## Factory Method

#### Factory method koristim:

* Za kreiranje raznih akcija na uređajima

Kako je potrebno kreiranje različitih akcija i to je potrebno da bude odvojeno zbog toga mi je dovoljan factory method jer mi je potrebna samo jedna tvornica. Da ih je bilo potrebno više koristio bih abstract factory.

## Builder

Builder se i dalje koristi kao i u prethodnim zadaćama. Samo je ovdje uklonjena metoda opremi i sada je isti kao u prvoj zadaći. Njegov slijed akcija je:

* učitavanje postavki,
* učitavanje podataka,
* inicijalizacije

Zbog toga sam napravio builder jer se vrlo jednostavno proširi da npr. čita postavke iz baze sustav. Dakle ovo je napravljeno zbog proširivosti i zbog lakšeg testiranja (napravi se builder koji ima mock-ane vrijednosti pa se koristi za testove).

## MVC

Poanta korištenja MVC uzorka dizajna je odvojiti aplikacijski model od prikaza. Ovdje imamo prikaz VT100 Terminala, ali nam je zbog tog uzorka vrlo jednostavno dodati novi uzorak gdje samo trebamo zamijeniti pogled. Bilo da se radi o html prikazu, desktop aplikaciji ili drugom terminalu.

## Observer

Observer se koristi kao dio MVC-a gdje nakon ažuriranja modela, Observer obavještava Pogled koji onda odlučuje što će s tom informacijom.

## Singleton

Najjednostavniji uzorak dizajna. Često se koristi. Ja ovdje koristim kod factory method (dovoljna mi je samo jedna instanca klase koja implementira ovu metodu) i kod aplikacijskog pomagača koji drži sve metode i podatke koji trebaju biti globalno dostupni u aplikaciji. Naravno, sve treba biti posloženo da je sigurno za višedretveni pristup instanci.

## Chain of responsibility

To je bio jedan od uvjeta zadaće. Chain of Responsibility nije učestali uzorak dizajna no koristi se za ulančanu provjeru ovisno u koji rang odgovornosti zahtjev spada. U ovoj implementaciji zadaće je dodana funkcionalost koja za traženi aktuator pregledava u kojoj je zoni vrijednost (dobro, oprez i opasnost) i da li se spušta ili raste kako bi se znalo što se događa s aktuatorom.

# Memento

Jedan od implicitnih uvjeta zadaće. Vraćanje i spremanje stanja bez korištenja datoteke ili slične pohrane.

# Prototype

Koristi se kako bi memento klonirao vrijednost umjesto da drži isti objekt zbog sprečavanja neželjenih promjena.

### Uzorci koji su ostali isti:

* **Factory Method** - samo za implementaciju Uređaja je ostala ista.

### Uzorci koji su uklonjeni:

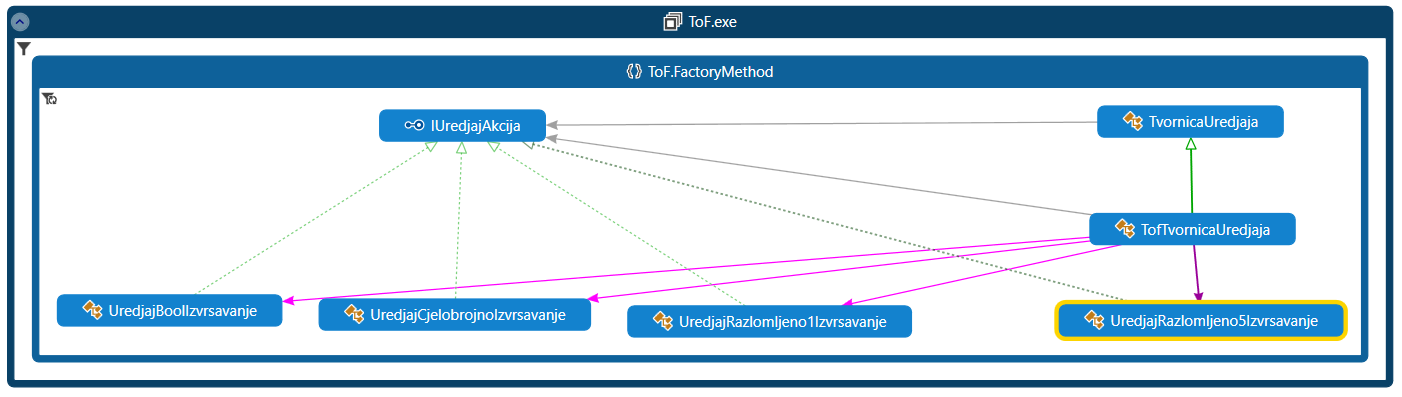
* **Factory Method** - samo za implementaciju Testera je uklonjena jer nije više potrebna.
* **Decorator** – nema više spremanja na način kao u prethodnoj zadaći
* **Composite** – informacije se sada prikazuju na drugačiji način
* **Visitor -**  akcije nad uređajima se sada rade na drugačiji način

