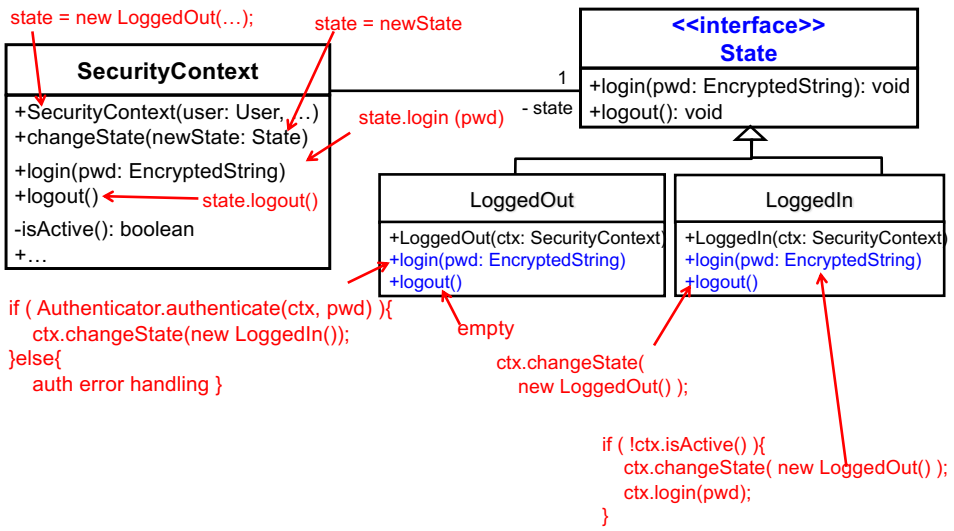
HW 5: Implement this:



- Implement the login example with *State*.
  - No need to seriously implement an authentication mechanism; i.e., `Authenticator`, `User` and `EncryptedString`.
    - You can keep them as simple as possible.
    - e.g., `Authenticator.authenticate()` always return true without doing actual authentication.
    - e.g., `User` and `EncryptedString` have no data fields and no methods.
  - No need to seriously implement `isActive()`.
    - e.g., Return true in the first login, and return false afterword.
- Feel free to add any new methods that you feel are necessary.
  - e.g., `getState()` in `SecurityContext`
- Feel free to modify any existing methods whenever appropriate.
  - e.g., `logout(): State` and `login(...): State`

- Do unit testing to verify all state transitions.
  - A simple strategy:

```
SecurityContext ctx = new SecurityContext(...);
assertTrue(ctx.getState() instanceof LoggedOut);

ctx.login(...);
assertTrue(ctx.getState() instanceof LoggedIn);
```

  - Alternatively, add `equals()` into `LoggedOut` and `LoggedIn` to perform equality check (i.e. state-to-state comparison).
    - Then, use `assertEquals()`.
- Deadline: Nov 6 midnight

Optional Requirements

- Implement `LoggedOut` and `LoggedIn` as *singleton* class.
- Implement `isActive()` in a more reasonable (or serious) way.
  - Keep login timestamps in `SecurityContext` (e.g., `LinkedList<LocalDateTime>` as a data field)
  - Judge if a user is actively logged in or not by checking the last login timestamp.
- You will get extra points if you work on an optional requirement(s).

