

Krav

- Tre krav måste vara uppfyllda:
 - Abstrakt datatyp
 - Arv
 - Polymorfism och dynamisk bindning
- Den abstrakta datatypen kallas ofta klass
 - Data kallas **attribut**, funktionaliteten finns i **metoder**
 - Anrop av en metod kallas att skicka ett **meddelande** till ett objekt

63

Inkapsling

- Inkapsling brukar också tas upp som ett krav på ett objektorienterat språk
- C++ har tre nivåer av inkapsling
 - private
 - protected
 - public
- Java har även 'package scope'

64

Tillbaka till 80-talet

- Problem med objektbaserade språk
 - Abstrakta datatyperna borde kunna återanvändas
 - Alla typdefinitioner oberoende och på samma nivå
- Lösningen heter **arv**
 - Data och funktionalitet kan ärvas och modifieras
 - Återanvändning utan ändring i redan skriven kod

65

Återanvändning

- Inom systemet
- Av generella bibliotek
- Applikationsspecifika
- Väldefinierade klasser
- Köpa färdigt
- Inom koncernen

66

Problem med återanvändning

- Målet ej uttalat av projektledningen
 - Avsätta resurser i planeringen
- Socialt/psykologiskt
 - Snäva tidsramar - ej generell kod
 - Använder ej annans kod
- Administrativt
 - Insamling, kvalitetsgranskning
 - Dokumentera, katalogisera, distribuera
 - Söka och finna

67

Arv ...

- En subclass ärver från en basclass (kallas också superclass)
 - Åtkomst av basclassens attribut och metoder kan styras
 - Metoder kan omdefinieras (override)
 - Två sorters metoder och två sorters attribut
 - Instansmetoder och attribut
 - Klassmetoder och attribut

68

Computing with an Object-Oriented Language

- Ett objektorienterat program som körs kan ses som en samling datorer/objekt som kommunicerar via meddelanden
- Varje objekt är en abstraktion av en dator i den mening att den lagrar data och kan manipulera det
- Objektorienterad programmering är att lösa problem genom att identifiera objekten i problemet och sedan simulera dem, dess processer och den nödvändiga kommunikationen

69

Avvägningar vid design av ett objektorienterat språk

- Tre sätt att angripa problemet med typsystemet:
 - Den 'rena' modellen
 - Allt är objekt, från det minsta heltal till det största datasystem
 - + Elegant form
 - De enklaste operationer måste ske genom metदानrop
 - Utgå från ett imperativt språk och lägg till en objektmodell
 - + Snabbare
 - Mycket större, mer komplext system

70

Tre sätt ...

- Objektmodell i botten men med 'imperativ stil' för primitiva datatyper
 - + Snabbt för heltal och andra primitiva typer
 - Komplikationer när metoder bara kan ta objekt som parametrar -> Wrapper Classes
- C++, Java hör till den senare typen
- Allt är objekt utom de primitiva datatyperna som int, double, ...
- String och fält [] är objekt i Java

71

Statisk eller dynamisk typkontroll?

- Om språket skall vara starkt typat:
 - -> statisk typkontroll
 - Restriktioner vid polymorfism
 - Två sorters typkontroll vid metदानrop:
 - Parametrarna måste stämma
 - Returtypen likaså
 - Kanske tillåtet med 'assignment compability'
- Alternativet är dynamisk typkontroll
 - Vänta med kontrollen tills metoden anropas
 - Kostsamt och försenar typkontrollen

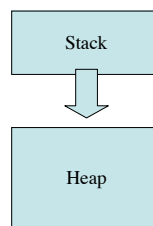
72

Allokering/deallokering av objekt

- Var allokeras objekten? Tre sätt :
 - Statiskt av compilatorn
 - Dynamiska objekt på stacken
 - Eller på heapen med new
- Om new det enda sättet -> uniform metod
 - Inga pekare som måste derefereras a la C++
- C++ har alla sätten - och alla problemen
- Java har bara new (allt på heapen)

73

Stack och heap



- Dynamiska variabler, tex parametrar, lokala variabler
- int, double, referenser
- Allt som skapas med new, dvs alla objekt, inklusive fält och strängar
- Risk för fragmentering -> Garbage Collector

74

Vi använder new, sen då?

