Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

# TDT4102 Prosedyre- og objektorientert programmering Vår 2017

**Øving 6**

## Frist: 2017-02-24

## Mål for denne øvinga:

- Lære om operatorar og interaksjon mellom klasser av ulike typar.
- Lære å implementere og å bruke klasser.
- Lære å bruke bibliotek frå tredjepart

## Generelle krav:

- Bruk dei eksakte namna og spesifikasjonane som er gjevne i oppgåva.
- Det er valfritt om du vil bruke ein IDE (Visual Studio, Xcode), men koden må vere enkel å lese, kompilere og køyre.

## Tilrådd lesestoff:

- Kapittel 5.4, 7, 8.1, 8.3,& 10.3 Absolute C++ (Walter Savitch)

## DEL 1: NTNU-samkøyring (50%)

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 seter 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 blir kalla på om det er ledige sete).

Prototypar 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) {
        this->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 et 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 (15%)

**a) Deklarer ei klasse `Person` som skal ha dei private medlemsvariablane `name` og `email`, begge av typen `std::string`. I tillegg skal klassa ha ein privat medlemsvariabel `car`, som er ein peiker til `Car`.**

Klassa skal ha ein konstruktør som set `name`, `email` og `car` til verdiar gjeve av parameterlista. Deklarer ein `get`-funksjon både for `name` og `email`. Deklarer også ein `set`-funksjon for `email`.

*Hint: Du kan lese meir om `get`- og `set`-funksjonar i boka (kap. 6.2 i seksjonen «Accessor and Mutator Functions»).*

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

Bruk initialiseringsliste i konstruktøren.

**c) Lag ein konstruktør som tek inn `name` og `email`. Bruk delegerande konstruktør.**

Her kan det vere nyttig å bruke `nullptr` for å indikere at ein person ikkje eig ein bil.

**Nyttig å vite: Delegerande konstruktør**

Ein delegerande konstruktør (en. *delegating constructor*), nytt i C++11, let oss bruke ein annan konstruktør for å initialisere variablane, og dermed unngår vi å duplisere kode.

```
class Eksempel {
    public:
    Eksempel(int a) {
        // Kode for konstruktør ein
    }
    Eksempel(int a, int b) : Eksempel(a) {
        // Kode for konstruktør to
        // Konstruktør ein nyttast for å initiere det første argumentet
    }
}
```

Sjå boka s. 301-302 (6th Ed.) for meir om delegerande konstruktørar.

**d) Lag medlemsfunksjonen `hasAvailableSeats`.**

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

**e) Overlast `operator<<`, som skal skrive ut innhaldet i Person til ein `std::ostream`. Drøft:**

- Kvifor bør denne operatoren ta `const` argument? (t.d. `const Person& p`)
- Når bør vi, og når bør vi ikkje (evt. kan ikkje) nytte `const`-argument?

**Nyttig å vite: Overlasting av `operator<<`**

Vi kan overlaste `operator<<` til å kunne ta inn objekt av eigendefinerte typar eller klasser. Sidan operatoren tek inn ein `std::ostream` som første argument, kan han ikkje implementerast som ein medlemsfunksjon.

Deklarasjonen til ein overlasta `operator<<` for ein `Person`-klasse får definisjonen

```
std::ostream& operator<<(std::ostream& os, const Person& p)
```

Du kan lese meir om `operator<<` i boka s. 352-358 (6th Ed.).

*Tips: Å overlaste denne operatoren tidleg vil gjere livet ditt enklare når du skal teste ei klasse du lagar.*

**f) Overlast `operator<` og `operator==`.**
Samanlikningar skal gjerast på `name`, slik at det er mogleg å bruke operatorane til å sortere personar etter namn i alfabetisk rekkefølge.

**g) 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 (25%)

**a) Deklarer ein `enum`, med namn `Campus`, som inneheld verdiane `TRH`, `GJO`, og `AAL`. Overlast `operator<<` for `Campus`.**
La deklarasjonen av `enum Campus` ligge i `Meeting.h`.

**b) Definer klassa `Meeting`.** Klassa skal ha følgjande private medlemsvariablar:

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

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

**c) Lag medlemsfunksjonen `addParticipant`.**
Vi ønsker at `participants` skal vere sortert etter namn. Sørg for at det nye `Person`-objektet blir sett inn på rett plass. Dette skal gjerast **utan** bruk av `std::sort`.

