Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

# TDT4102 Prosedyre- og objektorientert programmering Vår 2019

**Øving 8**

## Frist: 2019-03-08

## Mål for denne øvinga:

- Lære om operatorar og interaksjon mellom klasser av ulike typar.
- Lære å implementere og å bruke klasser.
- Lære å bruke enkle klasser for eit enkelt grafisk brukargrensesnitt (GUI).

## Generelle krav og tilrådinger:

- Bruk dei eksakte namn og spesifikasjonar gjeve i oppgåva.
- Teorioppgåver svarar du på med kommentarar i kildekoden slik at læringsassistenten enkelt finn svaret ved godkjenning.
- Det anbefalast å nytte ein programmeringsomgjevnad(IDE) slik som Visual Studio eller XCode.
- 70% av øvinga må godkjennast for at den skal vurderast som bestått.
- **Hugs å lage eit TDT4102-grafikk prosjekt for denne øvinga.**
- Øvinga skal godkjennast av stud.ass. på sal.

## Tilrådd lesestoff:

- Kapittel 16 og 17 i PPP (Sjå forresten fagets førebelse pensumliste som ligg på BB under praktisk informasjon)

## DEL 1: NTNU-samkøyring (60%)

Du har fått sommarjobb i det nye studentføretaket «EcoTrans» som er starta av nokre miljømedvitne NTNU-studentar. Omorganiseringa til nye NTNU har skapt eit behov for koordinert transport mellom byane Trondheim, Ålesund og Gjøvik. Kvar veke reiser mange tilsette og studentar mellom desse byane, ofte i privatbilar med ledige sete. EcoTrans vil lage eit enkelt datasystem for å stimulere til miljøvenleg samkøyring, og i denne oppgåva skal du skrive nokre kodebitar for eit slikt system.

# 1 Car-klassa (10%)

**a) Deklarer ei klasse `Car`.**

`Car` skal ha eit heiltal `freeSeats` som privat medlemsvariabel som indikerer kor mange ledige sete det er i bilen. `Car` skal også ha to `public` medlemsfunksjonar `hasFreeSeats` og `reserveFreeSeat`. `hasFreeSeats` returnerer `true` om bilen har ledige seter, og `false` elles. `reserveFreeSeat` «reserverer» eit ledig sete ved å dekrementere `freeSeats`-variabelen (du kan gå ut frå at funksjonen berre verte kalla på om det er ledige sete).

Deklarasjonar for medlemsfunksjonane:

```
bool hasFreeSeats() const;
void reserveFreeSeat();
```

**Nyttig å vite: Const correctness**

Det er god praksis å markere medlemsfunksjonar som ikkje endrar objektet med `const`. Dette gjer det enklare å finne feil i koden, og let oss bruke medlemsfunksjonane sjølv om objektet er konstant.

```
class NumberClass {
    int number;
    public:
    NumberClass(int number)
        : number{number} {}

    // markert const
    int getNumber() const {
        return number;
    }

    // ikkje markert const
    void setNumber(int newNumber) {
        number = newNumber;
    }
}
```

```
int main () {
    NumberClass x{3};

    int i = x.getNumber(); // OK
    x.setNumber(i+1); // OK

    const NumberClass y(4);
    int j = y.getNumber(); // OK

    // IKKJE OK!
    // Kompileringsfeil:
    // kan ikkje kalle ein funksjon
    // som ikkje er markert const,
    // på eit const objekt
    y.setNumber(j+1);
}
```

**b) Deklarer og implementer ein konstruktør som tek inn kor mange ledige seter bilen har.**

**c) Implementer `hasFreeSeats` og `reserveFreeSeat`.**

Hugs at deklarasjonen skal vere i `Car.h`, og definisjonen (implementasjonen) skal vere i `Car.cpp`.

# 2 Person-klassa (20%)

**a) Deklarer ei klasse `Person`.**

Denne skal ha dei private medlemsvariablane `name` og `email`, begge av typen `string`. I tillegg skal klassa ha ein privat medlemsvariabel `car`, som er ein peiker til `Car`. Legg merke til at vi ønskjer å bruke ein peikar og ikkje referanse til `Car`-objektet.

Grunnen til at vi ønskjer å bruke ein peikar er fordi einkvar peikar kan ha verdien `nullptr` og det passar fint for å representere at ein person *ikkje* har bil. Om vi nytta referanse i staden for peikar ville det vorte vanskelegare å representere dette. Dette er godt forklart i læreboka §17.9.1, og mot slutten av det avsnittet er det oppsummert når ein anbefaler pass-by-value, peikar, eller referanse-parameter.

