TDP004 Objektorienterad Programmering
Fö 6 Objektorientering forts.

Introduktion

- Viktiga objektorienterade begrepp:
  - Superklass, subklass, arv.
  - Abstrakta klasser.
  - Virtuella funktioner.
  - Polymorfi.
  - Static.
- Laboration på objektorientering.

Repetition från fö 2

I filen ExampleClass.h:

```
class ExampleClass
{
public:                                 // Publik åtkomstdeklaration
    ExampleClass(int defaultID);        // Konstruktor
    ~ExampleClass();                    // Destruktor
    bool SetID(int aID);                // Medlemsfunktioner.
    const int GetID() const;
    float f(float a, float b, float c) const;
private:                                // Privat åtkomstdeklaration
    int m_ID;                           // Klassvariabel.
    …
};                                      // Observera ;
```

För det mesta definieras funktionerna i .cc/.cpp-filen.

Repetition från fö 2, forts.

- Med åtkomstdeklarationerna public, protected och private reglerar man åtkomst till klassens data och funktioner för *andra* klasser. Den egna klassen har alltid obegränsad tillgång.
- Public – fritt tillgänglig för läs och skriv.
- Private – ingen tillgänglighet alls.
- Protected – tillgänglig enbart för subklasser.
- I class är allt private som default.
- I struct är allt public som default.

Arv

- Inom OO är arv ett viktigt begrepp: vi kan se arv som specialisering/förfining av en klass och dess funktioner.
- Klassen i toppen av arvshierarkin kallas superklass/basklass.
- Den klass som ärver av superklassen kallas subklass.
- Arvshierarkier ritas tex med UML.

Arv, forts.

- En subklass är en specialisering av en superklass.
- Åtkomsten av en superklass funktioner och data varierar beroende på vilken sorts arv det är.
- I C++ finns det public, protected och private arv.
- Data/funktioner som är private i basklassen ärvs aldrig trots att det finns private arv.

Olika sorters arv

- public: de ärvda funktionerna/datan behåller den åtkomst de hade i superklassen.
- protected: de ärvda funktionerna/datan blir protected i subklassen.
- private: de ärvda funktionerna/datan blir private i subklassen.
- Den ärvande klassen har alltså möjlighet att styra hur andra klasser ska komma åt de ärvda funktionerna.
- Man inte kan göra funktioner/data mera tillgängliga genom arv.

Virtuell funktion

- Hittills har vi alltid implementerat alla funktioner i klassen.
- Vi kan även deklarera virtuella funktioner, dessa har nyckelordet virtual framför returvärdet:
  virtual double GetSpeed();
- En virtuell funktion är ett kontrakt som alla subklasser måste uppfylla.
- En superklass som definierar en virtuell funktion *kan* själv implementera den (i C++, men inte i alla andra OO språk).
- Nyckelordet virtual skrivs enbart i .h-filen.

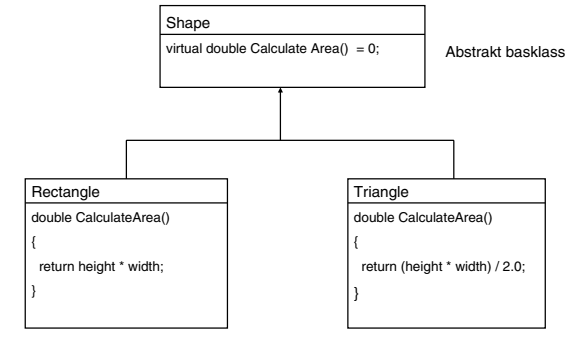
Virtuella funktioner, forts.

- Vad är poängen med virtuella funktioner?
- *Att kunna delegera utförandet av en viss beräkning till subklasserna och inte bry sig om hur de utför den.*
- Detta koncept kallas *polymorfism* (mångformighet) och en sådan funktion kan definieras på olika sätt.

Rent virtuell funktion

- En rent virtuell funktion får inte definieras i den deklarerande klassen:

  virtual double GetSpeed() = 0;
- = 0 skrivs enbart i .h-filen.
- Detta gör det omöjligt att skapa en instans av klassen: det blir en abstrakt klass.
- Alla rent virtuella funktioner måste implementeras i subklasserna.



Enkelt klassdiagram. Subklasserna definierar funktionen CalculateArea() på olika sätt.

Konstruktorer etc utelämnade.

Polymorfism

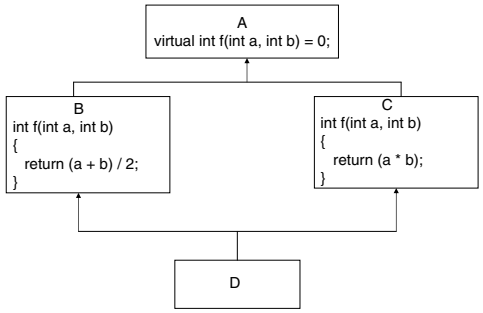
```
std::vector<Shape*> shapesVector;
Shape* shape1 = new Triangle(…);
Shape* shape2 = new Rectangle(…);
shapesVector.push_back(shape1);
shapesVector.push_back(shape2);
shapesVector.push_back(…);
Ex 1:
for(unsigned int i = 0; i < shapesVector.size() ; ++i)
{
    std::cout<< shapesVector[i]->CalculateArea() << " ";
    /* Anropar korrekt CalculateArea(), vi vet
    inte vilken implementation innan anropet sker. */
}

Ex 2:
shape1->CalculateArea();
    //Anropar CalculateArea() i klassen Triangle.
shape2->CalculateArea();
    //Anropar CalculateArea() i klassen Rectangle.
```

Polymorfism, forts.

