# Kreacijski paterni

## Singleton

Singleton pattern bi se mogao iskoristiti ako bi recimo implementirali dobijanje informacija o proizvodu sa nekog drugog web-servisa. U tom slučaju bi klasa koja vrši zahtjev drugom servisu mogla biti singleton da se ne bi desilo da se kreira prevelik broj objekata i pošalje previše requestova.

## Builder

Builder pattern **ćemo** iskoristiti sa klasom *Filter*. Ova klasa ima mnoge atribute poput načina sortiranja, price range, specifični zahtjevi za određenim modelima i sl. Da bi olakšali kreiranje ovih objekata te da ne bi imali nepotrebne inicijalizacije (jer neki atributi neće biti iskorišteni npr. korisnik želi sve Intel proizvode ali nije ga briga za price-range) koristimo builder pattern.

## Prototype

Prototype pattern **ćemo** iskoristiti nad klasom *Product*. Razlog za ovo je što je u mnogim metodama potrebno praviti kopije product-a dobijenog iz baze. Npr ako želimo dodati novi *Product* čiji id znamo u wishlist korisnika potrebno je praviti novi product isti kao onaj dobijen iz baze jer ako dodamo direktno onaj iz baze dolazi do prepisivanja FK-a postojećeg entity-a u bazi umjesto kreiranja novog. Slično ponašanje postoji i u drugim metodama / klasama. Zbog toga se to olakša kreirajući jednom Prototype pattern za *Product* te ga samo koristimo gdje je potrebno.

## Factory

Kada bi naše klase za tipove produkata bile različite (npr. klasa za laptope, klasa za mobitele i sl.- ovo nije slučaj radi jednostavnosti) tako da imaju različite tipove i količinu informacija mogli bismo koristiti ovaj pattern da na osnovu konteksta vratimo pravi tip produkta.

## Abstract Factory

Ako bismo uveli brendove kao tipove mogli bismo imati posebne fabrike za te brendove iz kojih su izvedene stvarne fabrike koje vraćaju proizvode.