Klassa skal ha ein konstruktør som set `name`, `email` og `car` til verdiar gjeve av parameterlista. For `car` nyttar vi `nullptr` som eit såkalla «default argument» (standard-verdi). Det tyder at konstruktøren kan brukast med berre de to første parametrane, og då vil den tredje få denne standard-verdien. Sjå også nyttig-å-vite-boks om dette temaet. Deklarer ein `get`-funksjon både for `name` og `email`. Deklarer også ein `set`-funksjon for `email`.

**b) Implementer konstruktøren og get-/set-funksjonane frå førre deloppgåve.**

Bruk initialiseringsliste i konstruktøren.

**Nyttig å vite: default arguments**

For å unngå at ein skal definere fleire ulike funksjonar som gjer det same, men har ulik antal parametrar i parameterlista, så finnast det *default arguments*.

Til dømes kan ein funksjon som alltid skal leggje saman to tal vere standardisert til å inkrementere det første argumentet med ein, dersom det andre argumentet ikkje er oppgjeven. I staden for å lage to funksjoner som gjer same arbeid eller kallar på ein annan, er det formålstenleg å samle dei. Dette kan ein også bruke med medlemsfunksjonar i klassar.

```
void Adder(int a, int b = 1);
//void Adder(int a = 1, int b); // Gir kompileringsfeil

int main() {
    Adder(42, 42); // a+b=84
    Adder(42); // a+b=43
}

void Adder(int a, int b) {
    cout << "a+b=" << a+b;
}
```

Default argument skrivast i deklarasjonen, men ikkje i definisjonen. Dei må også skrivast sist i parameterlista.

**c) Lag medlemsfunksjonen `hasAvailableSeats`.**

Funksjonen returnerer `true` om personen eig ein bil og bilen har ledige sete.

**d) Overlast `operator<<`, som skal skrive ut innhaldet i Person til ein `ostream`.**

**Drøft:**

- Kvifor bør denne operatoren deklarerast med `const`-parameter? (t.d. `const Person& p`)
- Når bør vi, og når bør vi ikkje (ev. kan ikkje) nytte `const`-parameter?

**e) Skriv testar for `Person` i `main`**

Opprett fleire personar, og prøv å teste ulike tilfelle (t.d. har personen bil? Kva med når personen ikkje har bil?).

# 3 Meeting-klassa (30%)

**a) Deklarer ein `scoped enum`, med namn `Campus`, som inneheld verdiar for dei ulike byane. Overlast `operator<<` for `Campus`.**

La deklarasjonen av `enum class Campus` liggje i `Meeting.h`.

**b) Definer klassa `Meeting`.**

Klassa skal ha følgjande private medlemsvariablar:

```
int day;
int startTime;
int endTime;
Campus location;
string subject;
const Person* leader;
set<const Person*> participants;
```

Implementer `get`-funksjonar for `day`, `startTime`, `endTime`, `location`, `subject`, og `leader` som ein del av klassedefinisjonen.

**c) Lag medlemsfunksjonen `addParticipant`.**

Den skal leggje ein peiker til eit `Person` objekt i `participants`.

**d) Legg til `meetings` som ein statisk, privat medlemsvariabel.**

`meetings` skal innehalde peikarar til alle møtene som er oppretta.

```
static set<const Meeting*> meetings;
```

Du må også initialisere variabelen i `Meeting.cpp`:

```
set<const Meeting*> Meeting::meetings;
```

**e) Lag ein konstruktør for Meeting-klassa som tek inn `day`, `startTime`, `endTime`, `location`, `subject`, og `leader`.**

Hugs å leggje til det nye møtet i `meetings`, og at møteleiaren også er ein deltakar.

**f) Lag ein destruktør for `Meeting`-klassa.**

Her må du fjerne peikaren til objektet frå `meetings`.

**g) Lag funksjonen `getParticipantList`.**

Funksjonen har ingen parametrar, og returnerer ein `vector` med namn på deltakarane.

**h) Overlast `operator<<` for `Meeting`.**

Denne operatoren skal IKKJE vere ein `friend` av `Meeting`. Du står fritt til å velje format sjølv, men du skal skrive ut `subject`, `location`, `startTime`, `endTime`, og namnet på møteleiaren. I tillegg skal han skrive ut ei liste med namna på alle deltakarane.

Test funksjonen din frå `main`.

