TDA 545: Objektorienterad programmering

Föreläsning 2:

Typer, klasser, tilldelning (assignment)

Magnus Myréen

Chalmers, läsperiod 1, 2015-2016

Läsanvisning

Dessa *slides* samt kapitel 1

- kort inledning till klasser och objekt,
- identifierare (namn på saker),
- litteraler (konstanta värden)
- typer, variabler och tilldelning
- primitiva typer och klasser
- fördefinierade primitiva typer
- uttryck, operatorer, prioritet, typomvandling, Javas API, strängar, wrappers

För nästa gång: läs kapitel 3 (val/beslut) och 4 (iteration)

Klasser och Objekt

Den verkliga världen
består av "saker"

Modellen beskrivs med
klasser (och variabler)

När programmet körs
skapas objekt
= instanser av klasserna

En trafikkorsning
bil1, bil2, bil3 trafikant1, trafikant2 trafikljus1, trafikljus2
class Bil class Trafikant class Trafikljus class Trafikkorsning
Bil bil1 Bil bil2 Trafikant magnus, lena Trafikljus ljus1, ljus2

Objekt är en viktig "nedbrytningskomponent". Varje objekt tillhör en klass som definierar dess beteende och tillstånd.

Objekt representerar "verkliga" saker

Varje objekt har:

Identitet: representeras med namnet	bil1
Tillstånd: representeras med instansvariabler och deras värden	color ”red” speed 55km/h
Beteende/funktionalitet: representeras med instansmetoderna	incSpeed(...) setColor(...) ...

Klassen används för att skapa objekt dvs en klass definierar hur ett visst objekt "ser ut," dess data och funktionalitet.

Objektet hör till en klass och sägs vara en ***instans*** av den klassen. Klassen är objektets **typ**.

Objektets Beteende/Funktionalitet =
de åtgärder objektet är berett att utföra.

Tillståndet (state) behövs för att stödja funktionaliteten.

Beteende/Funktionalitet

Två typer av beteende:

En fråga:

Ändrar inte objektets tillstånd
kan vara enkel tex vad är värdet av fältet "x"
eller mer komplexa tex räkna ut arean (utgående från värdet av instansvariablerna längd och bredd)

Ett kommando som ändrar tillståndet:

Det är alltid objektet självt som ändrar sitt tillstånd.

Ibland är objekt "***immutable***".

Då kan dom inte ändras när dom väl fått ett värde.
Gäller tex för objekt av typen String.

Senare definierar vi (fördjupa och repetera för en del) en del grundläggande begrepp som identifierare, litteral, variabel, typ mm

Identifierare

En identifierare är ett namn på saker som variabler, metoder, klasser, mm

Regler för identifierare i Java:

- Skall vara en bokstav följd av noll eller flera tecken där tecken får vara bokstav, siffra eller "_". (Undvik dock å, ä, ö m.fl.)
- Små och stora bokstäver tolkas **olika**.
  Odd är alltså inte samma namn som **odd**

Konventioner om identifierare i Java:

- Klassnamn börjar alltid på stor bokstav
- Variabler och metoder börjar på liten
- flera ord sätts samman med stora bokstäver inuti ordet tex StringTokenizer, `skickaBlommor, nbrOfFingers,`
- konstanter skrivs med STORA bokstäver tex `PI, PLANKS_KONSTANT`

Råd om identifierare i Java:

- **Skall vara beskrivande** dvs namnet skall avslöja (för andra) vad namnet står för. (Beskrivande är inte lika med långa!)

… beskrivande namn

Beskrivande namn innebär också tex:

- **Använd förkortningar försiktigt**
  Vad är en `termProcess()`?
  - termometer process, (svenska)
  - terminate process,
  - terminal process?

- **Undvik namn som kan betyda flera saker**
  - Vad gör `empty()`?
    tömmer något? kalla den `makeEmpty()`
    frågar om något är tomt? -> `isEmpty()`
  - `isPrinterReady` är bättre än tex
    `printerStatus`

Typer

En typ är (en beskrivning av) ett eller flera relaterade värden tillsammans med operationer på dessa värden

Vi behöver kunna modellera verkligheten och abstrahera bort dess detaljer.