**d) Legg til `meetings` som ein statisk, privat medlemsvariabel.**
`meetings` skal innehalde peikarar til alle møtene som er oppretta.

```
static std::vector<const Meeting*> meetings;
```

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

```
std::vector<const Meeting*> Meeting::meetings;
```

**e) Lag ein konstruktør for Meeting-klassa som tek inn `day`, `startTime`, `endTime`, `location`, `subject`, og `leader`.**
Hugs å legge 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 `std::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 ein liste med namna på alle deltakarane.
Test funksjonen din frå `main`.

**i) Skriv funksjonen `findPotentialCoDriving`.**
Funksjonen skal ikkje ta inn noko, og skal returnere ei liste med `Person`-peikarar til personar som har ledige seter.
*Hint: Funksjonen vil ha følgjande returtype `std::vector<const Person*>`.*

## DEL 2: TargetPractice med SFML (50%)

Du skal no byrje på eit nytt prosjekt, les installasjonsinstruksjonane for hjelp til å sette opp SFML-prosjekt på di platform.

Du må også laste ned utdelte filer frå itslearning. Viss du brukar Visual Studio, bruk eksempelprosjektet som utgangspunkt, men bruk dei utdelte filene i staden for `main.cpp`. Dersom du brukar Xcode, lag eit nytt SFML-prosjekt og legg til dei utdelte filene. Denne øvinga byggjer på øving 3, så du må også kopiere `cannonball.cpp`, `cannonball.h`, `utilities.cpp` og `utilities.h` frå øving 3 inn i prosjektet ditt. Dersom du ikkje har gjort øving 3, kan du laste ned LF for øving 3 og kopiere filane derifrå.

Vi har i headerfila `GameObjects.h` definert (medlems-)funksjonar, konstantar og medlemsvariablar. Vi har også implementert nokre funksjonar for deg i `GameObjects.cpp`. Dersom du treng fleire variablar og/eller hjelpefunksjonar kan du legge til desse undervegs.

SFML definerer i utgangspunktet koordinaten $(0, 0)$ til å vere øvst til venstre. Vinklar blir målt i grader, der 0 grader er nedover og positiv retning for rotasjon er med klokka. For å gjere koordinatrekninga enklare har vi i den utdelte koden spegelvendt koordinatsystemet, slik at $(0, 0)$ er nede til venstre, 0 grader er opp og positiv retning for rotasjon er mot klokka.

### 4 Target (10%)

I denne oppgåva skal vi lage eit einsfarga kvadrat, som skal fungere som målskive. Klassa `Target` er allereie deklarert i `GameObjects.h` - det kan vere lurt å sjå på deklarasjonen før du byrjar å implementere funksjonane.

**a) Implementer konstruktøren til `Target`.**

For å initialisere `shape` til rett størrelse kan du bruke ein `sf::Vector2f`:

```
shape = sf::RectangleShape(sf::Vector2f(size, size));
```

Hugs å initialisere fyllfargen (du vel denne sjølv) og posisjonen til `shape`. Du kan bruke medlemsfunksjonane `setFillColor` og `setPosition` til dette.

*Hint: Det kan vere lurt å slå opp i dokumentasjonen (http://www.sfml-dev.org/documentation/2.3.2/annotated.php) for å finne ut korleis desse funksjonane fungerer.*

**b) Implementer medlemsfunksjonane `update`, og `draw`.**

`update` tek ikkje inn noko, og gjer ingenting, vi har berre med denne funksjonen slik at klassene kan brukast på same måte. `draw` tek inn `sf::RenderWindow&`, og teiknar `shape` på vindauget som vert teken inn som argument.

*Hint: `sf::RenderWindow` har ein medlemsfunksjon `draw` som tek inn ulike `sf::Shape`-objekt, mellom andre `sf::RectangleShape` og `sf::CircleShape`.*

**c) Test at `Target`-klassa fungerer.**

Opprett eit `Target`-objekt i `main` i `game.cpp`, med ein tilfeldig posisjon. Bruk `update` og `draw`, og sørg for at objektet blir plassert ulike stader kvar gong spelet startar.

*Hint: Sjå på kommentarane i `game.cpp` for hint om kvar du skal opprette objektet, og kalle på funksjonane.*

# 5 Cannon (20%)

I klassedeklarasjonen har vi allereie definert størrelsen på `Cannon`, i tilegg til ein del andre variablar. Les gjennom deklarasjonen av `Cannon`, før du byrjar å implementere klassene.

**a) Implementer konstruktøren til `Cannon`.**

Bruk medlemsvariablane til å initialisere `shape`-objektet. Vi ønsker å rotere rektangelet rundt endepunktet, og må dermed flytte referansepunktet (*origin*) til `shape`:

```
shape.setOrigin(width/2, 0);
shape.setPosition(0, 0);
```

Hugs å konstruere `shape` først, og gi figuren ein farge.

