenu.add(new MenuItem("삭제", new ProjectDeleteCommand(projectDao)));
    mainMenu.add(projectMenu);

    MenuGroup boardMenu = new MenuGroup("게시판");
    boardMenu.add(new MenuItem("등록", new BoardAddCommand(boardDao)));
    boardMenu.add(new MenuItem("목록", new BoardListCommand(boardDao)));
    boardMenu.add(new MenuItem("조회", new BoardViewCommand(boardDao)));
    boardMenu.add(new MenuItem("변경", new BoardUpdateCommand(boardDao)));
    boardMenu.add(new MenuItem("삭제", new BoardDeleteCommand(boardDao)));
    mainMenu.add(boardMenu);

    mainMenu.add(new MenuItem("도움말", new HelpCommand()));
    mainMenu.add(new MenuItem("명령내역", new HistoryCommand()));

    mainMenu.setExitMenuTitle("종료");
  }
```

```
private void addApplicationListener(ApplicationListener listener) {
  listeners.add(listener);
}

private void removeApplicationListener(ApplicationListener listener) {
  listeners.remove(listener);
}


public static void main(String[] args) {
  App app = new App();
  app.execute();
}

void execute() {

  init();

  String appTitle = "[프로젝트 관리 시스템]";
  String line = "-----------------------------------";

  try {
    mainMenu.execute();

  } catch (Exception ex) {
    System.out.println("실행 오류!");
    ex.printStackTrace();

  } finally {
    try {
      //((MapUserDao) userDao).save();
      ((ListUserDao) userDao).save();
    } catch (Exception e) {
      System.out.println("회원 데이터 저장 중 오류 발생!");
      e.printStackTrace();
      System.out.println();
    }

    try {
      //((MapBoardDao) boardDao).save();
      ((ListBoardDao) boardDao).save();
    } catch (Exception e) {
      System.out.println("게시글 데이터 저장 중 오류 발생!");
      e.printStackTrace();
      System.out.println();
    }

    try {
      //((MapProjectDao) projectDao).save();
      ((ListProjectDao) projectDao).save();
    } catch (Exception e) {
      System.out.println("프로젝트 데이터 저장 중 오류 발생!");
      e.printStackTrace();
      System.out.println();
    }
  }

  System.out.println("종료합니다.");

  Prompt.close();

  // 애플리케이션이 종료될 때 리스너에게 알린다.
```

```
    for (ApplicationListener listener : listeners) {
      try {
        listener.onShutdown();
      } catch (Exception e) {
        System.out.println("리스너 실행 중 오류 발생!");
      }
    }

  }
}
```

▼ **[HelloApplicationListener] - 생성 (가벼운 테스트)**

```
package bitcamp.myapp.listener;

public class HelloApplicationListener implements ApplicationListener {
  @Override
  public void onStart() {
    System.out.println("안녕하세요!!!");
  }

  @Override
  public void onShutdown() {
    System.out.println("또 만나요!!!");
  }
}
```

▼ **[StyleApplicationListener] - 생성 (가벼운 테스트)**

```
package bitcamp.myapp.listener;

public class StyleApplicationListener implements ApplicationListener {
  @Override
  public void onStart() {
    System.out.println("------------------------------------------------");
  }

  @Override
  public void onShutdown() {
    System.out.println("---------------------^^----------------------");
  }
}
```

▼ **[App] - main 메서드 수정**

```
  public static void main(String[] args) {
    App app = new App();

    // 애플리케이션이 시작되거나 종료될때 알림 받을 객체의 연락처를 등록한다.
    app.addApplicationListener(new HelloApplicationListener());
    app.addApplicationListener(new StyleApplicationListener());

    app.execute();
  }
```

- **Observer 객체 구현**

▼ **[InitApplicationListener] - 구현체 생성 및 메서드 구현 (App의 init( )와 execute( )에서 가져온다)**

```
package bitcamp.myapp.listener;
```

```
public class InitApplicationListener implements ApplicationListener {

  UserDao userDao;
  BoardDao boardDao;
  ProjectDao projectDao;