Template Method Design Pattern

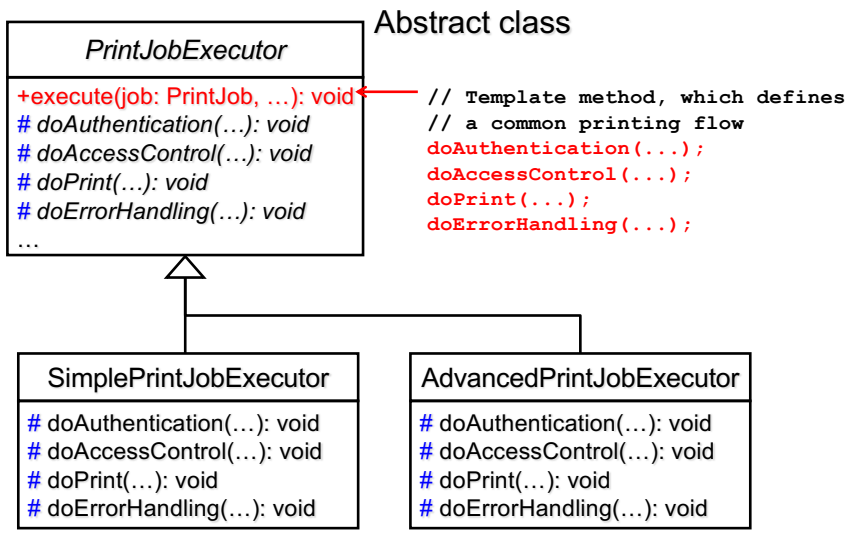
Template Method

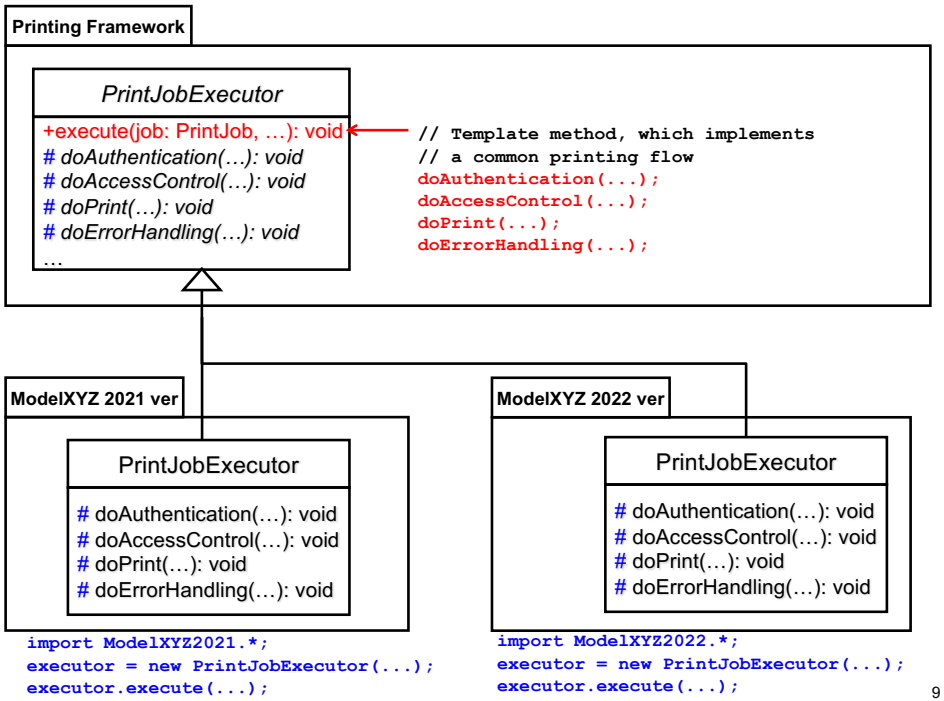
- Intent
  - Define the template (or abstract flow) of an algorithm in a superclass's method
    - Defer the implementation details of some steps in the algorithm to subclasses.
    - Have subclasses override (or redefine) those steps without changing the algorithm's template/flow.
  - Can reuse (or enforce) the template in all subclasses.