**i) Skriv funksjonen `findPotentialCoDriving`.**

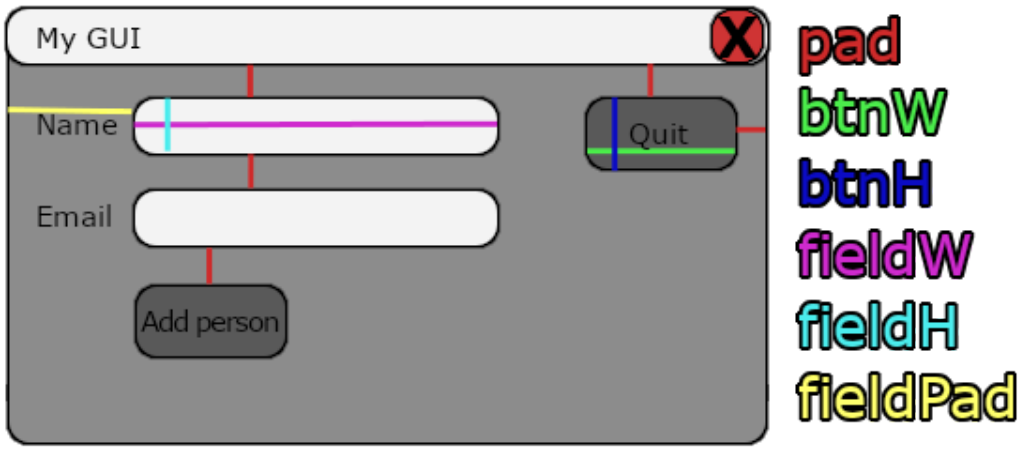
Dette skal vere ein medlemsfunksjon i `Meeting`. Funksjonen skal ikkje ta inn noko, og skal returnere ei liste med `Person`-peikarar til personar som har ledige seter i bilen til *andre* møte på same stad, på same dag og med både start-tid og slutt-tid som er mindre eller lik ein time forskjellig.

*Hint: Funksjonen vil ha følgjande returtype:* `vector<const Person*>`.

## DEL 2: GUI for samkjøring og møteplanlegging (40%)

For å gjere programmet meir brukarvenleg vil dykk lage eit grafisk brukargrensesnitt (GUI) der passasjerar kan melde seg inn. Den skal ha to tekstfelt, eit for å skrive inn passasjerens namn og eit for e-post. Det skal også vere to knappar: ein for å leggje til personen og den andre for å avslutte programmet.

Før ein går laus på kodinga til eit GUI, er det veldig lurt å danne seg ei skisse for korleis vindauget skal sjå ut. Det er også lurt å namngje alle ulike avstander i vindauget. Dette er delvis slik ein ikkje treng å sjonglere alle dei ulike verdiane i hovudet, men mest fordi ein då kan endre på alle felles avstandar på ein stad. Under kjem ei mogeleg skisse av GUI-et der kvar farge er kopla til ein variabel. Det er ikkje eit krav at du følgjer denne skissa, men alle elementa skal vere med til slutt.




Det er anbefalt med `using namespace Graph_lib;` i `.h` fila for å unnlate `Graph_lib::`-prefikset.

Alle variablar og funksjonar i resten av oppgåvene skal vere medlem av vindauge-klassa vi lagar.

# 4 Oppretting av GUI (20%)

Til denne oppgåva skal vi bruke `Window` i staden for den tidlegare `Simple_window`, noko som gjer at vi overlet programstyringa til vindauget sjølv. I praksis tyder det at vi må lage vår eiga vindauge-klasse som arvar frå `Window` og at klassa skal styre programmet.

**a) Lag ei ny klasse `MeetingWindow`, som arvar `public` frå `Window`, og konstruktøren `MeetingWindow(Point xy, int w, int h, const string& title)`.**
Sidan `Window` ikkje har ein default-konstruktør, må du kalle `Window`-konstruktøren i initialiseringslista, der argumenta skal vere dei du tok inn i `MeetingWindow`-konstruktøren. La konstruktørkroppen stå tom inntil vidare.

**b) Legg inn eit kall til funksjonen `gui_main` i `main`.**
Denne overlet programstyringa til vindauget du har laga. Lag også eit objekt av typen `MeetingWindow` før denne funksjonen og prøvekøyr. Du skal få opp eit blankt vindauge.

