



ESPNGen

Project Group #35
CSCI5448

Chris Pillion, John Zavidniak, Akshat Kambli

Be Boulder.



University of Colorado **Boulder**

Objectives

- Describe a high level overview of ESPNGen
- Demonstrate the ability to “Signup as a Journalist”
- Demonstrate the ability to “Submit News” as a Journalist
- Demonstrate the ability to “View News” as a User (if time permits)
- Describe significant design choices and patterns that were used in the development of the application

Use Case: Journalist Signup

- Demonstration of ESPNGen
 - Showing the ability to “Signup as a Journalist”

Use Case: Submit News

- Demonstration of ESPNGen
 - Showing the ability to “Submit News” as a Journalist

Use Case: View News

- Demonstration of ESPNGen
 - Showing the ability to “View News” as a User
- Note:
 - This Use Case Demonstration may be skipped if time is running short.

Design: Page and PageAction

- Our application can be thought of as similar website, broken into a number of Pages.
- Each Page has a method to display its content, and actions which can be performed by the user.
- Controller holds current page being viewed by the application user.

Design: How Page Works

Page

```
# PREVIOUS_PAGE_ID:String  
# LOGOUT_ID:String  
# HOME_ID: String  
- pageActions: Map<String, PageAction<?>>  
  
+ Page()  
+ addPageActionStringArr(identifier:String, pageAction: Consumer<String[]>): void  
+ addPageAction(identifier:String, pageAction: Consumer<String>):void  
+ addPageActionCommandConsumer(identifier:String, pageAction:Consumer<String>): void  
+ removePageAction(identifier:String): boolean  
+ containsPageAction(identifier:String): boolean  
+ performLogoutAction(arg:String): void  
+ performAction(identifier:String, arg:String):void  
+ freezeInput(identifier:String, arg:String): boolean  
+ performDefaultAction(identifier:String, arg:String): void  
+ displayPage():void
```

- All viewable pages inherit from Page
- Processes PageActions created in the subclasses
- Uses Strategy Design Pattern to display the page content

Design: How Page Works (cont.)

Controller
<ul style="list-style-type: none">- <u>previousPages</u>: Queue<Page>- <u>currentPage</u>: Page- <u>currentSport</u>: Sport- <u>currentAccount</u>: Account <ul style="list-style-type: none">+ <u>initSessionFactory()</u> : void+ <u>setCurrentAccount(account:Account)</u>:void+ <u>setCurrentPage(page:Page)</u>:void+ <u>returnToPreviousPage()</u>:void+ <u>logout()</u>: void+ <u>goToLobbyPage()</u>:void+ <u>getCurrentAccount()</u>: Account+ <u>getCurrentAccount(clazz: Class<T>)</u>: <T extends Account>+ <u>sendCommandToPage(command:String, arg:Object)</u>:void

Design: PageAction Class

Overcame the problem
of each action
potentially accepting a
different argument type.

```
public class PageAction<T> {  
  
    private final Consumer<T> consumer;  
    private final BiFunction<String, String, T> transformer;  
  
    public PageAction(Consumer<T> consumer, BiFunction<String, String, T> transformer) {  
        this.consumer = consumer;  
        this.transformer = transformer;  
    }  
  
    public void accept(String command, String argument) {  
        consumer.accept(transformer.apply(command, argument));  
    }  
}
```

PageAction

- consumer: Consumer<T>
- transformer: BiFunction<String, String, T>
- + PageAction(consumer:Consumer<T>, transformer:BiFunction<String, String, T>)
- + accept(command:String, argument:String): void