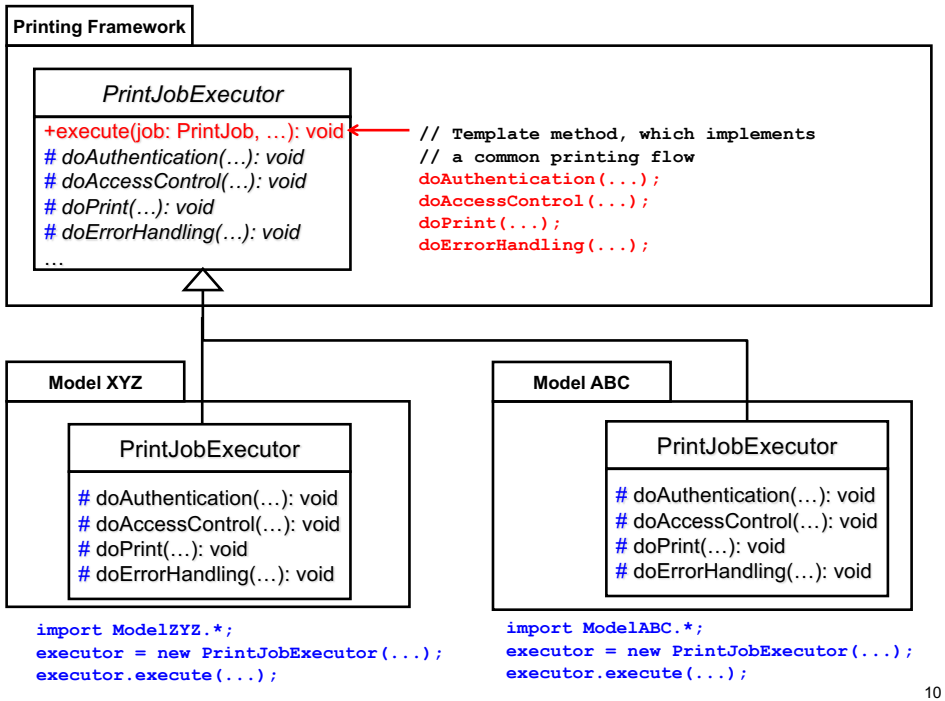
Template for Printing Procedure

- Suppose you are implementing printer firmware
  - Common printing flow
    - Authentication
      - Enforce who can access the printer and who cannot.
    - Access control
      - Enforce who can use which services.
    - Print
    - Error handling
      - Manage what to do upon errors
  - Can define this common flow in a superclass and reuse the flow across its subclasses, which implement different firmware.
    - Different versions for the same printer model.
    - Different firmware for different printer models.

