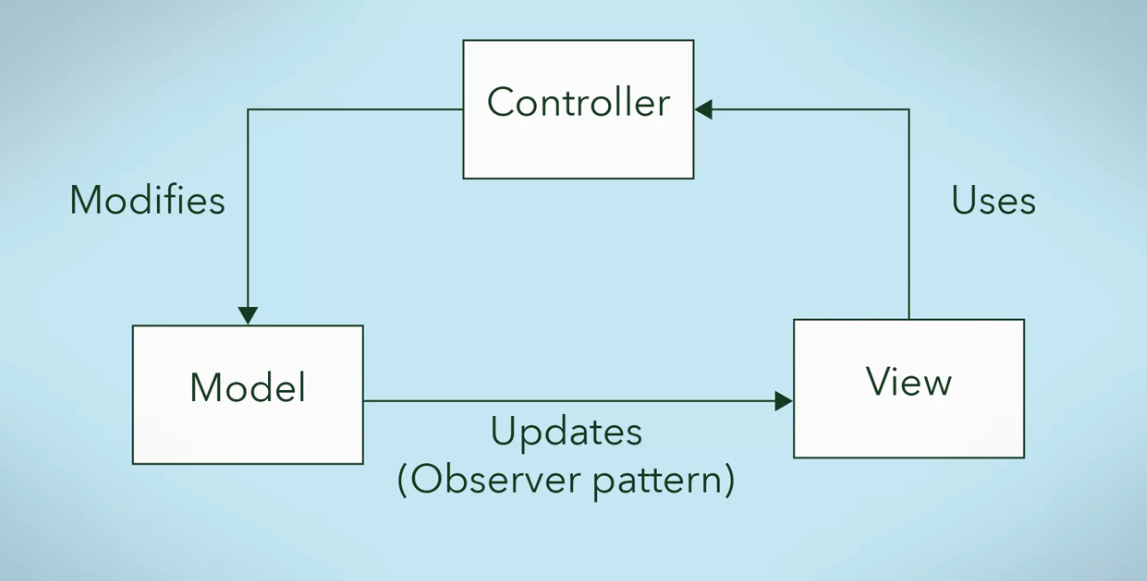
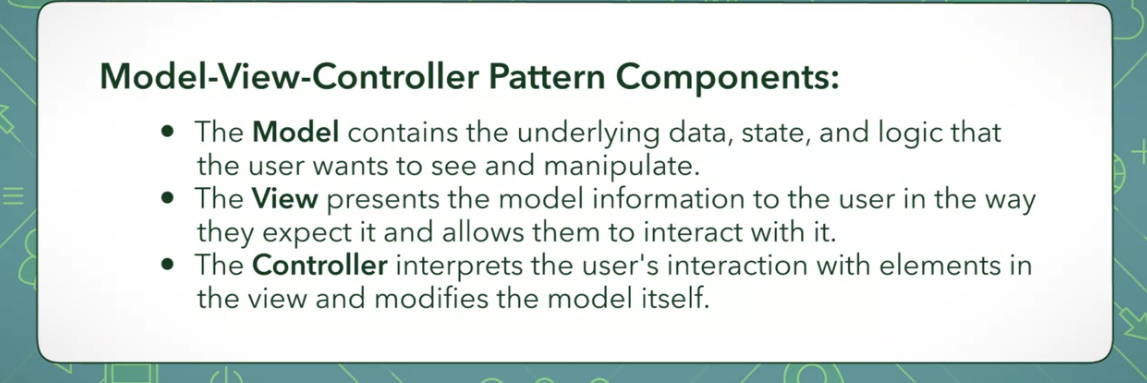
* Sa zicem ca la un magazin, vanzatorul inca foloseste calculatorul de mana pentru a calcula pretul la cumparaturile clientilor. Vanzatorul trebuie sa stie toate preturile la produse si sa le calculeze pe fiecare/
* Clientii, si vanzatorul, s-au saturat de asta si vor deja o User Interface, adica vanzatorul sa aiba un scaner, si pe ecran sa apara deodata preturile la produse si produsele
* De fiecare data cand auzim de **User Interface**, mereu ne referim la MVC Pattern
* **MVC** – Model, View, Controller. El prevede ca responsabilitatile unui system care ofera o User Interface sunt divizate la aceste 3 componente
* Cam asa arata el:



**Model(Backend)** – contine datele si logica pe care userul le manipuleaza si vrea sa le vada. Model contine propriile metode si fields de care are nevoie.Cand anumite date se modifica in back end, model tot se va modifica si deci pe baza lui, view tot se va modifica, ca sa arate noile schimbari deja. Model are propriile metode si fields de care are nevoie pentru a exista.Model nu depinde de Controller si exista independent. Models sunt **Entity Objects**

**View(Frontend)** – ofera la user posibilitatea de a vedea Model, sau cel putin parti din el. Asta e o pagina WEB sau screen ce arata preturile si produsele la clienti. Modificarea lui Model duce si la modificarea lui View. Anume de asta si se foloseste Observer Design Pattern. Daca model isi modifica statusul, el are grija si views sa se modifice, adica sa le anunte.Deci, view este un observer, si model un Subject. Este un **Boundary Object**

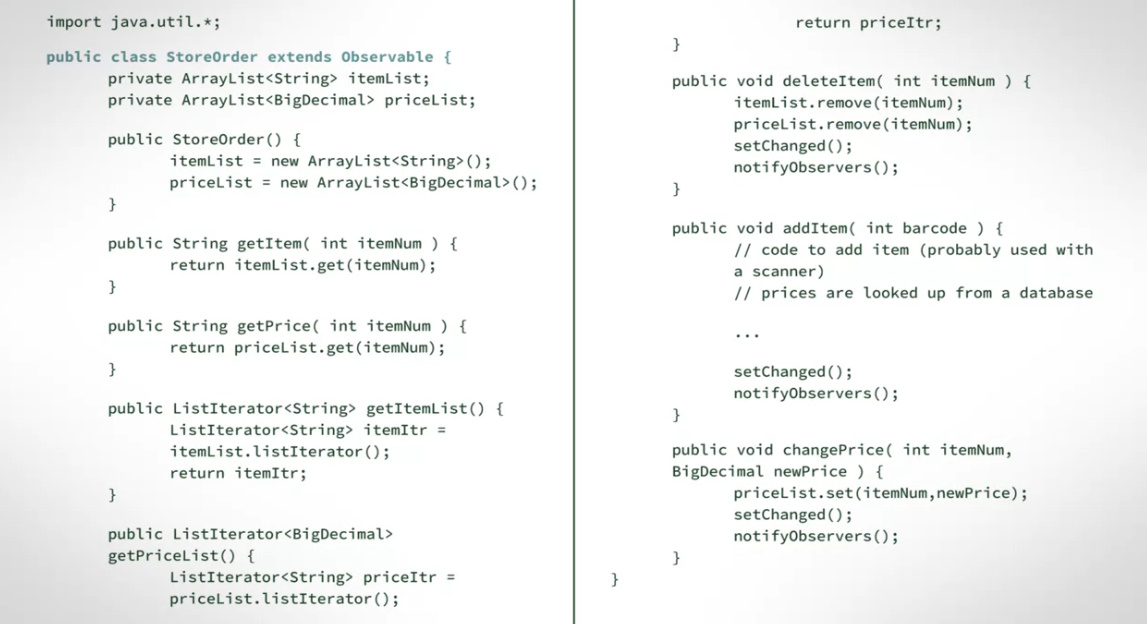
**Controller** – view poate trimite date, gen prin butoane, fielduri etc. si aceste date se duc la Controller. View nu comunica direct cu Model. Controller isi ia responsabilitatea de a lucra cu datele introduse de user si sa trimita date la view, si asa View se focuseaza doar pe user interface.Este un **Control Object**

****

* MVC foloseste Separation of Concers
* MVC nu ofera o implementare concreta. El doar prevede ca separam backend, frontend si cream o comunicare intre ele, si gata, putem sa il mai modificam.

