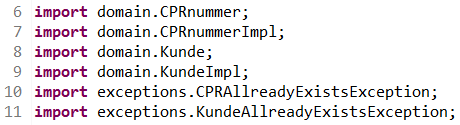
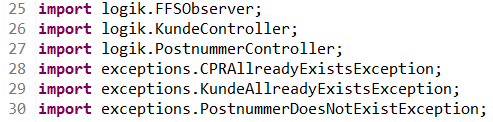
# GRASP

Et gennemgående koncept i udviklingen af vores system har været evnen til at kunne videreudvikle og/eller genbruge væsentlige dele af programmet. Vi har blandt andet valgt at anvende interfaces til de fleste klasser, for at gøre det lettere at udskifte/opdatere de dertilhørende klasser. Desuden har vi oprettet flere controllere, der hver især fungerer som information experts. Til funktionalitet på lånetilbud findes der derfor en lånetilbud controller, der håndterer logikken (der derved fjerner alt logik fra viewet) for lånetilbud. Vi har fulgt dette system igennem, men den eneste undtagelse af CPR numre. Logikken til dette ligger i kunde controlleren, da vi mente en CPR controller ville blive for lille, og alligevel være koblet til kunde controlleren. Et kig på importlisten for kundeController viser også hvilke forbindelser denne har til andre klasser



KundeController anvendes primært af KundePanel, der ligesom controlleren har begrænset sit import til de relevante klasser. Ud over en længere række GUI imports ser listen således ud:



Postnummer er ligesom CPR tæt knyttet til kunde, men der ligger alligevel lidt logik for sig selv, derfor har denne fået sin egen controller. Dette gør det ikke bare muligt at udskifte dele af eller hele controlleren, men også nemt at finde frem til den præcise metode man har brug for.

## Kobling

I vores system har vi selvfølgelig gået efter og få den mindst mulig kobling imellem objekter, dov stadig haft i hoved at for lav kobling kan føre til forvirring i det samlede billed af systemet. Da en vis kobling imellem objekter aldrig kan undgås har vi, som nævnt tideligere, gjort benytte af interfaces og controllers. Dette har selvfølgelig noget og gøre med hvilke design patterns vi har valgt og gå ud fra. Ved brug af disse interfaces har det lykkedes os, og skabe dependencies, hvilket er den svageste kobling, fremfor nedarvning som er den stærkeste. Man kan vel sige at vi har gået efter ikke at bruge nedarvning. Dette har dov været nødvendigt i vores tests. Det er også blevet brugt i vores GUI kode, men her har nedarvningen været af Jpanels osv.

## Samhørighed

Som tidligere nævnt har en høj samhørighed været i højsædet under udviklingen. Vi har så vidt muligt begrænset hver klasse til det relevante scope. Således kan der trækkes en lige linje fra KundePanel til KundeController, via KundeLogik til Kunde i domain.

For at mindske koblingen ligger der både metoder til at oprette kunde og finde kunde i kundeController. Disse metoder har som sådan ikke noget med hinanden at gøre, og for at øge samhørigheden kunne disse være lagt i klasser for sig selv. Vi har dog valgt at lægge dem sammen, dels for at undgå for små controller klasser, dels for at have en høj samhørighed ved controllere.