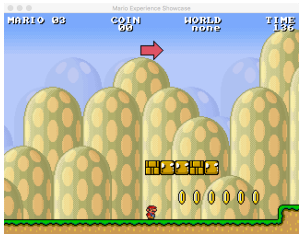
Template Method Applied

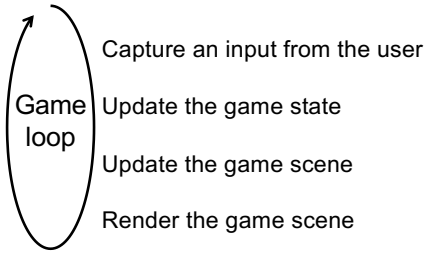


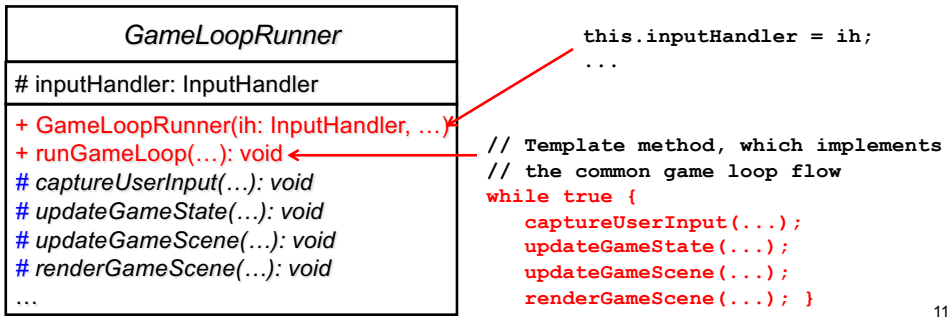


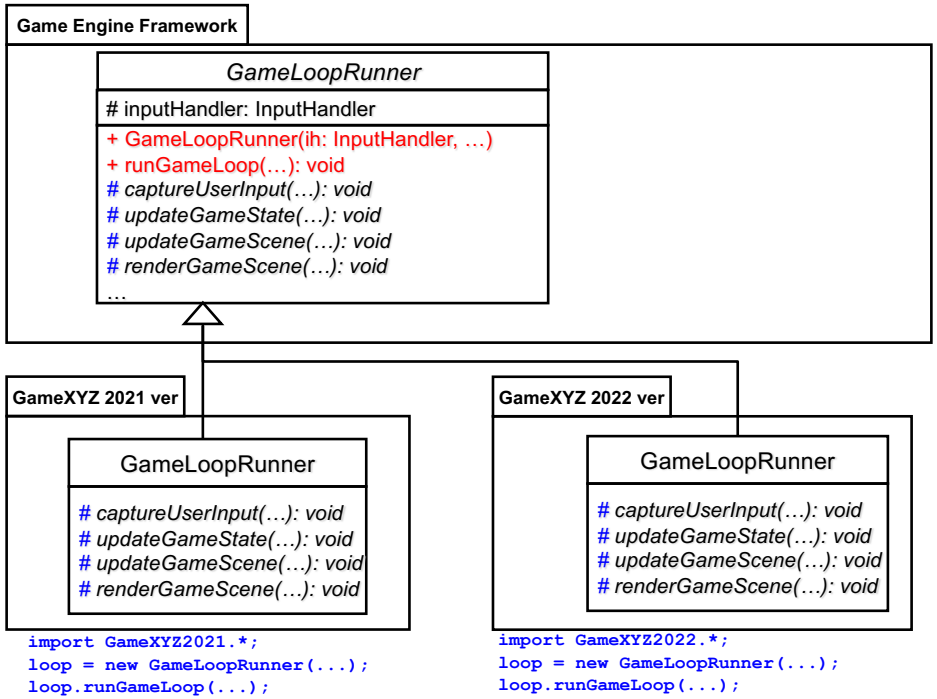


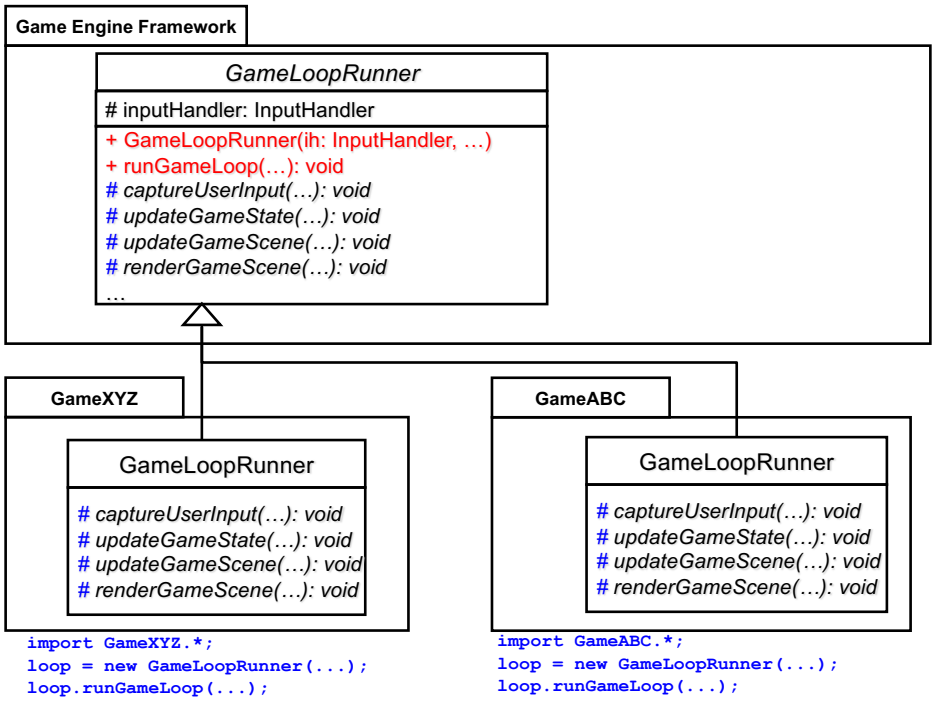
Game Loop as a Template












```
import GameXYZ.*;
loop = new GameLoopRunner(...);
loop.runGameLoop(...);
```

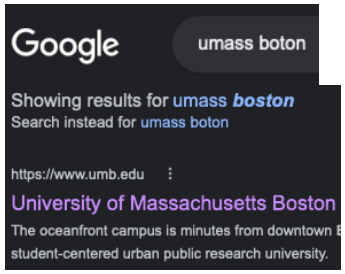
```
import GameABC.*;
loop = new GameLoopRunner(...);
loop.runGameLoop(...);
```

Benefits

- Can define the flow of an algorithm explicitly.
  - Can separate individual steps in the flow clearly.
- Can use the same flow consistently across different subclasses.

