

## Klasseinndeling

Vi skal lage punkter, linjer og mangekanter i 2D og 3D. Disse objektene har forskjellig antall koordinater i 2D og 3D, men klassene skal ellers oppføre seg (parvis) ganske så likt.

Jeg kan lage en Punkt-klasse som har ansvaret for både 2D-punkter og 3D-punkter. Da får jeg ulogisk inndelt, og ulesbar kode hvor oppførselen til objektet avhenger av et internt flagg (f.eks. en boolean eller antall koordinater i en liste). Grøss.

Alternativt kan jeg lage en Punkt\_2D- og en Punkt\_3D-klasse. Logisk inndelt og lesbart, men en del gjentakelse av kode. All grunnleggende funksjonalitet er jo den samme, forskjellen er bare antall koordinater.

Mon tro om den beste løsningen vil være å lage en Punkt-klasse med subklasser for Punkt\_2D og Punkt\_3D? Kanskje jeg kan ha individuelle konstruktører med henholdsvis 2 og 3 standardparametre, men samtidig ha felles operatører som går igjennom koordinatene (2 eller 3) og behandler dem likt? Jeg velger nå å lage en Punkt\_2D-klasse og en Punkt\_3D-klasse osv. og evt. refaktorere i neste øving.

Klassene mine blir da foreløpig: *Point\_2D*, *Point\_3D*, *Line\_2D*, *Line\_3D*, *Polygon\_2D*, *Polygon\_3D*.

## Konstruktører

Passer på å ha fornuftige standardparametere i konstruktørene, slik at det kan opprettes punkter, linjer og mangekanter hvor koordinatene som ikke oppgis i konstruktøren er 0. F.eks. vil "Point\_3D p(1, 1);" gi et punkt med koordinater (1, 1, 0). "Polygon\_3D poly;" gir en (kollapset) mangekant med tre punkter som alle har koordinater (0, 0, 0). Jeg tilbyr også konstruktører som kan bygge et objekt av to mindre objekter.

## Konstruktøroversikt

- Point\_2D
  - Point\_2D(const int x = 0, const int y = 0);
- Point\_3D
  - Point\_3D(const int x = 0, const int y = 0, const int z = 0);
- Line\_2D
  - Line\_2D(const Point\_2D &p1 = Point\_2D(), const Point\_2D &p2 = Point\_2D());
- Line\_3D
  - Line\_3D(const Point\_3D &p1 = Point\_3D(), const Point\_3D &p2 = Point\_3D());
- Polygon\_2D
  - Polygon\_2D();
  - Polygon\_2D(const Line\_2D &, const Point\_2D &);
  - Polygon\_2D(const Point\_2D &, const Line\_2D &);
  - Polygon\_2D(const Line\_2D &, const Line\_2D &);
- Polygon\_3D
  - Polygon\_3D();
  - Polygon\_3D(const Line\_3D &, const Point\_3D &);
  - Polygon\_3D(const Point\_3D &, const Line\_3D &);
  - Polygon\_3D(const Line\_3D &, const Line\_3D &);

## Innkapsling

Alle felter deklarerer som private. Jeg får da beskyttet dataene i klassen min (andre kan ikke lese/skrive uten at jeg har laget gettere/settere). Jeg sikrer også at andre klasser ikke utvikler avhengigheter med hensyn på mine feltnavn. Jeg kan også da teste hver klasse i isolasjon.

## Filinddeling

Jeg deler inn applikasjonen på vanlig måte hvor klassene splittes i header (.h) og implementasjon (.cpp). *main()* ligger i main.cpp. Jeg har med en "#ifndef, #define, #endif"-konstruksjon i hver headerfil, slik at de kan inkluderes flere steder uten å utføres flere ganger.

## Operatorer

Jeg overrider "<<"-operatorer hvor den har en std::ostream på venstresiden, og punkt, linje eller polygon på høyresiden. Passer på at den returnerer referanse til en std::ostream, slik at jeg kan hekte flere på hverandre.