Verkliga "saker" har olika egenskaper - olika dataobjekt har därför olika "typ".

- modellerar verkligheten
- definierar egenskaper dvs vad man kan göra
- definierar hur bitmönstret skall tolkas (kompilatorn kan kolla tillåtna operationer)

Variabler

En variabel är en minnesplats med namn, typ och innehåll

Den kan bara innehålla värden som har samma typ som den själv.

tilldelning (assignment)

```
int nbrOfFingers = 10;
```

typen bestämmer variabelns egenskaper, dvs vad man kan göra med variabeln, den är en "mall" för hur en variabel/objekt "ser ut"

variabler har ***namn*** som är en identifierare

OBS att "=" är inte likhet utan tilldelning
läses: nbrOfFingers får värdet 10

2 olika slags typer i Java

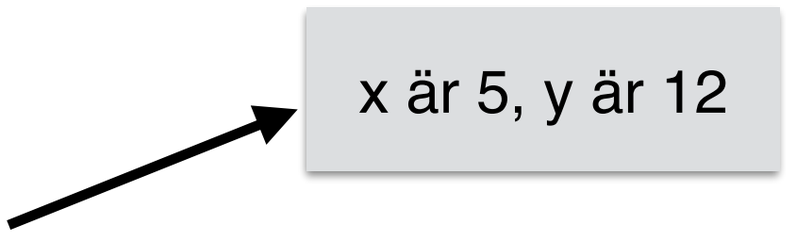
Primitiva typer (int, double, char, boolean, ...)

- enkla odelbara värden som ett tal eller ett tecken
- endast fördefinierade operatorer, tex '+', '/', inga metoder finns.
- finns fördefinierade i språket

```
int nbrOfFingers = 10;
```

Klasstyper
(eller referenstyper eller sammansatta typer)

- vanligen komplexa värden uppbyggda av flera primitiva typer och/eller klasstyper tex `String, Rectangle, Boll`
- definieras av användaren
- metoder som man skriver själv
- alltså: komplexa data och metoder




```
Boll b = new Boll(5,12);
```

Likheter och skillnader mellan primitiv typ och klasstyp

```
int nbr = 10;
```

här får **nbr** värdet 10 (vilket är typ **int**)

nbr är 10

```
Boll boll = new Boll(5, 12);
```

här får **boll** ett *referens* värde, dvs adressen på objektet som **new Boll(5, 12)** producerar.

boll är

x är 5, y är 12

int är en primitiv typ och variabeln **nbr** tillhör den typen.

Boll är en klasstyp och variabeln **boll** pekar på (refererar till) en instans av den klassen.

Fördefinierade primitiva typer

Type	Values	Representation
boolean	false, true	byte
char	from '\u0000' to '\uffff' inclusive, that is, from 0 to 65535	16-bit unsigned integer
byte	from -128 to 127, inclusive	8-bit twos complement signed integer
short	from -32768 to 32767, inclusive	16-bit twos complement s.i.
int	From -2 147 483 648 to 2 147 483 647, inclusive	32-bit twos complement signed integer
long	-9223372036854775808 to 9223372036854775807, inclusive	64-bit twos complement signed integer
float	from $\pm 3.40282347 \times 10^{38}$ to $\pm 1.40239846 \times 10^{-45}$	32-bit single-precision flo.p
double	$\pm 1.7976931348623157 \times 10^{308}$ to $\pm 4.9406564581246544 \times 10^{-324}$	64-bit double-precision floating poin

Deklarationer av variabler, objekt och konstanter

Alla variabler måste deklareras.

Primitiva variabler

deklaration

Variablerna kan initialiseras vid deklarationen.

```
int thisMonth;
int thisYear = 1987;
int nextYear = thisYear + 1;
double moms = 25.0;
```