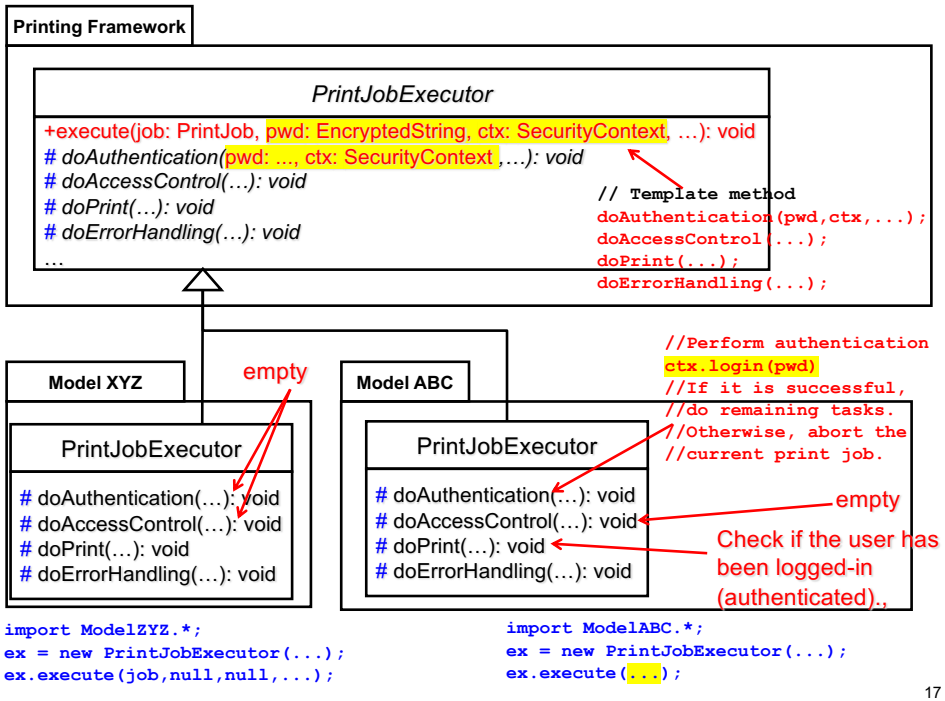
Other Template Examples

- Loan initiation (loan setup)
  - Credit score analysis
  - Transaction history analysis
  - Household analysis
  - etc. etc.
- Keyword search
  - Keyword analysis
  - Search
  - Search result ordering



HW 6

- Combine the printer example and login example
  - Implement two printer models.
  - One of the two models requires each user to log in (i.e., go through authentication)
    - Only logged-in (authenticated) users can run print jobs.
  - The other model skips/ignores authentication.
    - All users can run print jobs.



- No need to do access control.
  - `doAccessControl()` can be empty.
- If a login (authentication) fails, abort the current print job and let the caller of `execute()` know that.
  - e.g., Have `doAuthentication()` throw an exception and have `execute()` re-throw it.
- Deadline: Nov 6 midnight
  - This is the last HW in the first half of HW assignments.
  - HW 1 to 6 will be due at Nov 6 midnight.

Important Notice

- You must work on your coding alone (by yourself).
  - You can discuss HW assignments with other students. However, you must do your coding yourself.
- It is an academic crime to
  - Copy (or steal) someone else's code and submit it as your own work.
  - Allow someone else to copy (or steal) your code and submit it as his/her work.
    - Use a private repo to avoid this.
- You will end up with a serious situation if you commit this academic crime.
  - The University, College, Department and I have no mercy about it.