## Factory Method

#### Factory method koristim u dva navrata:

* Za kreiranje testera uređaja i
* Za kreiranje raznih akcija na uređajima

Kako je potrebno kreiranje različitih testera i 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 je ostao kao i u prethodnoj zadaći samo se promijenila logika pa su se tako i promijenile implementacije metoda i dodan je još jedan korak (Opremi) uz korake koji su već bili:

* učitavanje postavki,
* učitavanje podataka,
* inicijalizacije,
* opremanje

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).

## Decorator

Decorator je implemetiran za bufferirano spremanje linija. Kako već postoji cijelo stablo koje je konstruirano pomoću decoratora (streamovi) samo što nema implementacije sa linijama već brojem bajtova kao veličinom ja sam to iskoristio i dodao još jednu konkretnu implementaciju konkretnog dekoratora koji zapisuje na način kao što je opisano u zadaći.

## Iterator

Iterator se koristi za iteriranje po kolekciji mjesta kako je opisano u zadaći te dodatno za iteriranje nad kolekcijom uređaja.

## 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.

## Composite

Composite koristim za prikaz informacija o uređajima i njihovim povezanim uređajima kao što je opisano u zadaći.

## Visitor

Visitor koristim ukoliko dođe do promjene na senzoru da se prikače svi aktuatori koji su povezani na senzor koji je promijenio stanje (ali samo jednom se dodaje) da bi prvo izvršio radnju a onda ispisao vrijednost.

### Uzorci koji su ostali isti:

* **Factory Method** - obje implementacije (Uredjaji i Testeri). Ovdje nije promijenjena nijedna linija koda.

### Uzorci koji su uklonjeni:

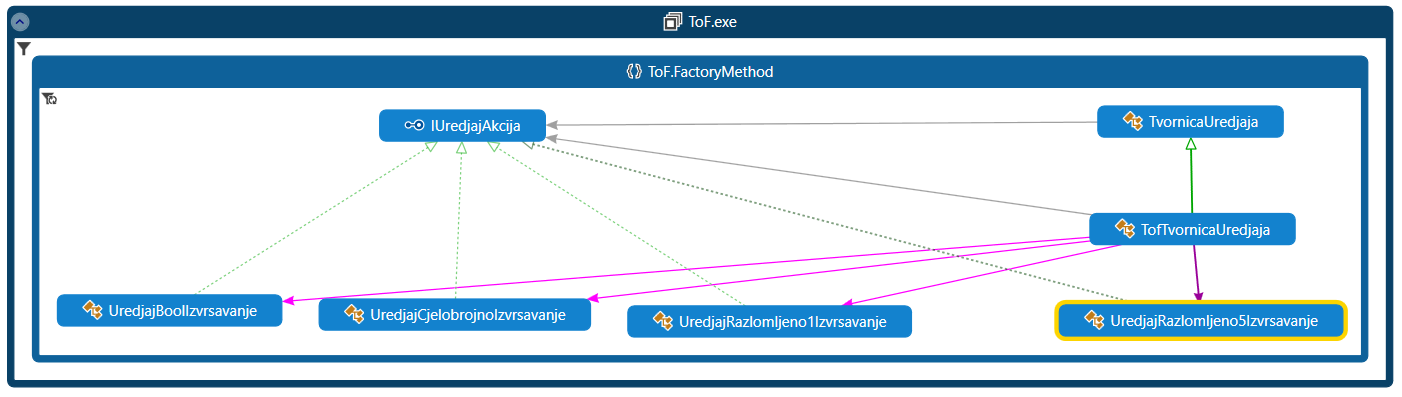
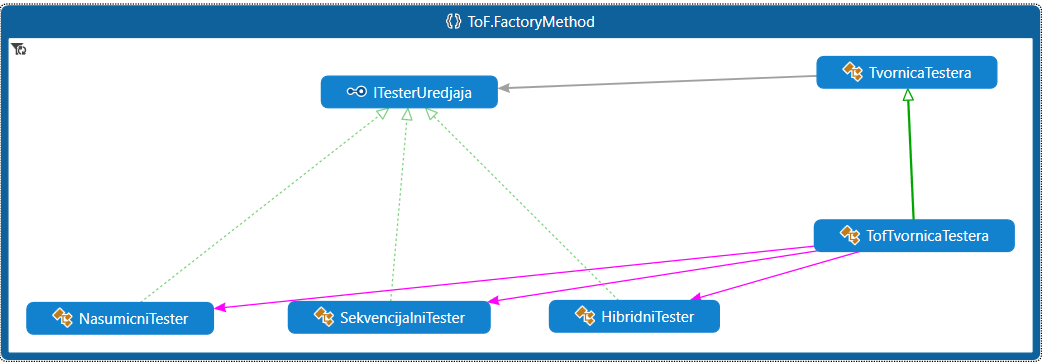
* **Prototype** – ionako je bio tu isključivo radi testiranja.
* **Adapter** – izbačen je jer zbog promjene načina rada nije više adekvatan za upotrebu u potrebnom slučaju (različite radnje aktuatora i senzora).