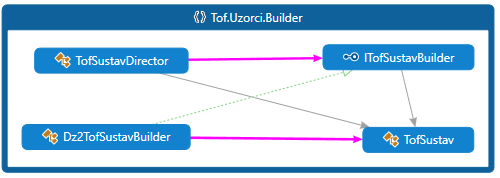
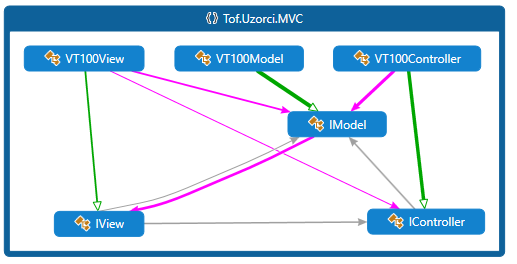
### Uzorci koji su promijenjeni:

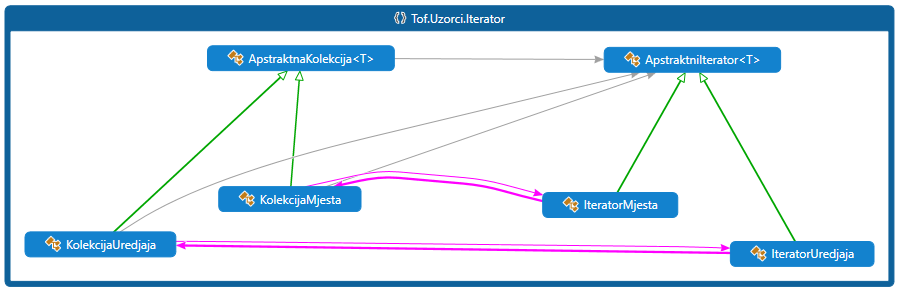
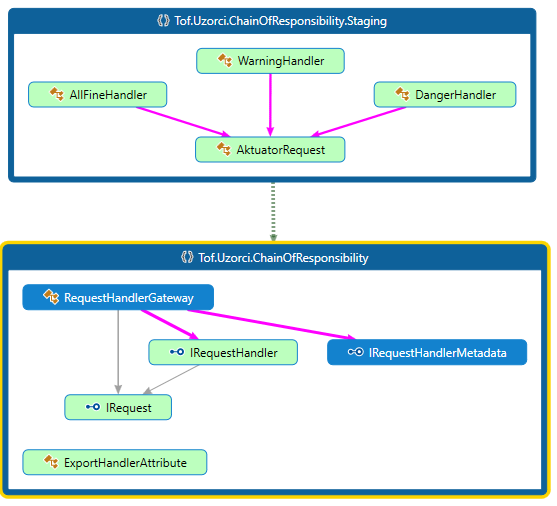
* **Builder** – maknuta metoda u slijed izgradnje (Opremi) koja je bila dodana u prošloj zadaći. Sada je builder isti kao što je i bio u prvoj zadaći (tj. ima iste metode) a razlikuju se u implementaciji.
* **Singleton** – Male preinake. Čišćenje aplikacijskog pomagača i micanja viška koda

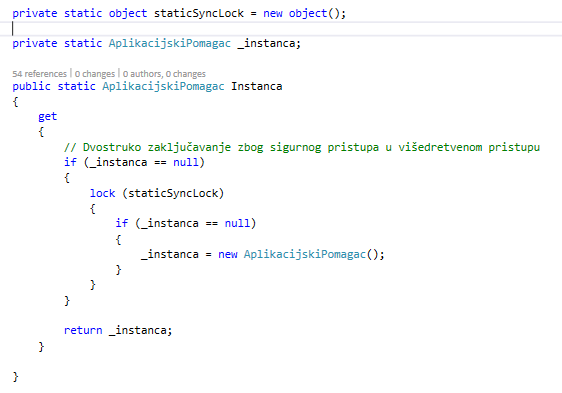
### Uzorci koji su dodani:

* **MVC** – Koristi se za komunikaciju između rezultata operacija, podataka i korisnika.
* **Observer** – dio mvc-a. Gdje nakon promjena Model obavještava View da je došlo do promjene gdje View onda prikaže promjene
* **Chain od Responsibility** – koristi se za vlastitu funkcionalnost. Vlastita funkcionalnost pregledava u kojoj je zoni vrijednosti aktuator (sve je ok, oprez i opasnost). Svaki handler je odgovoran za svoj raspon. Napravljena je i generička implementacija Chain of Responsibility uzorka.
* **Memento** – koristi se za spremanje stanja tof sustava i vraćanje kod naredbi za isto (VP za vraćanje podataka i SP za spremanje podataka)
* **Prototype** – koristi se za kloniranje tof sustava kod kreiranja mementa





Singleton