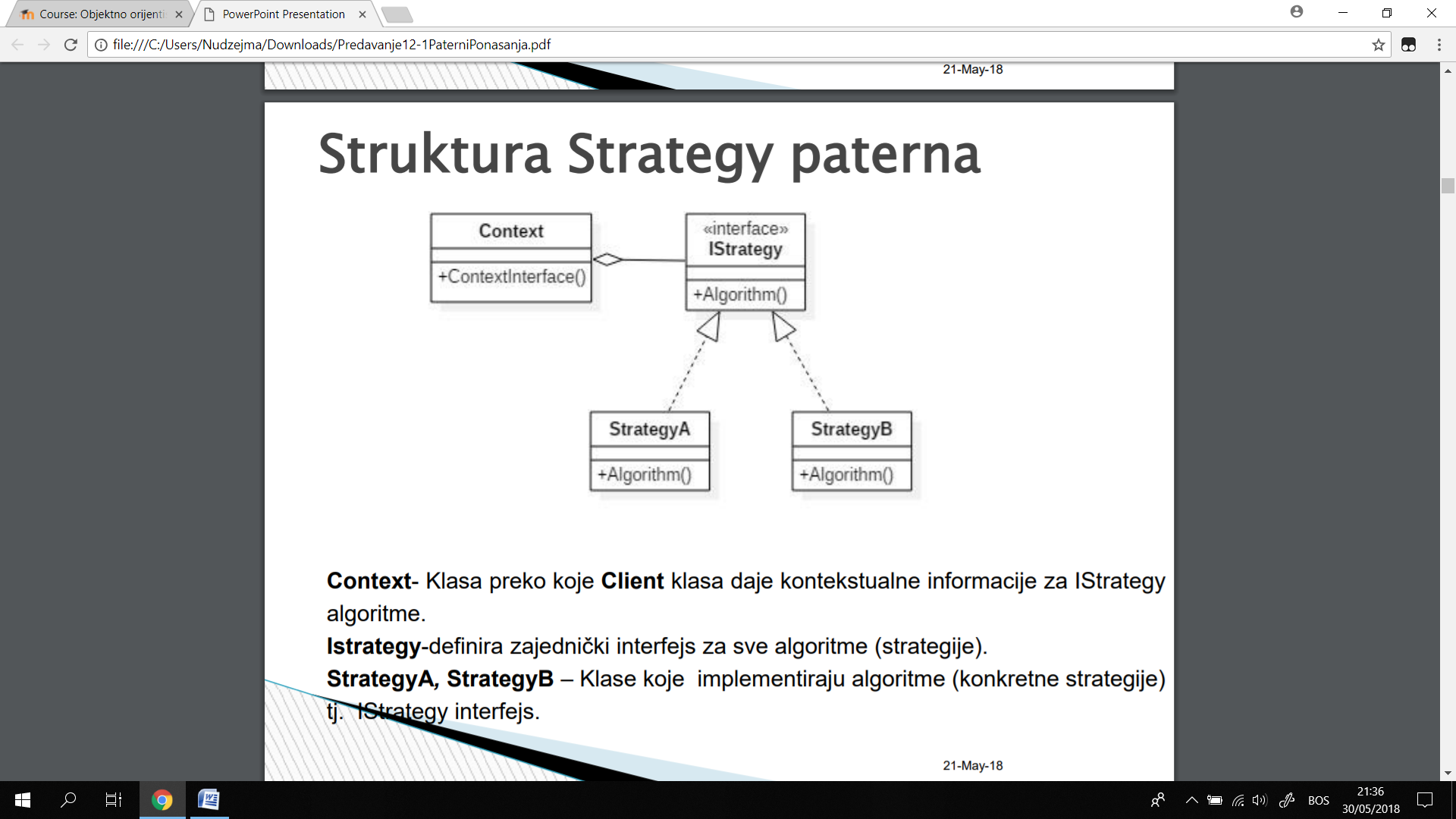
**Strategy pattern**

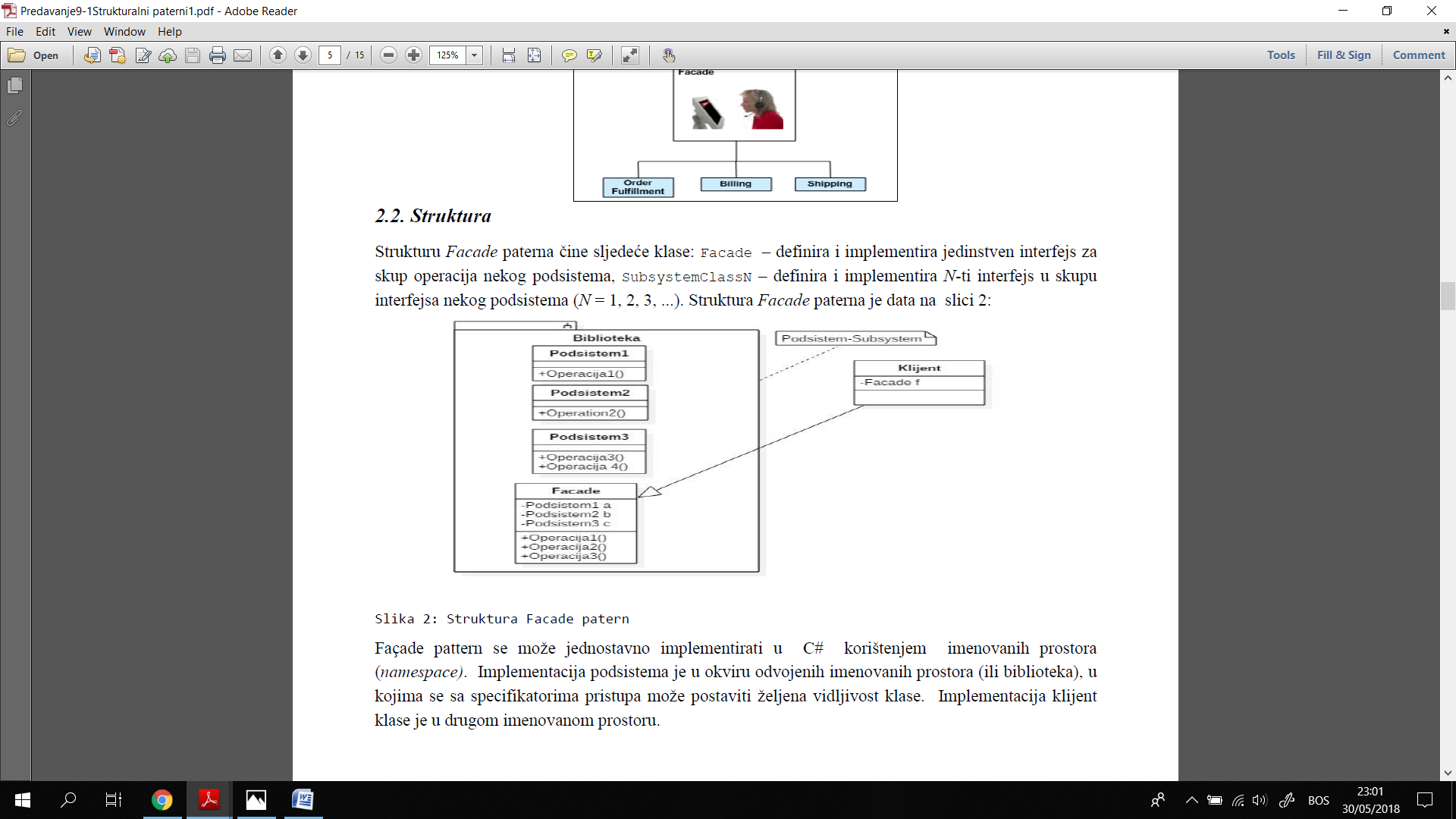
Ovaj pattern je jedan od patterna ponašanja i on izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Ovaj pattern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje.



U našem programu bi se ovaj patternmogao iskoristiti na sljedeći način. Klijent odabira koja mu je usluga potrebna, te shodno tome, vrši se narudžba za printanje, izradu slika, izradu personaliziranih predmeta ili izrada reklamnog materijala. Ovo je realizirano na način da postoji apstraktna klasa narudžba, i četiri izvedene klase za četiri usluge koje su u ponudi. Da se ova funkcionalnost implementira preko Strategy pattern-a potrebno bi bilo uvesti klasu Context koja bi imala klase PromjenaStrategije i AlgoritamStrategije pomocu kojih bi se vrsio izbor strategije.

**Facade pattern**

Ovaj pattern je jedan od strukturalnik paterna i koristi se kada sistem ima više identificiranih podsistema (subsystems) pri čemu su apstrakcije i implementacije podsistema usko povezane. Osnovna namjena Facade pattern-a je da osigura više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika). Operacije koje su potrebne određenoj korisničkoj perspektivi mogu biti sastavljene od različitih dijelova podsistema. Može se više fasada postaviti oko postojećeg skupa podsistema i na taj način formirati više prilagođenih pogleda na sistem.