**b) Implementer `Cannon::update`.**

Medlemsfunksjonen skal gi `shape` rett vinkel. Sidan utgongsvinkelen, `theta`, er målt frå bakkenivå, må vi trekke 90 grader frå `theta` for få korrekt vinkel. Bruk `shape.setRotation` for å sette vinkelen til `shape`.

**c) Implementer `draw`.**

Sidan alt arbeidet blir gjort i `update` skal ikkje `draw` gjere noko anna enn å teikne `shape`.

**d) Test at `Cannon` fungerer.**

Opprett eit `Cannon`-objekt i `main`, og bruk `update` og `draw` slik at objektet blir oppdatert og teikna.

**e) Implementer funksjonane `incrementAngle`, `decrementAngle`, `incrementVelocity` og `decrementVelocity`.**

Desse medlemsfunksjonane skal gjere vinkelen og farten større eller mindre.

**f) Test at funksjonane fungerer.**

I `main`, legg til nye `case` i ein `switch`, slik at du kan styre kanonen ved hjelp av t.d. piltastane. Sjå i kommentarane for indikasjonar på kor du skal legge til desse. Her er eksempel på eit `case`:

```
case sf::Keyboard::Up:
    cannon.incrementVelocity()
    break;
```

Du kan også bruke `Down`, `Left` eller `Right` i staden for `Up`.

**g) Implementer medlemsfunksjonen `shoot`.**

Denne funksjonen skal opprette eitt objekt av typen `Cannonball` og returnere dette. Sjå i headerfila for å finne kva for argument du skal bruke. Du kan bruke `getTipX()` og `getTipY()` for å finne startkoordinatane til kanonkula.

I `main` må du gjere det følgjande:

- Opprett ein `vector<Cannonball>`, som skal innehalde alle kanonkulene.
- Legg til ein ny `case` i `switch`-ein, slik at når brukaren trykkjer på mellomromstasten (`sf::Keyboard::Space`) fyrer kanonen av, og det returnerte `Cannonball`-objektet blir lagt til i `vector`-en.

*Merk at dette ikkje vil kompilere enno, då vi ikkje har laga konstruktøren til `Cannonball`.*

# 6 Cannonball (20%)

**a) Implementer konstruktøren til `Cannonball`.**

Pass på at du får initialisert alle medlemsvariablane!

*Hint:*

- *Du kan bruke `getVelocityVector` frå `utilities.h`.*
- *Konstruktøren til `sf::CircleShape` tek berre eitt argument - radiusen (`size`).*
- *Det kan vere lurt å kalle `shape.setOrigin(size, size)` for å sette referansepunktet i senteret til sirkelen.*
- *Hugs å gi `shape` ein farge.*

Programmet ditt skal no kompilere. Du vil likevel ikkje kunne skyte kanonkuler enno, då vi ikkje har implementert `draw` og `update`. Sørg for at programmet ditt kompilerer før du går vidare.

**b) Implementer medlemsfunksjonane `getPosx` og `getPosY`.**

Desse skal returnere posisjonen i x- og y-retning, ved eit gitt tidspunkt $t$.

Bruk `getAirTime` (som er allereie er implementert) til å finne tida kanonkula har vore i lufta.

*Hint: Du har allereie implementert funksjonar som kan gjere dette i `cannonball.h`.*

**c) Implementer medlemsfunksjonane `update` og `draw`.**

`update` treng berre å oppdatere posisjonen til `shape` vha. `setPosition`.

Kall `update` og `draw` på alle kanonkuler (på korrekt stad) i `main`.

Du skal no ha ein fungerande implementasjon. Test det følgjande før du går vidare:

- Du kan justere vinkelen på kanonen, og kula blir skoten ut i rett vinkel.
- Du kan justere farten som kanonkuler blir skotne ut i.

**d) Lag ei løkke i `main` som fjernar alle kanonkulene som har landa frå `cannonballs`.**

Vi gjer dette for å unngå å gjere utrekningar på desse, etter at kanonkule har landa.

*Hint: Bruk medlemsfunksjonen `hasLanded`.*

**e) I `main`, sjekk om spelaren har treft målet. Dersom det er tilfellet, set `gameOver` lik `true`.**

Du kan bruke funksjonen `hitTarget` for å sjekke om eit `Cannonball`-objekt har kollidert med ein `Target`-objekt.

Test applikasjonen. Dersom du har fulgd øvinga skal kanonkulene stå stille etter at ei kanonkule har treft målet, og det skal heller ikkje vere mogleg å justere vinkelen på kanonen etter at spelet er over.