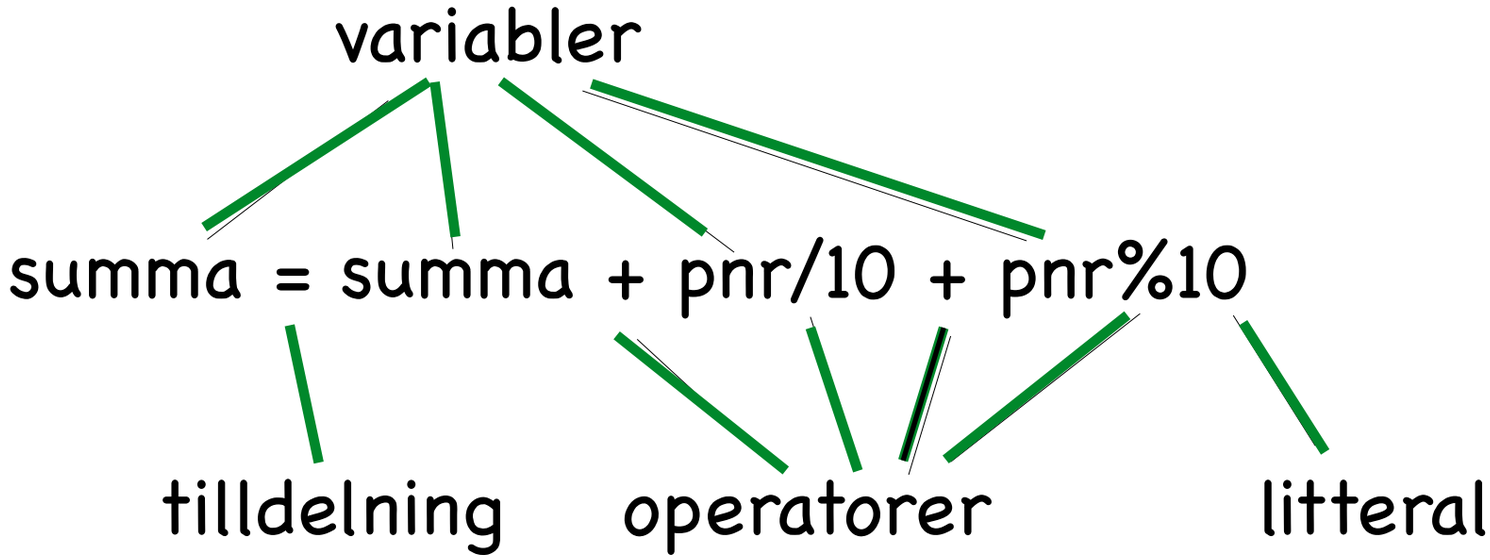
Objekt

```
Boll boll = new Boll();
Boll boll2;
boll2 = new Boll();
```

Konstanter

```
static final int MILLENNIUM = 2000;
static final char CAPA = 'A';
static final float TAL = n + 3.15;
static final double PI = 3.141593;
```

Aritmetiska uttryck, tilldelning mm.



Aritmetiska uttryck byggs upp utgående från litteraler och variabler och genom att deluttryck sätts samman med hjälp av operatorer tex *, +, /, %, ...

prioritet: ! (negering),...., *, /, % , ..., +, -

Aritmetiska uttryck, tilldelning mm.

varje uttryck får en typ som bestäms av de ingående litteralernas och variablernas typer

9 + 2/3	int	≡ 9
9.0+2/3	double	≡ 9.0
9 + 2/3.0	double	≡ 9.6...
9 + 2.0/3.0	double	≡ 9.6...
9 + (double)2/3	typomvandling (typecast)	≡ 9.6...

Fördefinierade klasstyper i Java?

Det finns inga sådana men ...

I Java distributionen får man med ett API (Application Programmers Interface) som är en samling *paket* som innehåller massor av användbara klasser.

(*Paket* = samling med klasser)

En del är hårdare knutna till språket tex de som finns i *paketet* java.lang som alltid finns tillgängligt och en del är mindre hårt knutna och man måste explicit säga till att man vill komma åt dem.

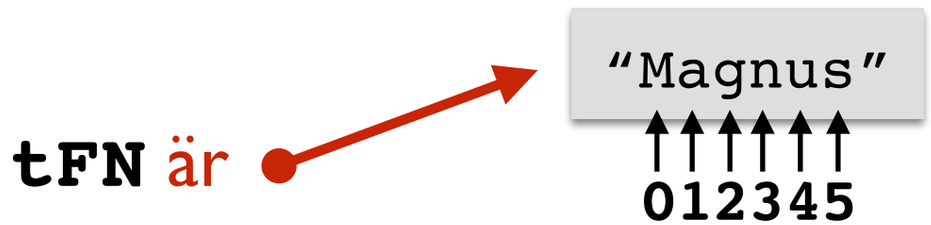
Paketet java.lang innehåller tex

- alla wrapper klasser som Integer, Double, …
- String och StringBuilder
- Object och Math
- System (som i System.out.print….)