**c)** Før du byrjar på å leggje inn element, er det lurt setje inn nokre heiltal i klassa som definerer oppsettet i vindauget. Sjå skissa over for eit forslag. Sidan desse ikkje skal endrast og er definerte før kompileringa, er det lurt å deklarere dei `static constexpr int` og gi dei verdiar direkte i `.h` fila. Breidda og høgda til vindauget kan du hente ut med dei nedarva medlemsfunksjonane `x_max()` og `y_max()`. Desse funksjonane eksisterer frå før, så du treng ikkje å implementere noko med dei.

**Nyttig å vite: Kort om callback og `Widget`**

Ein callback-funksjon vert kalla av ein knapp i GUI-et når du trykkjer på den og har signaturen `void cb_my_callback(Address, Address pw);`. Den første parameteren bryr vi oss ikkje om, men den andre er adressa til GUI-vindauget. For at du skal kunne bruke vindauget, må du først fortelje funksjonen at den må tolke `pw` som ein `MeetingWindow`-referanse. Dette gjer du med å kalle `reference_to<MeetingWindow>(pw)`, og deretter kan du kalle medlemsfunksjonar med vanleg dot-notasjon. Eksempel og ytterlegare informasjon står i PPP 16.3.1.

Alle GUI-element vi skal bruke arvar frå type `Widget` og må konstruerast i konstruktøren til `MeetingWindow`. Konstruktørane deira tek alle inn dimensjonane til elementet samt ein merkelapp (label). For å aktivere elementa må du også passe på å kalle `MeetingWindow` sin medlemsfunksjon `attach` med dei.

**d) Lag callback-funksjonen `void cb_quit(Address, Address pw)`.**

For at callback-funksjonen skal få tilgong til private medlemmar, må du deklarere funksjonen som static i `MeetingWindow`.

Denne callback-funksjonen skal nyttast av avslutnings-knappen, så derfor må den kalle den arva medlemsfunksjonen `hide` som avsluttar vindauget.

**e) Legg inn eit `Button`-objekt, `quitBtn`, som privat medlemsvariabel.**

`quitBtn` må konstruerast i initialiseringslista. Det siste argumentet er callback-funksjonen den skal bruke, og for å sende inn ein funksjon som argument skriv du funksjonsnamnet utan parentes (`cb_quit`). Prøv å køyre programmet her og sjå om det funkar som forventa.

## 5 Person-funksjonalitet (20%)

**a) Legg til to `In_box`: `personName` og `personEmail`.**

Desse er to innskrivingsfelt for parametrane til ein ny `Person`.

**b) Legg inn `Vector_ref<Person> people`, og så definer og implementer ein ny funksjon, `void addPerson()`.**

Denne vektoren skal innehalde alle personar som vert lagt til. `addPerson` skal lese det som er skrive inn i tekstfelta og leggje til ein ny person i vektoren med desse argumenta. Dette skal vere eit anonymt/namnlaust objekt, så det er viktig at du brukar `new`:

```
people.push_back(new Person{/*Dine argument*/});
```

For å hente innhaldet i tekstfelta, må du kalle medlemsfunksjonen `get_string`. Sjekk også om ein av parametrane manglar.

**c) Legg til ein ny `Button`, `personNewBtn` med ein tilhøyrande callback-funksjon, `cb_new_person`**

Callback-funksjonen skal kalle `addPerson`.

**d) Test om programmet fungerar som venta.**

Ein enkel måte å gjere dette på er å lage ein `public` funksjon som printar alle personane i vektoren. Denne funksjonen kan du kalle i `main` etter `gui_main`.

# 6 Utviding av GUI (Frivillig)

I denne oppgåva kan du fullføre GUI-et for samkøyringa. Det skal no verte to sider: ei for `Person` og ei for `Meeting`. Ein skal kunne sjå alle møte/personar som er innførte og kunne leggje inn nye. Vindauget skal ha ein knapp for å avslutte, to knappar som respektivt byttar til `Meeting`- og `Person`-sida, eit tekstfelt for informasjon, eit felt for kvar parameter å leggje inn, og to knappar som respektivt legg til ein ny `Person` eller `Meeting`. I tillegg skal der vere to felt der du kan velje ein person å leggje til eit spesifikt møte, og ein knapp som utfører dette.

**a) Legg til ein ny `In_box`, `personSeats`, og ein `Vector_ref<Car>`, `cars`.**

Denne skal du bruke for å gi personane ein bil i `addPerson`. Dersom `personSeats` har eit tal som er større enn null, noko som du finn ut ved å kalle `get_int`, lagar du eit `Car`-objekt i `cars` og gir peikaren til denne til `Person`-konstruktøren. Sjåføren må først ha plass, så du må også "reservere" eit sete i `Car`-objektet!