  @Override
  public void onStart() {

    //    userDao = new MapUserDao("data.xlsx");
    //    boardDao = new MapBoardDao("data.xlsx");
    //    projectDao = new MapProjectDao("data.xlsx", userDao);

    userDao = new ListUserDao("data.xlsx");
    boardDao = new ListBoardDao("data.xlsx");
    projectDao = new ListProjectDao("data.xlsx", userDao);

    MenuGroup mainMenu = new MenuGroup("메인");

    MenuGroup userMenu = new MenuGroup("회원");
    userMenu.add(new MenuItem("등록", new UserAddCommand(userDao)));
    userMenu.add(new MenuItem("목록", new UserListCommand(userDao)));
    userMenu.add(new MenuItem("조회", new UserViewCommand(userDao)));
    userMenu.add(new MenuItem("변경", new UserUpdateCommand(userDao)));
    userMenu.add(new MenuItem("삭제", new UserDeleteCommand(userDao)));
    mainMenu.add(userMenu);

    MenuGroup projectMenu = new MenuGroup("프로젝트");
    ProjectMemberHandler memberHandler = new ProjectMemberHandler(userDao);
    projectMenu.add(
        new MenuItem("등록", new ProjectAddCommand(projectDao, memberHandler)));
    projectMenu.add(new MenuItem("목록", new ProjectListCommand(projectDao)));
    projectMenu.add(new MenuItem("조회", new ProjectViewCommand(projectDao)));
    projectMenu.add(new MenuItem("변경", new ProjectUpdateCommand(projectDao, memberHandler)));
    projectMenu.add(new MenuItem("삭제", new ProjectDeleteCommand(projectDao)));
    mainMenu.add(projectMenu);

    MenuGroup boardMenu = new MenuGroup("게시판");
    boardMenu.add(new MenuItem("등록", new BoardAddCommand(boardDao)));
    boardMenu.add(new MenuItem("목록", new BoardListCommand(boardDao)));
    boardMenu.add(new MenuItem("조회", new BoardViewCommand(boardDao)));
    boardMenu.add(new MenuItem("변경", new BoardUpdateCommand(boardDao)));
    boardMenu.add(new MenuItem("삭제", new BoardDeleteCommand(boardDao)));
    mainMenu.add(boardMenu);

    mainMenu.add(new MenuItem("도움말", new HelpCommand()));
    mainMenu.add(new MenuItem("명령내역", new HistoryCommand()));

    mainMenu.setExitMenuTitle("종료");
  }

  @Override
  public void onShutdown() {
    try {
      //((MapUserDao) userDao).save();
      ((ListUserDao) userDao).save();
    } catch (Exception e) {
      System.out.println("회원 데이터 저장 중 오류 발생!");
      e.printStackTrace();
      System.out.println();
    }

    try {
      //((MapBoardDao) boardDao).save();
```

```
      ((ListBoardDao) boardDao).save();
    } catch (Exception e) {
      System.out.println("게시글 데이터 저장 중 오류 발생!");
      e.printStackTrace();
      System.out.println();
    }

    try {
      //((MapProjectDao) projectDao).save();
      ((ListProjectDao) projectDao).save();
    } catch (Exception e) {
      System.out.println("프로젝트 데이터 저장 중 오류 발생!");
      e.printStackTrace();
      System.out.println();
    }
  }
}
```

- 애플리케이션 환경 정보를 다룰 객체 정의

bitcamp.context, bitcamp.listener 패키지 생성 후 ApplicationListener는 bitcamp.listener로 이동

▼ [ApplicationContext] - 애플리케이션 환경 정보를 다룰 클래스를 생성

✔ 메인 메뉴 및 Map 저장소(objContainer) 생성
✔
getMainMenu( ), addAttribute( ), getAttribute( ) 생성

```
package bitcamp.context;

import bitcamp.menu.MenuGroup;

import java.util.HashMap;
import java.util.Map;

public class ApplicationContext {

  MenuGroup mainMenu = new MenuGroup("메인");

