Exercise 2

Indledning:

Formålet med denne øvelse er at træne implementering af *associationer* mellem klasser.

Opgaven går ud på at implementere kernen i et ekspeditionssystem, der består af en række klasser. Visse af disse er givet på forhånd og findes implementeret i det sæt af source code-filer, der findes til denne opgave. Andre skal du implementere selv.

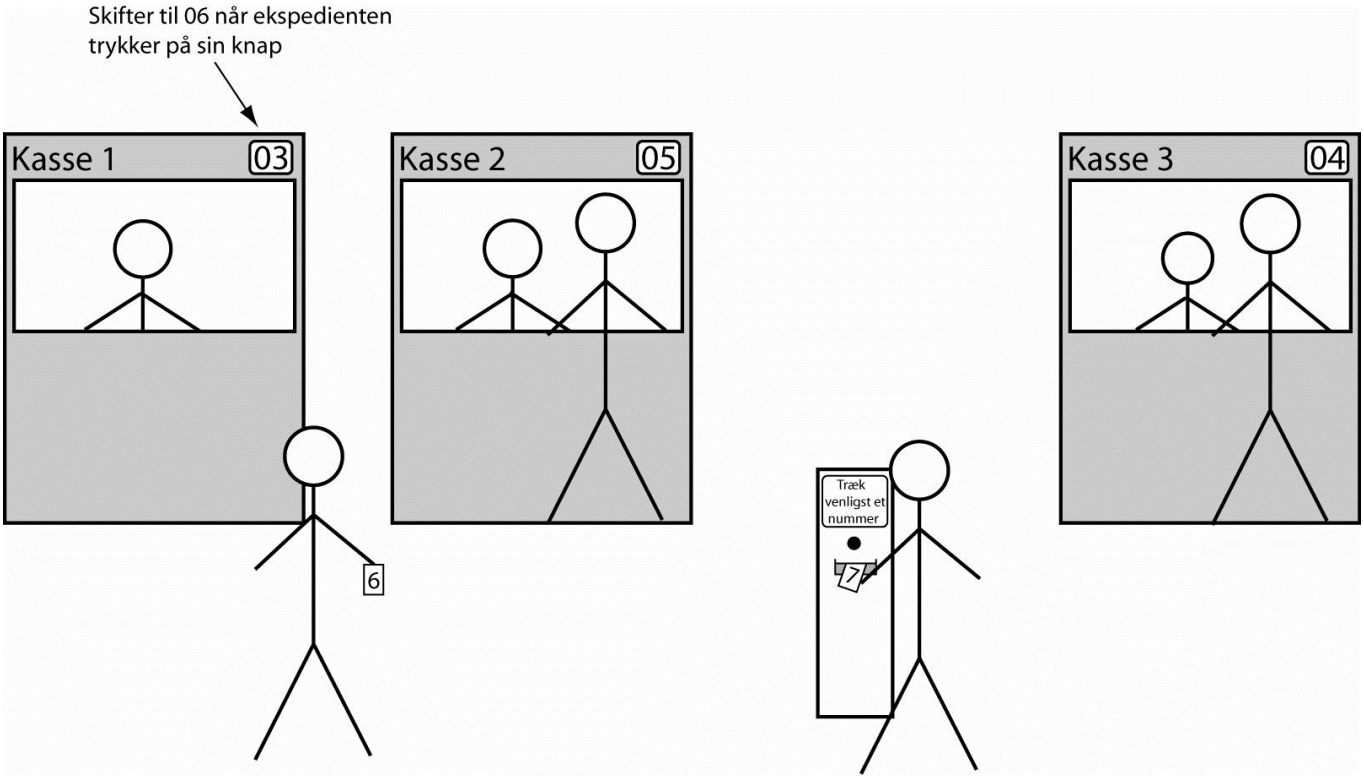
Gennem løsningen af denne opgave vil du opnå forståelse for, hvordan associationer mellem klasser, sådan som de kommer til udtryk på et UML klassediagram, implementeres i C++.