- Vilken funktion som exekveras vid anrop till CalculateArea() avgörs vid exekveringen, sk dynamisk/sen bindning.
- En icke-virtuell funktion har sk statisk/tidig bindning: det bestäms vid kompilering vilken funktion som kommer att anropas.

Multipelt arv

- Det är fullt möjligt att ärva från flera klasser.
- Deklaration som vanligt.
- Man får se upp om flera basklasser i sin tur är subklasser till samma basklass, och implementerar virtuella funktioner olika.
- Detta problem löses på olika sätt i olika OO-språk.
- Exempel på nästa sida.



D* d = new D();

d->f(1, 3);?

Vad händer då funktionen f anropas i klassen D?

Detta utseende i arvshierarkin kallas för "The diamond pattern".

Virtuella destruktorer

- En virtuell destruktor i basklassen behövs vid arv, för att säkerställa att allt minne i basklassen lämnas tillbaka.
- En basklass bör definiera en virtuell destruktor även om den inte har något att ta bort.
- En kompilatorgenererad destruktor är *aldrig* virtuell.
- Det är ingen nackdel att göra *alla* destruktorer virtuella.

Static

- En static medlems-/variabel-funktion finns bara i ett exemplar som hör till *klassen*, inte till någon instans av klassen. Användbart när man vill ha någonting som det bara finns en av och som alla instanser kan använda.
- Ex: alla instanser av en klass ska ha ett unikt ID-nummer.

Static variabel

- I .h-filen

```
static int ourID;          // Static variabel.
int m_ID;                  // Vanlig medlemsvariabel.
```

- I .cc-filen

```
/* En static variabel måste initieras utanför funktionerna. */
int classname::ourID = 1;
// Men kan användas i alla klassens funktioner.
classname:: classname()
{
  m_ID = ourID;            // Tilldela ett unikt ID.
  ourID++;
}
```

Static funktioner

- Även funktioner kan vara static.
- *En static funktion kan anropas även om det inte finns några instanser av klassen.*
- Får endast använda statiska klassvariabler (lokala variabler som vanligt).
- Kan endast anropa andra funktioner som inte tar this-pekare som parameter (static-funktioner och konstruktor).

Lokala static variabler

- Även lokala varibler kan vara static.
- Ex:
  - Vi vill igen ge alla instanser ett unikt ID-nummer.

```
classname:: classname()
{
    static int ourID = 0;   // Exekveras endast en gång.
    m_ID = ourID;           // Tilldela ett unikt ID.
    ourID++;
}
```

  - Nackdelen jämfört med förra metoden är att vi inte har tillgång till ourID utanför konstruktorn, om vi tex skulle vilja få reda på det nuvarande värdet på ourID.

Copy constructor och tilldelningsoperator

- Copy constructor för att korrekt kopiera instanser.
- Kompilatorns copy constructor är public som default.
- ExampleClass(const ExampleClass&);
- Skapas av kompilatorn om inte programmeraren gör det.

Copy constructor och tilldelningsoperator, forts.

- Tilldelningsoperator för att tilldela en instans.
- ExampleClass& operator=(const ExampleClass&);
- Defaultbeteende för den automatgenererade är bitvis kopiering av data.

Exempel där tilldelningsoperator behövs

```
class Athlete
{
   public:
    Athlete(const char* value)
    ~Athlete();
   private:
   char* name;
};
```

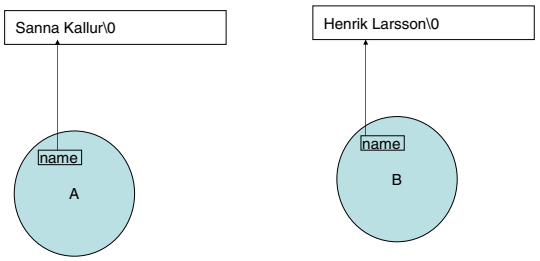
Exempel, implementation

```
Athlete::Athlete(const char* value)
{
     if(NULL != value)
     {
        name = new char[strlen(value) +1];
        strcpy(name, value);
     }
     else
     {
        name = new char[1];
        name= '\0';
     }
}
Athlete::~Athlete()
{
    delete [] name;
}
```

Exempel, användning

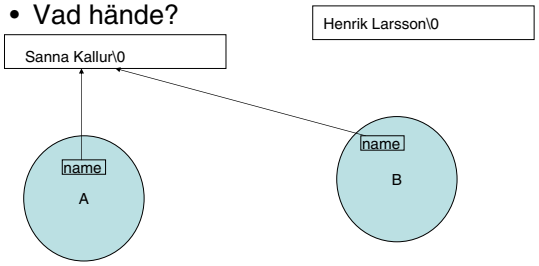
Athlete* A = new Athlete(”Sanna Kallur”);

Athlete* B = new Athlete(”Henrik Larsson”);



Exempel, tilldelning

- B=A;
- Vad hände?



Exempel, avslutning

- Problem
  - 1.Minnesläcka. Urprungligt minne för variabeln ”name” för instans B (”Henrik Larssson\0”) kommer inte att tas bort.
  - 2. Antag att destruktorn för instans A körs och name tas bort. Vad händer när konstruktorn för instans B körs?
- Lösning
  - Implementera tilldelningsoperator.

Exempel där copy constructor behövs

```
void doNothing(Athlete a)
{
   //Nothing is performed.
}
Athlete C(”Mats Sundin”);
doNothing(C);
```

Ingen copy constructor finns.
Vad händer med C när doNothing returnerar?
Lösningen är att implementera copy constructor.

Sammanfattning

- Arv, superklass och subklass.
- Public, protected och private arv.
- Multipelt arv.
- (Rent) virtuella funktioner, abstrakta klasser.
- Polymorfi.
- Dynamisk och statisk bindning.
- Nyckelordet static för funktioner och variabler.