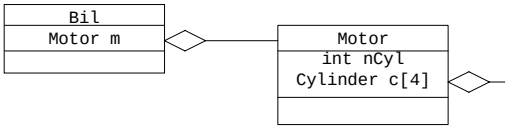
Mer relationer, dynamisk bindning och polymorfism

- Aggregat
- Association
- Polymorfism
- Dynamisk bindning
- Programexempel

Aggregat

- Aggregat är en 'består-av'-relation



- Ett aggregat går inte att dela upp, och delarna skapas/tas bort samtidigt som den 'ägande' klassen

Exempel

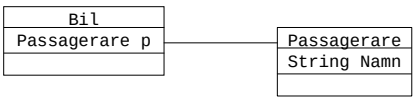
```
#include <iostream.h>
#include <string.h>

class Motor
{
  private:
    int nCyl;
  public:
    Motor ()
    { cout << "Motor konstrueras" << endl; }
    ~Motor ()
    { cout << "Motor destrueras" << endl; }
};
```


Exempel forts.

```
class Bil
{
  private:
    Motor m;
  public:
    Bil ()
    { cout << "Bil konstrueras" << endl; }
    ~Bil ()
    { cout << "Bil destrueras" << endl; }
};


int main()
{
  Bil b;
  cout << "klart !" << endl;
}
```

Association

- Association är en 'har'- eller 'känner-till'-relation



- Implementeras med pekare i C++
- De associerade objekten måste skapas 'manuellt'

Exempel

```
#include <iostream.h>
#include <string.h>

class Hjul
{
  private:
    int hjulNr;
  public:
    Hjul ()
    { cout << "Hjul konstrueras" << endl; }
    ~Hjul ()
    { cout << "Hjul destrueras" << endl; }
};
```

```
class Bil
{ private:
    Hjul *h[4];
  public:
    Bil ()
    { cout << "Bil konstrueras" << endl;
      for (int i = 0; i < 4; i++)
      { h[i] = new Hjul; }
    }
     ~Bil ()
     { cout << "Bil destrueras" << endl;
       for (int i = 0; i < 4; i++)
       { delete h[i]; }
     }
};

int main()
{
  Bil b;
  cout << "klart !" << endl;
}
```

Dynamisk bindning och polymorfism

- I C++ är pekare till basklasser *polymorfa*, dvs de kan peka på objekt av en subklass typ
- Vid statisk bindning sker all bindning vid kompileringen ->
  - Vid ett metodanrop är det den polymorfa pekarens **typ** som avgör vilken metod som avses
- Vid dynamisk bindning sker bindningen inte förrän programmet körs ->
  - Vid ett metodanrop är det typen på det som pekaren **pekar på** som avgör vilken metod som avses

Virtuella funktioner

- Statisk bindning är default - nyckelordet *virtual* används för att få dynamisk binding

```
class Figur
{
  public:
    virtual double getArea();
};
class Rekt:public Figur
{
  private:
    double b,h;
  public:
    double getArea() { return b*h; } //virtuell
};
```

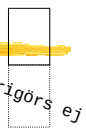

Virtuella anrop med pekare

```
class Cirk:public Figur
{
  private:
    double r;
  public:
    double getArea() { return M_PI*r*r; }
};

Figur *fp[2];
fp[0] = new Cirk(5);
fp[1] = new Rekt(3,4);

for (int i=0;i<2;i++)
  cout << ”Area ” << fp[i]->getArea();
```


Virtuella anrop med referens

```
void skrivUtArea(Figur& f)
{
  cout << ”Arean är ” << f.getArea() << ”.”;
}
Cirk jorden(40000000/M_PI/2);
skrivUtArea(jorden);
```


Virtuell destruktor



- Basklasssens destruktor anropas

```
for (int i=0;i<2;i++)
  delete fp[i]; // Frigör bara en figur
```

- Aktuella klassens destruktor anropas om den är virtuell

```
class Figur
{
  virtual ~Figur();
}
```