# Report for assignment 4 project:

# Public Photography Contest With Live Voting

*Team 9, Innopolis University group BS21-02:*

Zaitseva Sofi, Shulepin Danila, Sokolov Yaroslav, Urzhumov Vladislav

---

Short description:

The task was to implement the Photography Contest system using Java with the application of some software design patterns.

Before working on the code part, our team specified the main classes and methods that should be used in the code. According to deducted conclusions and ideas, we constructed the UML diagram of the future program. During the elaboration of new ideas and features, we discussed it and complemented the diagram. Only after the successful adoption of changes, we added them to the code. The final version of UML diagram and the program itself are attached to the zipped folder. Also, the diagram can be reached via link to the Google Drive.

The program contains three main classes:

- PhotoContest itself (it has the list of participants and the current stage)
- Photographer (photographers are the (possible) participants of the contest. They can register themselves to the contest and send a photo)
- Admin (admin can manage the Contest, change Contest`s state, ban or promote participants)

For convenience and high reusability of code, our team implemented the following patterns:

1. State pattern
   The State pattern in general helps to divide the behavioral features of big classes into parts used in different certain conditions according to the current state of the instance. States are also classes that implement the general State interface. The main purpose of State pattern usage in our program is to change the behavior of the system and objects` status over time (as the Contest proceeds) without any change of class itself. There are two classes that have their states: PhotoContest and Photographer. They change their states almost simultaneously: the state of the Contest affects the state of participants (photographers).

2. Observer pattern

   Observer pattern in general is used for the "one publisher – a lot of subscribers" situation. Pattern lets the subscribers get the appropriate and necessary notifications when the important publisher`s event is present, The implementation of it consists of the additional interface (usually called the "Subscriber") with the simple method notification().
   Observer is used to provide the possibility of smart-notification-mailing (notifications should be sent to the participants). That is, on each particular state of Contest, notifications should be sent only to participants which are changing their State (for example, Registration to Failure) at the moment.

---

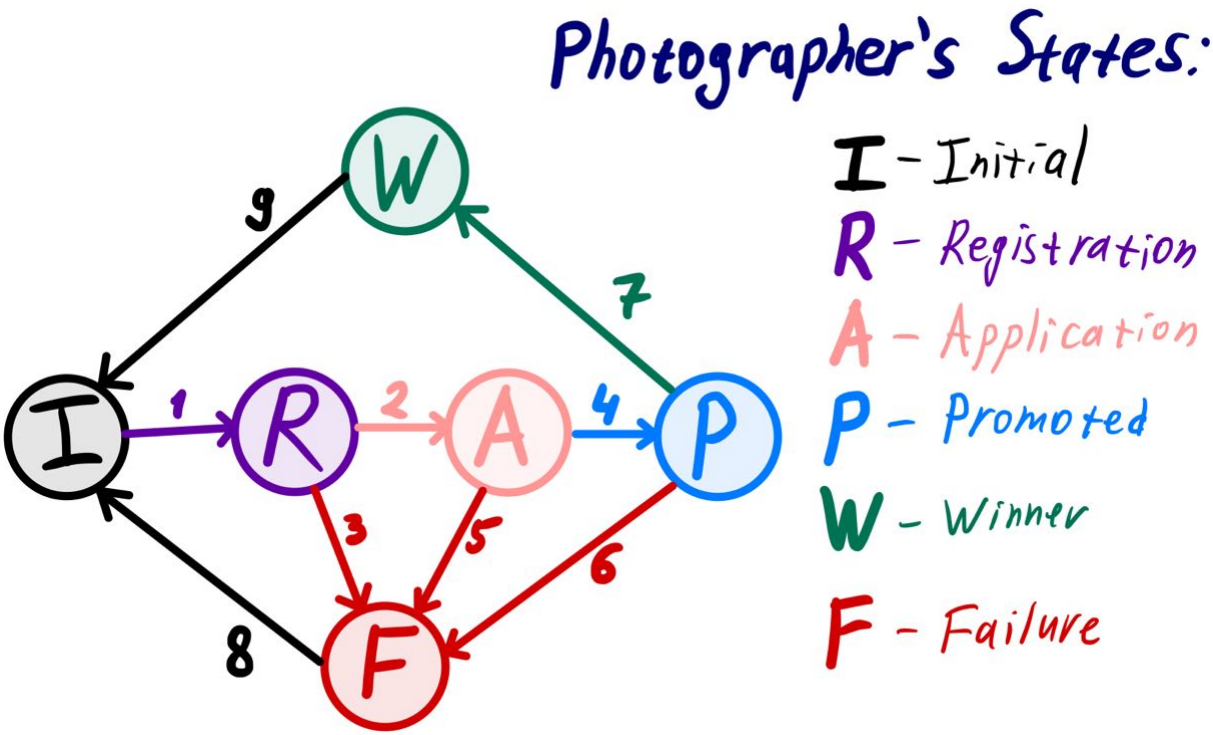
More detailed description:

The Photography Contest is managed by the Admin. Some of the Photographer class instances are the participants of the Contest. To enter the Contest, photographer should enter his name, contact information (phone number for SMS notifications, or e-mail address for notifications via e-mail), register to the Contest, and send a photo (registered participants without any photo will be failed at the beginning of second Contest stage).

The states of Photographer determine participants` current status (progress) in the Contest. Due to their status, system decides what notification to send each of them (if any notification needed). Photographer can have one of six following states:

[1] Initial (photographer has entered his/her name and contact information, but he/she is not a participant of any contest)
[2] Registration (photographer has registered himself/herself to the current Contest)
[3] Application (photographers` photo was applied)
[4] Promoted (photographer`s photo was promoted to the last (voting) stage of the Contest)
[5] Winner (photographer is the winner of the contest)
[6] Failure (photographer has failed the current contest and cannot be the winner)

In total, Photographer`s states and transitions between them can be synthesized into the following diagram:




Moreover, transitions between states should be specified (the digit corresponds to the number of transition on the diagram above, the reason of making such transition is written after the colon).

1. Initial -> Registration: Photographer have registered to the Contest.
2. Registration -> Application: Submitted photo to the Contest.
3. Registration -> Failure: Photographer did not applied a photo before the start of second contest stage.
4. Application -> Promoted: The application was promoted to the Voting stage.
5. Application -> Failure: The applied photo is plagiarized.
6. Promoted -> Failure: The photo did not get enough votes to win the Contest.
7. Promoted -> Winner: Photographer won the Contest.
8. Failure -> Initial: Contest was closed; all Photographers are in Initial state now.
9. Winner -> Initial: Contest was closed; all Photographers are in Initial state now.

---

The states of the Contest are the following:

[1] Application State (Corresponds to the first stage of the Contest. Photographers are able to register themselves to the Contest and send the photos to participate)
[2] Choice State (Corresponds to the second stage of the Contest. Registration of the new participants and Application of new photos are not allowed; The Admin checks the applications to promote those, whose photos are not absent, plagiarized or inconsistent)
[3] Voting State (The last interactive state of the Contest. When all the worthy applications were chosen by the Admin, Contest proceed to the Voting State,

and the audience can now vote for the photos they like. In our model, the voting state is simplified to the assignment/specification of vote number (rating) for each publication of Promoted Photographers.

[4] Awarding State (Photographers with the biggest number of votes are awarded (their state becomes the Winner state), others fail the Contest)

[5] Closed State (After the awards are distributed and winners are announced, the Admin closes the Contest)

The schematic representation of Contest States is the following:


The Observer pattern was implemented because the Photographers should be notified about any changes in their status. The Photographer instances are considered as "subscribers" and implement the Subscriber interface. The Contest is considered as the "publisher", which has interesting events to notify about.

---

By virtue of programming patterns usage, the code can be easily modified and expanded. As an option of future improvements, contest types specification can be implemented, so interested photographers could get notifications about new contests with the topics they are interested of. Another example of the future improvement is the two-step voting procedure (the main voting and the final voting with the best ones from the main voting part), and it can be implemented by adding only one new Photographer state and one Contest state.

Thank you for the attention and for the interesting task provided.