- Hur deallokera objekten?
- ->Explicit
 - Problem men 'dangling pointers'
 - C++ har **delete**
- -> Implicit
 - Vi behöver Garbage Collection i någon form
 - Java har GC

75

Klasser och objekt i C++

- Klasser deklarerar i en typdeklaration, precis som structs i C
- Ett eller flera objekt skapas med klassen som mall
 - Det finns två sätt att skapa objekt:
 - På heapen (med new)
 - Som lokala (automatiska) variabler

76

En klass - två filer

- Ett bra sätt att dela upp sitt system
- Många filer blir det - bra att ha någon struktur på filsystemet
- Varje klass har två filer
 - Headerfil (.h) med deklARATIONER (interfacet)
 - Klassfilen (.cpp) med definitionen (implementationen)
- Det går att slå ihop dem, men det försvårar användningen av klassen, och minskar inkapslingen

77

Exempel

- Headerfil (Interface)
Bara deklARATIONER av metoder och (tyvärr) attribut

```
class Konto
{
public:
    void sattIn(int kr); //metod
private:
    int saldo;           //attribut
};
```

78

Forts.

- Klassfil (Implementation)
Definitioner av metoder
- ```
void Konto::sattIn(int kr)
{
 saldo=saldo + kr;
}
```

79

## Kombinerat interface/implementation

- Kan användas för enstaka klasser, inget att rekommendera

```
class Konto
{
public:
 void sattIn(int kr) //metod
 {
 saldo=saldo + kr;
 }
private:
 int saldo; //attribut
}
```

80

## Flera klasser

- Endast .h-filerna behöver 'synas'

```
#include "Passagerare.h"
#include "Bil.h"
Bil::gasa()...
Bil::bromsa()...

#include "Passagerare.h"
#include "Bil.h"
Passagerare::metod1()...
Passagerare::metod2()...
```

- Problem vid korsvis beroende

81

## Korsvis användning

- Två headerfiler som refererar till varandras klassdeklARATIONER:

```
#include "Passagerare.h"
class Bil
{
private:
 Passagerare *pass;
};

#include "Bil.h"
class Passagerare
{
private:
 Bil *minBil;
};

class Bil; //Def kommer
class Passagerare
{
private:
 Bil *minBil;
}
```

82

## Åtkomstnivåer

- Tre nivåer av inkapsling
  - public - åtkomst för alla
  - protected - Åtkomst för subklasser
  - private - Åtkomst endast inom klassen

```
class Bil
{
public:
 void gasa();
protected:
 int hastighet;
private:
 int modell;
};
```

83

## Åtkomst forts.

- Både attribut och metoder kan ha alla nivåer
- En nivå (ex private) gäller tills nästa anges
- Default är **private**
- Attribut bör vara private
  - Använd åtkomstmetoder för att inte bryta inkapslingen
- Metoder som endast använd inom klassen bör vara private, eller protected om det kan tänkas att subklasser kan komma att omdefiniera dem (svårt val ibland)

84

## Klassattribut

- 'Vanliga attribut hör till ett objekt, dvs flera objekt -> flera uppsättningar attribut med olika värden
- Klassattribut är gemensamma för alla objekt av en klass - 'hör till klassen'
- Nyckelordet **static** anger klassattribut/metod
- Klassmetoder kan bara modifiera klassattribut (vilket värde avses annars?)

85

## Exempel

- Räkna för antalet skapade objekt

```
class Bil
{
public:
 Bil() {
 antal++;
 }
 int getantal() {
 return antal;
 }
private:
 static int antal;
};
```

86