  Map<String, Object> objContainer = new HashMap<>();


  public MenuGroup getMainMenu() {
    return mainMenu;
  }

  public void addAttribute(String name, Object value) {
    objContainer.put(name, value);
  }

  public Object getAttribute(String name) {
    return objContainer.get(name);
  }

}
```

▼ [ApplicationListener] - 수정

```
package bitcamp.listener;

import bitcamp.context.ApplicationContext;
```

```
// 애플리케이션의 상태 변경을 알림 받을 객체의 호출 규칙
//
public interface ApplicationListener {

  void onStart(ApplicationContext ctx);  // 애플리케이션이 시작될 때 호출됨
  void onShutdown(ApplicationContext ctx);  // 애플리케이션이 종료될 때 호출됨

}
```

▼ **[App] - 수정**

```
package bitcamp.myapp;

import bitcamp.context.ApplicationContext;
import bitcamp.listener.ApplicationListener;
import bitcamp.myapp.listener.InitApplicationListener;
import bitcamp.util.Prompt;

import java.util.ArrayList;
import java.util.List;

public class App {

  List<ApplicationListener> listeners = new ArrayList<>();
  ApplicationContext appCtx = new ApplicationContext();


  private void addApplicationListener(ApplicationListener listener) {
    listeners.add(listener);
  }


  private void removeApplicationListener(ApplicationListener listener) {
    listeners.remove(listener);
  }



  public static void main(String[] args) {
    App app = new App();

    // 애플리케이션이 시작되거나 종료될때 알림 받을 객체의 연락처를 등록한다.
    app.addApplicationListener(new InitApplicationListener());

    app.execute();
  }

  void execute() {

    // 애플리케이션이 시작될 때 리스너에게 알린다.
    for (ApplicationListener listener : listeners) {
      try {
        listener.onStart(appCtx);
      } catch (Exception e) {
        System.out.println("리스너 실행 중 오류 발생!");
      }
    }

    String appTitle = "[프로젝트 관리 시스템]";
    String line = "----------------------------------";

    try {
      appCtx.getMainMenu().execute();
```

```
    } catch (Exception ex) {
      System.out.println("실행 오류!");
      ex.printStackTrace();
    }

    System.out.println("종료합니다.");

    Prompt.close();

    // 애플리케이션이 종료될 때 리스너에게 알린다.
    for (ApplicationListener listener : listeners) {
      try {
        listener.onShutdown(appCtx);
      } catch (Exception e) {
        System.out.println("리스너 실행 중 오류 발생!");
      }
    }

  }
}
```

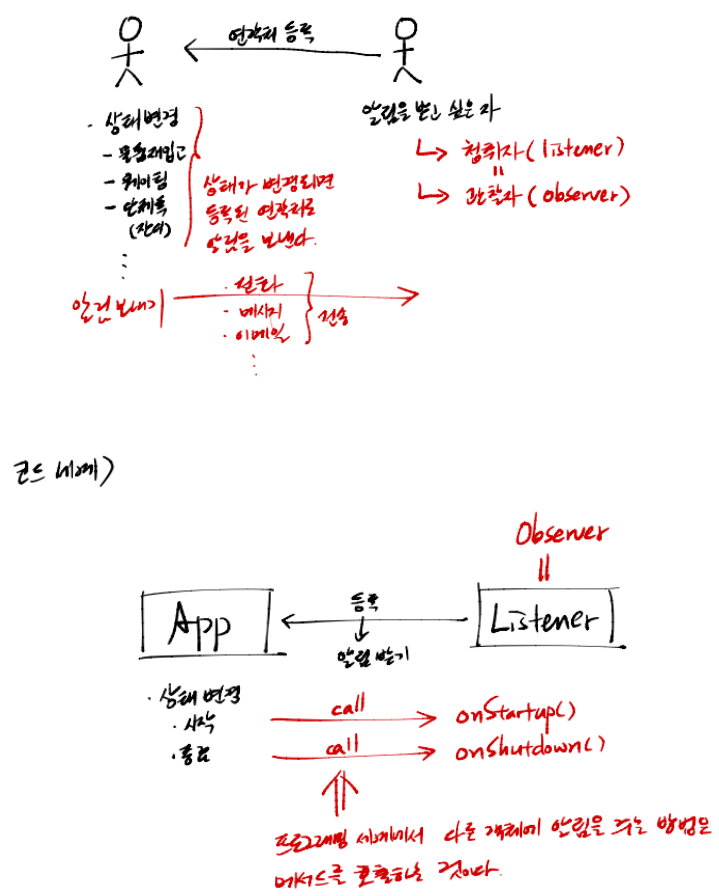
▼ **[InitApplicationListener] - 수정**