****

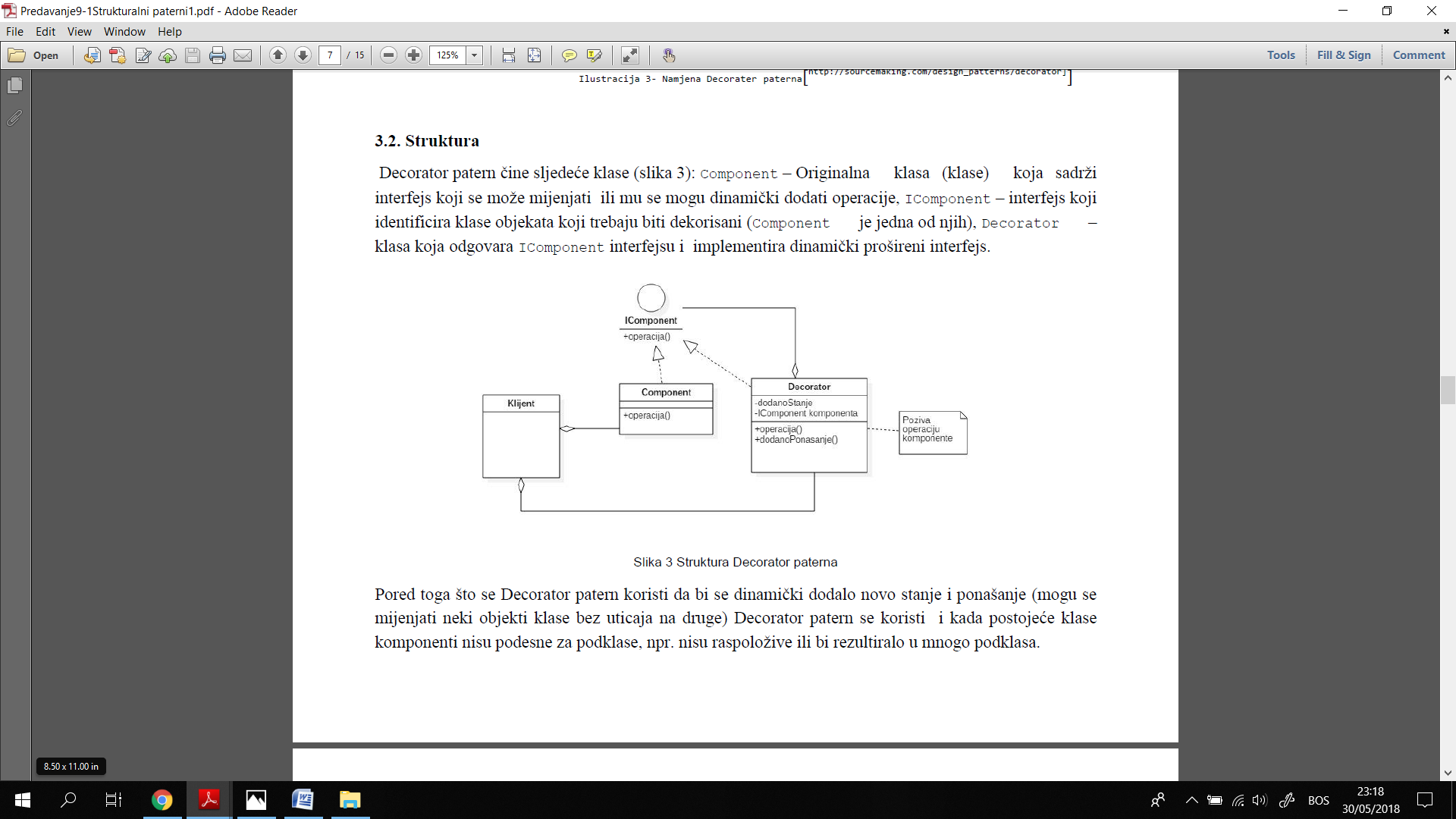
U našem programu bi se mogao implementirati ovaj pattern na sljedeći način. Obavljanje narudžbe bi se moglo odvojiti u više podsistema. Jedan bi mogao biti za recimo boju, drugi za količinu materijala/narudžbe, treći za naplatu istog... Klasa Narudzba bi objedinila te podsisteme tako što bi kao parametre sadržavala instance tih klasa i u svojim metodama koristila metode pojedinih podsistema (npr. u metodi koja vrši sastavljanje narudžbe bi se koritila metoda iz klase za kolicinu i vrstu materijala). Korisnik sve te podsisteme vidi kao jedan, pa otuda i naziv fasada.

**Decorator pattern**

Ovaj pattern je jedan od strukturalnik paterna i njegova osnovna namjena je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Objekat pri tome nezna da je urađena dekoracija što je veoma korisno za iskoristljivost i ponovnu upotrebu komponenti softverskog sistema. Na ilustraciji 3 je prikazana grafička upotreba Decorator paterna. Može se naprimjer koristi i za implementaciju različitih kompresija videa, simultano prevođenje.

Pored toga što se Decorator patern koristi da bi se dinamički dodalo novo stanje i ponašanje (mogu se mijenjati neki objekti klase bez uticaja na druge) Decorator patern se koristi i kada postojeće klase komponenti nisu podesne za podklase, npr. nisu raspoložive ili bi rezultiralo u mnogo podklasa.

Decorator patern se ne oslanja na čisto nasljeđivanje prilikom dodavanja novih atributa i ponašanja. Decorator patern nasljeđuje originalnu klasu i sadrži instancu originalne klase. Postojeći objekti se ne mijenjaju već se kreiraju novi.



U našem programu bi se mogao implementirati na sljedeći način. U slučaju da radnik preko svog računa želi iz nekog razloga napraviti narudžbu (nalazi se izvan grada/trebaju usluge nekom njegovom prijatelju), on prestaje biti obični radnik, ali nije ni obični korisnik zbog mogućih popusta za uposlenike. Da ne bi modificirali već postojeće klase, možemo korištenjem ovog pattern-a ostvariti navedenu funkcionalnost. Tj. imamo opciju obaviNarudžbu (moze biti i interfejs), nju implementira klasa Korisnik, ali i klasa Radnik (sa određenim dodacima).