**b) Legg til ein `Multiline_out_box`, `data`, som privat medlem.**

Denne typen er definert i den utdelte koden `"Graph_lib.h"` og er ein `Out_box` som kan vise fleire linjer. Denne skal fungere som display i vårt GUI.

**c) Deklarer og definer medlemsfunksjonen `void displayPeople()`**

I denne funksjonen skal du leggje inn alle personane til displayet `data`, og dette gjer du med å kalle medlemsfunksjonen `put` som tek inn ein streng. Denne vil tolke `\n` som ny linje. Sett eit kall av denne funksjonen ved slutten av `addPerson`. Køyr programmet og sjå om personane du leggjer inn kjem opp på displayet.

*Hint: `stringstream` har medlemsfunksjonen `str` som returnerar innhaldet sitt som ein `string`. Du kan utnytte dette og gjenbruke tidlegare kode.*

**d)** Vi har lyst å leggje til tilsvarande funksjonalitet for ei `Meeting`-side.

**Lag to nye funksjonar, `void showPersonPage()` og `void showMeetingPage()`.**

Desse skal vi bruke til å bytte mellom `Person` og `Meeting` sidene. For å gjere dette, må `showPersonPage` kalle medlemsfunksjonen `show` på alle element som er knytta til den sida, medan `showMeetingPage` må kalle `hide`. Det omvende gjeld for alle komande element som vert knytta til `Meeting`-sida. `data` kan vere felles, men då må du kalle `displayPeople` i `showPersonPage`.

**e) Legg ein `Menu`, `pageMenu`, og to tilhøyrande callback-funksjonar, `cb_persons` og `cb_meetings`.**

Callback-funksjonane skal respektivt kalle `showPersonPage` og `showMeetingPage`. Ein av parametrane til `Menu` er ein `enum` av typen `Menu::Kind` som fortel om menyen er vertikal eller horisontal. I konstruktøren til `MeetingWindow` skal du leggje til to `Buttons` i `pageMenu` som får sin respektive callback-funksjon.

*Obs! I `Graph_lib` er det ein feil som gjer at alle `Button`-objekt som leggast inn i ein `Menu` må være dynamisk allokerte med `new`. Når menyen går ut av scope vil den automatisk rydde opp etter seg, så du trengjer ikkje sjølv å kalle `delete` på knappane.*

**f) Legg inn fire nye `In_box` til `Meeting`-sida: `meetingSubject`, `meetingDay`, `meetingStart`, og `meetingEnd`.**

Desse skal vi seinare bruke for å leggje inn `Meeting`-objekt. Nå er det lurt å kalle `personPage` på slutten av `MeetingWindow`-konstruktøren.

**g) Legg inn to `Choice` til `Meeting`-sida: `meetingLocation` og `meetingLeader`.**

`Choice` er definert i den utdelte `Graph_lib.h` fila og lagar ei rullegardinliste. For å leggje til eit val må du kalle funksjonen `add` med ein strengparameter. Legg inn dei tre relevante stadane til `location` i `MeetingWindow`-konstruktøren. Før du leggjer til val i `leader`, er det lurt å lage ein `vector<const Person*>` medlemsvariabel. Utvid `addPerson` til at dersom personen har ein bil med ekstra sete, leggjer du ho til i den nye vektoren og i `leader`.

**h) Legg inn ein ny funksjon, `void addMeeting()`, ein callback som kallar denne, `cb_new_meeting`, og ein knapp med denne callbacken, `meetingNewBtn`.**

`addMeeting` skal lese parametrar frå felta og konstruere eit `Meeting` i ein ny `Vector_ref<Meeting>`, `meetings`. `Choice` har medlemsfunksjonen `value` som returnerar posisjonen til valet i lista, så bruk dette for å finne rett stad eller rett møteleiar i vektoren over bilførarar. Hugs å sjekke om parametrane er gyldige.

**i) Lag ein funksjon som leggjer inn alle møtene til displayet `data`.**

Kall på denne i `addMeeting` og `showMeetingPage`.

**j) Sett inn to nye `Choice`, `meetingChoice` og `personChoice`, ein knapp, `participantNewBtn` og ein funksjon `addParticipant`.**

`meetingChoice` skal innehalde alle møtene som val, medan `personChoice` skal innehalde alle personane. `addParticipant` skal leggje personen i `personChoice` til møtet i `meetingChoice`. Til det vil du trengje ein ny callback-funksjon og medlemsfunksjon.