Design: Choosing A Sport

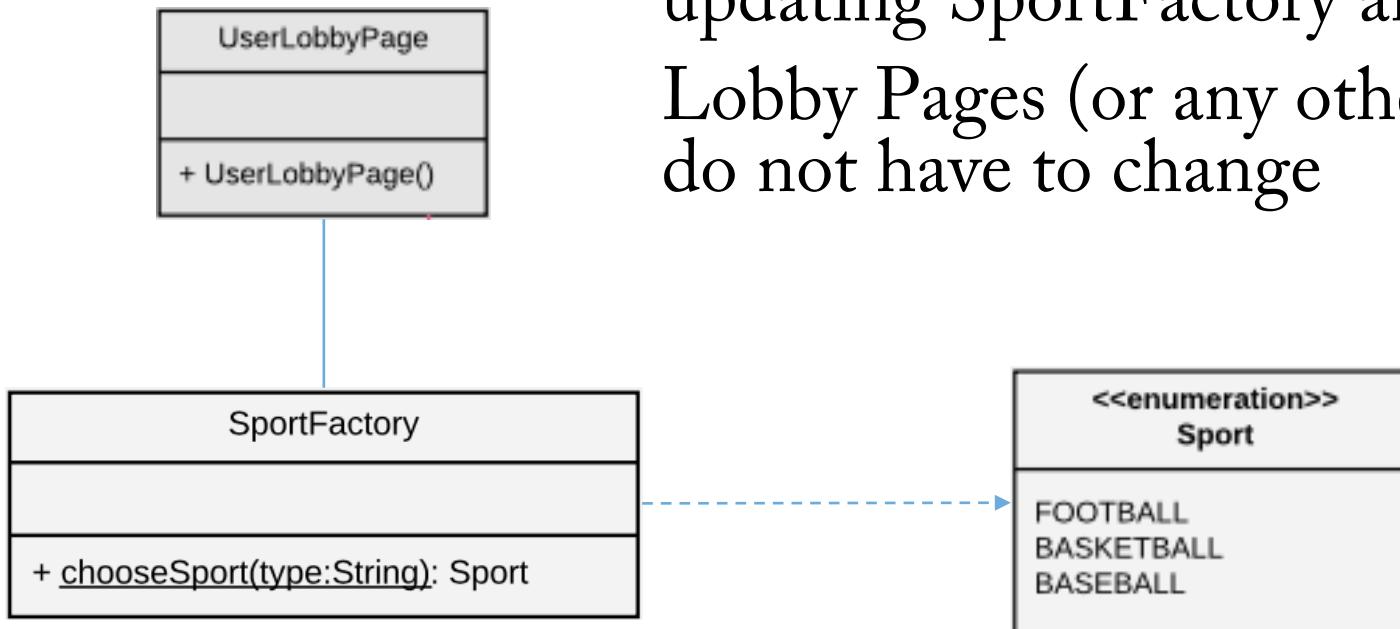
- At its core, our application serves as medium to inform the User about *sports*.
- Not all sports included
- Wanted a way to easily scale and expand sports covered

Design: SportFactory

```
public class SportFactory {  
  
    public static Sport chooseSport(String type) {  
  
        Sport sport = null;  
  
        if (type.equalsIgnoreCase("BASKETBALL"))  
            sport = Sport.BASKETBALL;  
        else if (type.equalsIgnoreCase("BASEBALL"))  
            sport = Sport.BASEBALL;  
        else if (type.equalsIgnoreCase("FOOTBALL"))  
            sport = Sport.FOOTBALL;  
  
        return sport;  
    }  
}
```

- Lets Factory decide which sport to select
- Only Sport Factory and Sport Enum codes will need updating
- Note: Sport stored as an Enum rather than a class, but can easily be extended to a future Sport class type
 - Concept is the same

Design: SportFactory (cont.)



Can add more sports by only updating SportFactory and Sport. Lobby Pages (or any other pages) do not have to change

Conclusion

- What We Learned:
 - Upfront diagramming helps identify potential issues with design and helps adapt the user requirements to actual structure
 - Even when there is an obvious choice for a design pattern, smart decision making still needs to be considered, as its easy to get lost in code once implementation begins
 - Communication is key in a team environment
- Demo Links:
 - https://github.com/cpillion/CSCI5448_Project/ESPNGen_Video_Pillion.mp4
 - https://github.com/cpillion/CSCI5448_Project/ESPNGen_Video_Zavidniak.mp4
 - https://github.com/cpillion/CSCI5448_Project/ESPNGen_Video_Kambli.mp4