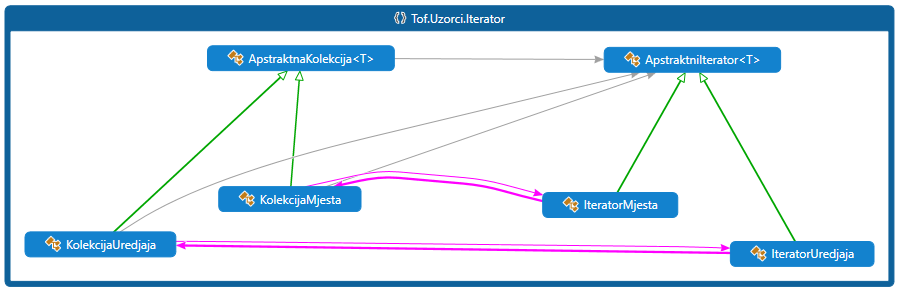
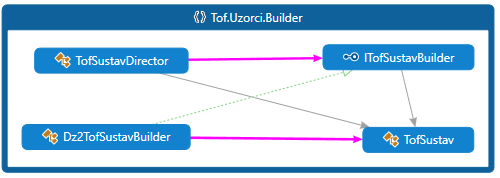
### Uzorci koji su prošireni:

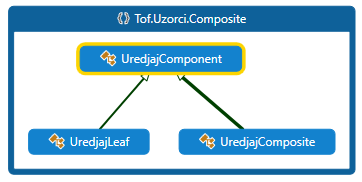
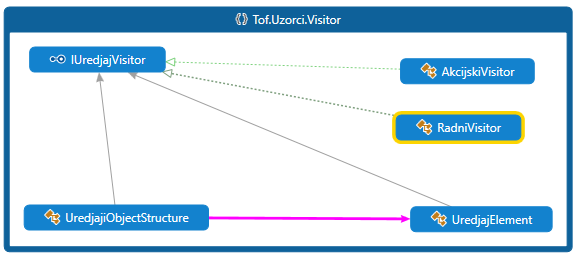
* **Builder** – dodana je još jedna metoda u slijed izgradnje (Opremi) i osim toga logika koja se promijenila u zadatku je ovdje implementirana kao druga konkretna implementacija buildera.

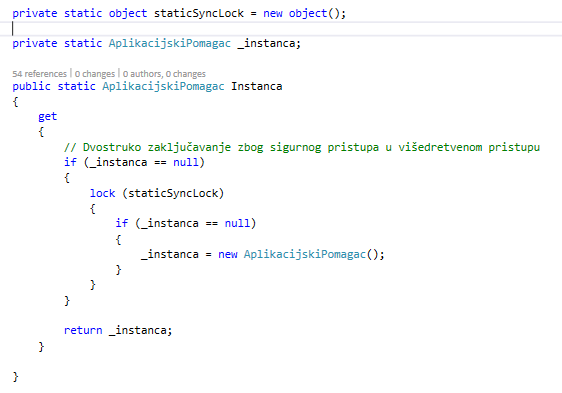
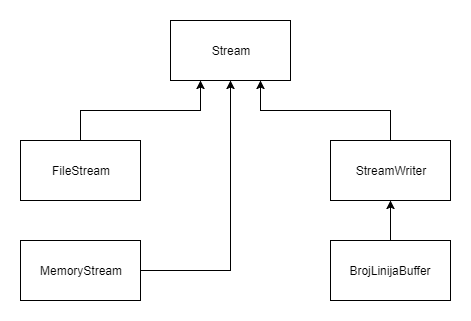
### Uzorci koji su dodani:

* **Decorator** – dodan je za pohranu loga pomoću međuspremnika na bazi broja linija sadržaja, a ne broja bajtova za što već postoji implementacija.
* **Iterator** – koristi se za iteriranje po kolekciji mjesta kako je opisano u zadaći te dodatno za iteriranje nad kolekcijom uređaja.
* **Composite** – koristi se za prikaz informacija o uređajima i njihovim povezanim uređajima kao što je opisano u zadaći
* **Visitor -** koristi seda ukoliko dođe do promjene na senzoru da se prikače svi aktuatori koji su povezani na taj senzor koji je promijenio stanje (ali samo jednom se dodaje) da bi prvo izvršio radnju a onda ispisao vrijednost.









Decorator Singleton