```
package bitcamp.myapp.listener;

import bitcamp.context.ApplicationContext;
import bitcamp.listener.ApplicationListener;
import bitcamp.menu.MenuGroup;
import bitcamp.menu.MenuItem;
import bitcamp.myapp.command.HelpCommand;
import bitcamp.myapp.command.HistoryCommand;
import bitcamp.myapp.command.board.*;
import bitcamp.myapp.command.project.*;
import bitcamp.myapp.command.user.*;
import bitcamp.myapp.dao.*;

public class InitApplicationListener implements ApplicationListener {

  UserDao userDao;
  BoardDao boardDao;
  ProjectDao projectDao;


  @Override
public void onStart(ApplicationContext ctx) {

  //    userDao = new MapUserDao("data.xlsx");
  //    boardDao = new MapBoardDao("data.xlsx");
  //    projectDao = new MapProjectDao("data.xlsx", userDao);

  userDao = new ListUserDao("data.xlsx");
  boardDao = new ListBoardDao("data.xlsx");
  projectDao = new ListProjectDao("data.xlsx", userDao);

  MenuGroup mainMenu = ctx.getMainMenu();

  MenuGroup userMenu = new MenuGroup("회원");
  userMenu.add(new MenuItem("등록", new UserAddCommand(userDao)));
  userMenu.add(new MenuItem("목록", new UserListCommand(userDao)));
  userMenu.add(new MenuItem("조회", new UserViewCommand(userDao)));
  userMenu.add(new MenuItem("변경", new UserUpdateCommand(userDao)));
  userMenu.add(new MenuItem("삭제", new UserDeleteCommand(userDao)));
  mainMenu.add(userMenu);
```

```
        MenuGroup projectMenu = new MenuGroup("프로젝트");
        ProjectMemberHandler memberHandler = new ProjectMemberHandler(userDao);
        projectMenu.add(
            new MenuItem("등록", new ProjectAddCommand(projectDao, memberHandler)));
        projectMenu.add(new MenuItem("목록", new ProjectListCommand(projectDao)));
        projectMenu.add(new MenuItem("조회", new ProjectViewCommand(projectDao)));
        projectMenu.add(new MenuItem("변경", new ProjectUpdateCommand(projectDao, memberHandler)));
        projectMenu.add(new MenuItem("삭제", new ProjectDeleteCommand(projectDao)));
        mainMenu.add(projectMenu);

        MenuGroup boardMenu = new MenuGroup("게시판");
        boardMenu.add(new MenuItem("등록", new BoardAddCommand(boardDao)));
        boardMenu.add(new MenuItem("목록", new BoardListCommand(boardDao)));
        boardMenu.add(new MenuItem("조회", new BoardViewCommand(boardDao)));
        boardMenu.add(new MenuItem("변경", new BoardUpdateCommand(boardDao)));
        boardMenu.add(new MenuItem("삭제", new BoardDeleteCommand(boardDao)));
        mainMenu.add(boardMenu);

        mainMenu.add(new MenuItem("도움말", new HelpCommand()));
        mainMenu.add(new MenuItem("명령내역", new HistoryCommand()));

        mainMenu.setExitMenuTitle("종료");
    }

    @Override
    public void onShutdown(ApplicationContext ctx) {
        try {
            //((MapUserDao) userDao).save();
            ((ListUserDao) userDao).save();
        } catch (Exception e) {
            System.out.println("회원 데이터 저장 중 오류 발생!");
            e.printStackTrace();
            System.out.println();
        }