Jeg overrider "+"-operatorer slik at man kan legge sammen to punkter til en linje, en linje og et punkt (og omvendt) til et polygon, to linjer til et polygon, polygon og punkt (og omvendt) til polygon, og polygon og linje (og omvendt) til polygon. Det virker fornuftig at "+"-operasjoner returnerer et nytt objekt, siden programmeren muligens ønsker å bruke parameterene videre.

Jeg overrider "+="-operatorer slik at man kan utvide et polygon med et punkt, linje eller polygon. Det virker fornuftig at "+="-operasjonene modifierer objektet på venstresiden. Jeg returnerer en referanse, isteden for å kopiere objektet.

## Operatoroversikt

- Point\_2D
  - std::ostream &operator<<(std::ostream &, const Point\_2D &);
- Point\_3D
  - std::ostream &operator<<(std::ostream &, const Point\_3D &);
- Line\_2D
  - Line\_2D operator+(const Point\_2D &, const Point\_2D &);
  - std::ostream &operator<<(std::ostream &, const Line\_2D &);
- Line\_3D
  - Line\_3D operator+(const Point\_3D &, const Point\_3D &);
  - std::ostream &operator<<(std::ostream &, const Line\_3D &);
- Polygon\_2D
  - Polygon\_2D operator+(const Point\_2D &);
  - Polygon\_2D operator+(const Line\_2D &);
  - Polygon\_2D operator+(const Polygon\_2D &);
  - Polygon\_2D &operator+=(const Point\_2D &);
  - Polygon\_2D &operator+=(const Line\_2D &);
  - Polygon\_2D &operator+=(const Polygon\_2D &);
  - Polygon\_2D operator+(const Line\_2D &, const Point\_2D &);
  - Polygon\_2D operator+(const Point\_2D &, const Line\_2D &);
  - Polygon\_2D operator+(const Line\_2D &, const Line\_2D &);
  - Polygon\_2D operator+(const Point\_2D &, const Polygon\_2D &);
  - Polygon\_2D operator+(const Line\_2D &, const Polygon\_2D &);
  - std::ostream &operator<<(std::ostream &, const Polygon\_2D &);
- Polygon\_3D
  - Polygon\_3D operator+(const Point\_3D &);
  - Polygon\_3D operator+(const Line\_3D &);
  - Polygon\_3D operator+(const Polygon\_3D &);
  - Polygon\_3D &operator+=(const Point\_3D &);
  - Polygon\_3D &operator+=(const Line\_3D &);
  - Polygon\_3D &operator+=(const Polygon\_3D &);
  - Polygon\_3D operator+(const Line\_3D &, const Point\_3D &);
  - Polygon\_3D operator+(const Point\_3D &, const Line\_3D &);
  - Polygon\_3D operator+(const Line\_3D &, const Line\_3D &);
  - Polygon\_3D operator+(const Point\_3D &, const Polygon\_3D &);
  - Polygon\_3D operator+(const Line\_3D &, const Polygon\_3D &);
  - std::ostream &operator<<(std::ostream &, const Polygon\_3D &);

## const correctness

Jeg markerer konstante parametere med *const* i tråd med prinsippet om å gi kompileren så mye informasjon som mulig. Jeg sikrer slik at verdier som ikke skal endres, ikke blir endret. Man vil få en kompilerfeil dersom noen forsøker å endre verdien. Jeg får også ørlite bedre ytelse. Metoder som ikke forandrer objektet sitt, markeres også med *const* for å signalisere til utviklern(e) at de ikke endrer objektet (og for å få en error dersom objektet blir forandret).

## Metoder markert som const

- Point\_2D
  - `int` getX() `const`;
  - `int` getY() `const`;
- Point\_3D
  - `int` getX() `const`;
  - `int` getY() `const`;
  - `int` getZ() `const`;
- Line\_2D
  - Point\_2D getStartPoint() `const`;
  - Point\_2D getEndPoint() `const`;
- Line\_3D
  - Point\_3D getStartPoint() `const`;
  - Point\_3D getEndPoint() `const`;
- Polygon\_2D
  - `std::list<Point_2D>` getPoints() `const`;
- Polygon\_3D
  - `std::list<Point_3D>` getPoints() `const`;