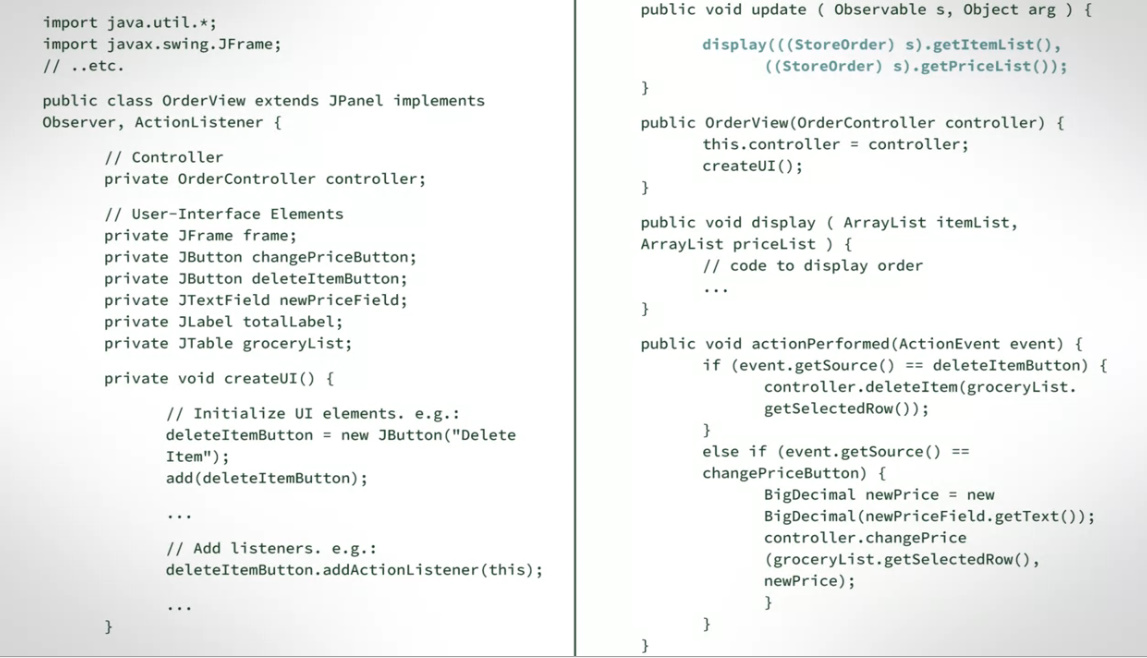
**Exemplu**

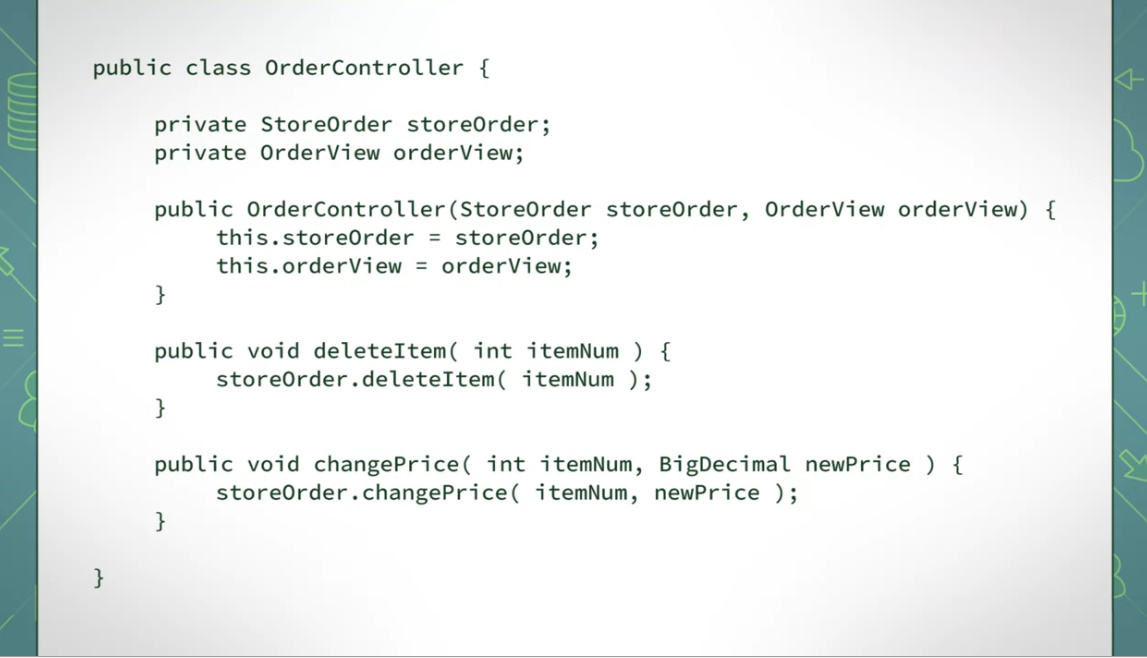
Cream un Model.



Java ofera o clasa Observable, ce permite sa cream Model ce va avea observers

Cream View





Un Sistem va avea mai multe Models, Controllers si Views

MVC poate avea multe implementari.