Ekspeditionssystem

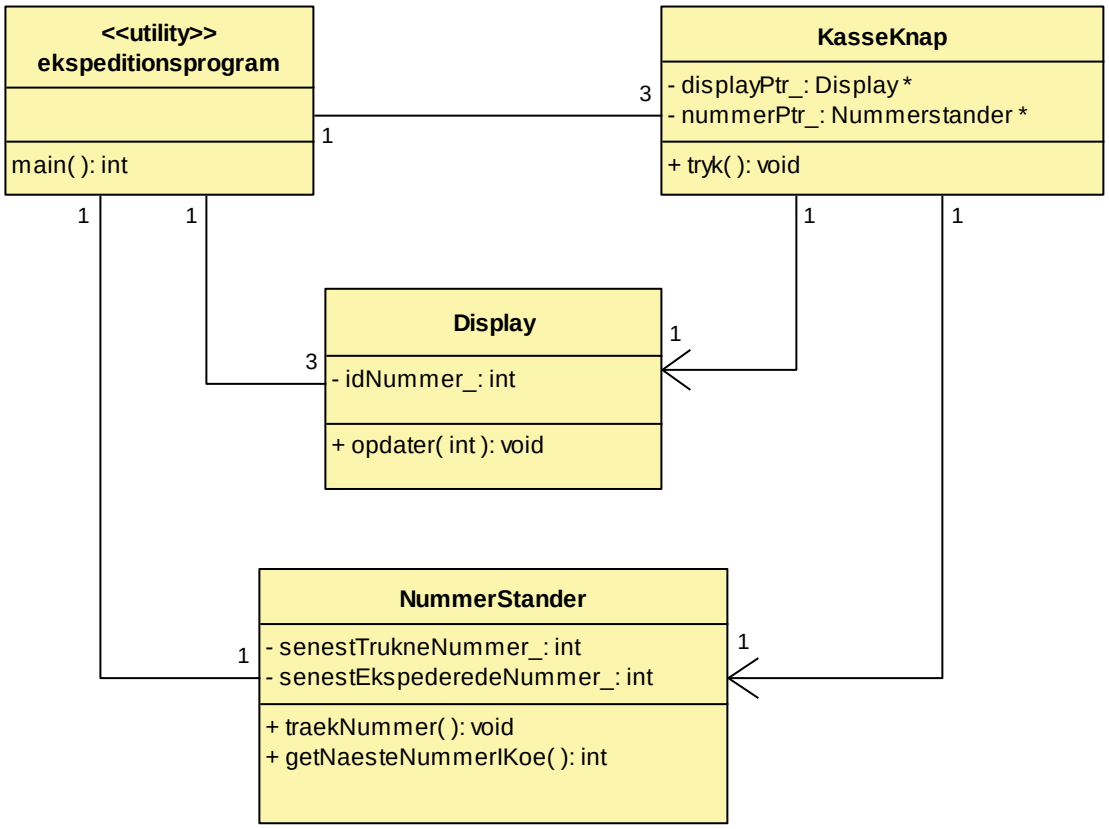
Dit firma har fået til opgave at lave et system til styring af kundeekspeditionen på posthuse, apoteker m.v. For at udvikle et sådant system kræves både stærkstrømstekniske, svagstrømstekniske og programtekniske discipliner, så du er blevet en del af en projektgruppe med personer indenfor alle 3 fagområder. Det er blevet besluttet at udvikle en første prototype, som beskrevet herunder.

Den første prototype af ekspeditionssystemet skal bestå af 3 kasser med hvert sit 2-cifrede digitaldisplay, hvorpå der vises hvilket nummer, der ekspederes ved den pågældende kasse. Ved hver kasse er der desuden en knap, som ekspedienten kan trykke på, således at nummeret skifter til det næste nummer i køen. Endelig er der en stander, hvor kunden kan trykke på en knap og derved trække sit nummer i køen.

Figuren herunder viser en situation, hvor kunden med nummer 3 **er** blevet ekspederet og har forladt stedet. Kunderne med nummer 4 og 5 er **ved** at blive ekspederet. Kunden med nummer 6 **venter** på at hans nummer kommer frem (det vil ske på kasse 1's display), og en ny kunde er ved at trække nummer 7.



Til den softwaremæssige del af systemet er man kommet frem til følgende klassediagram:



Klasserne `Display` og `Cursor` er implementeret i filerne, der ligger sammen med opgaveen. Disse filer skal inkluderes i det projekt, du opretter til din løsning.

Exercise 2.1: Implementeringsovervejelser

a) Hvad kaldes typen af relation mellem `KasseKnap` og `NummerStander`?

b) Hvilken betydning har det for din implementering, at der er en *retning* på relationen?

c) Hvordan vil du implementere denne type af relation?

Exercise 2.2: Klassen `NummerStander`

Implementér klassen `NummerStander`. Beskrivelsen af nogle af metoderne findes herunder – resten skal du selv udlede.

```
void traekNummer( void );
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: Metoden skal tælle attributen `senestTrukneNummer_` en op. Hvis den derved bliver 100 skal den nustilles.

```
int getNaesteNummerIKoe( void );
```

Parametre: Ingen

Returværdi: Nummer på næste kunde der skal ekspederes

Beskrivelse: Hvis der ingen kunder er i kø returneres -1. Ellers skal `senestEkpederedeNummer_` tælles en op. Hvis den derved bliver 100 skal den nulstilles. Til slut skal `senestEkpederedeNummer_` returneres .

Exercise 2.2: Klassen `KasseKnap`

Implementér klassen `KasseKnap`. Du kan se beskrivelsen af metoden `trykket()` herunder.

```
void tryk( void );
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: Nummeret på den næste kunde i køen hentes. Hvis det hentede nummer ikke er -1 skal kassens display opdateres med det hentede nummer.

Exercise 2.3: Hovedprogrammet

Sammen med denne øvelse ligger også filen

`ekspeditionsprogram`.cpp.

Heri er skrevet en ***del*** af hovedprogrammet. Inkluder filen i dit projekt og skriv koden til hoveprogrammet færdig.

Exercise 2.4: Komposition OG association

Løs eksamenssættet fra OPRG Vintereksamen Dec10/Jan11 (findes i opgavesamlingen fra 1. semester).