        try {
            //((MapBoardDao) boardDao).save();
            ((ListBoardDao) boardDao).save();
        } catch (Exception e) {
            System.out.println("게시글 데이터 저장 중 오류 발생!");
            e.printStackTrace();
            System.out.println();
        }

        try {
            //((MapProjectDao) projectDao).save();
            ((ListProjectDao) projectDao).save();
        } catch (Exception e) {
            System.out.println("프로젝트 데이터 저장 중 오류 발생!");
            e.printStackTrace();
            System.out.println();
        }
    }
}
```

관련 문법

[GoF] 36. 알림 받기 : GoF의 Observer 패턴



- **실생활 예)**

```
                연락처 등록
사람 A  ◀──────────────  알림받고싶은 자 => 청취자(Listener) = 관찰자(Observer)

상태 변경
-물품 재입고,     상태가 변경되면 등록된 연락처로 알림을 보낸다.
-웨이팅,
-단톡(잔디)
```

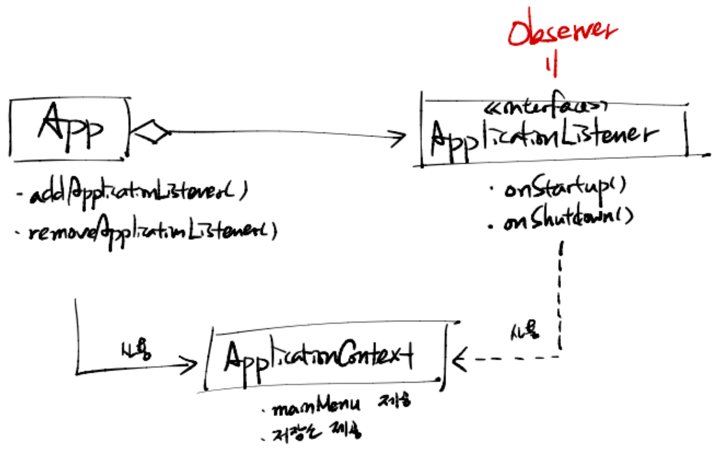
- **코드 세계)**

```
            등록(알림받기)
App  ◇──────────────▶  Listener (= Observer)

상태 변경
                call
- 시작  ──────────────▶  onStartup()
                call
- 종료  ──────────────▶  onShutdown()
```

  - **프로그래밍 세계에서 다른 객체에 알림을 주는 방법은 메서드를 호출하는 것이다.**

[GoF] GoF의 Observer 설계 패턴

* GoF 의 Observer 설계 패턴




```
* addApplicationListener()
* removeApplicationListener()                  Observer
 ┏━━━━━━━━━━━━━━━━━┓                      ┏━━━━━━━━━━━━━━━━━━━━━━━━━━┓
  <<MainClass>>                                  <<interface>>             * onStartup()
      App         ◇──────────────────▶      ApplicataionListener         * onShutdown()
 ┗━━━━━━━━━━━━━━━━━┛                      ┗━━━━━━━━━━━━━━━━━━━━━━━━━━┛
      |                                                   |
      |                                                   |
      |                                                   |
      |                                                   |
      |                 ┏━━━━━━━━━━━━━━━━━━━━━━━━┓          |
      └────────▶         ApplicataionContext      ◀- - - -┘
                        ┗━━━━━━━━━━━━━━━━━━━━━━━━┛
                        * mainMenu 제공
                        * 저장소 제공
```

- App과 ApplicationListener의 관계는 Aggregation(집합)이다.
    - 차와 네비게이션의 관계로 네비게이션은 자동차에 박혀있다. (어떤 객체가 다른 객체를 포함하는 경우)
    - 차와 네비게이션의 라이프사이클이 동일하지 않다.

- ApplicationListener과 ApplicationContext의 관계는 Dependency(의존)이다.
    - 일시적인 관계 ⇒ 인스턴스 필드로 만들지 않는다.
    - 메서드가 호출될 때 일시적으로 그 객체를 쓴다.

- Listener라는 것들은 다 Observer 패턴을 가리킨다.