Ex på en klass i Javas API

Det finns ingen primitiv strängtyp men det finns en klass för strängar: String

```
String teachersFirstName = "Magnus";
String tFN = new String("Magnus");
```




```
System.out.println("Magnus");
System.out.println(teachersFirstName);
String teachersFullName;
teachersFullName = teachersFirstName + " Myréen";

String tmp = "10" + 5; // ger "105"
// automatisk konvertering!
```

Det finns många metoder för att manipulera strängar i java.lang.String. Deras specifikationer beskrivs i APIn.

Hur hittar man vad som finns?

Man tittar i beskrivningen av Javas API

String (Java Platform SE 7) × TDA 545: Objektorienterad ×

docs.oracle.com/javase/7/docs/api/

Java™ Platform Standard Ed. 7

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd

Double
Enum
Float
InheritableThreadLocal
Integer
Long
Math
Number
Object
Package
Process
ProcessBuilder
ProcessBuilder.Redirect
Runtime
RuntimePermission
SecurityManager

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
char charAt(int index)
```
Returns the character at the specified index.
```
String str = "Magnus";
char c = str.charAt(1);
```

```
int compareTo(String anotherString)
```
Compares two strings lexicographically.

```
boolean equals(Object anObject)
```
Compares this string to the specified object.

```
int indexOf(int ch)
```
Returns the index within this string of the first occurrence of the specified character.

```
int length()
```
Returns the length of this string.

```
String substring(int beginIndex, int endIndex)
```
Returns a new string that is a substring of this string.

```
static String valueOf(double d)
```
Returns the string representation of the double argument.

Strängar

Man anropar en metod i klassen String genom

- att ange objektet som skall utföra metoden,
- en punkt,
- och sedan metodnamnet

Exempel

```
String teachersFirstName = "Magnus";

teachersFirstName.charAt(0) → 'M'
teachersFirstName.charAt(3) → 'n'
teachersFirstName.compareTo("Java") → -1
rsFirstName.indexOf('a') → 1
sFirstName.length() → 6
FirstName.substring(0,2) → "Ma"
String.valueOf(12.345) → "12.345"
```

OBS '→' är inte Java syntax, betyder här 'blir'

Den sista är en statisk metod

Mera om strängar

Operatorn + (konkatenering) kan användas på objekt av typen String.

Exempel

```
String str = "Hej";

str = str + " på dig!";
```

skapar en ny sträng “Hej på dig!”

Wrappers

Man vill ibland använda ett primitivt värde som om det vore ett objekt och därför finns det i Javas API en *wrapper*-klass för varje primitiv typ

char – Character, int – Integer, osv

som "wrappar" ett primitivt objekt i en klass.

Det ger vissa fördelar

- klasserna kan tex innehålla många metoder utöver de operatorer som finns
- när man vill lägga in ett värde av en primitiv typ i en av Collection klasserna tex ArrayList eller HashTable så måste man använda ett värde av motsvarande wrapperklass

Det innebär också vissa nackdelar: de tar större plats och omvandlingen mellan primitivt värde och klassvärde tar tid.

Wrapper klasserna är *immutable*, dvs deras värden går inte att ändra (som String)

Wrapper klassen Integer innehåller bland annat:

Konstruktorer

```
Integer(int value)
Integer(String value)
```

Några metoder

```
int intValue()
long longValue()
String toString()

String Integer.toString(int value)
Integer Integer.valueOf(String value)
int Integer.parseInt(String value)
int Integer.MAX_VALUE
```

Ett program

```
import javax.swing.*;
public class Namn {
  public static void main(String[] args) {
    String namn;
    namn = JOptionPane.showInputDialog("Vad heter du?");
    System.out.println(); // tom rad
    System.out.println( namn.toUpperCase() );
    namn = "123";
    System.out.println( Integer.parseInt(namn) );
  } // end main
} // end Namn
```

Kommentarer

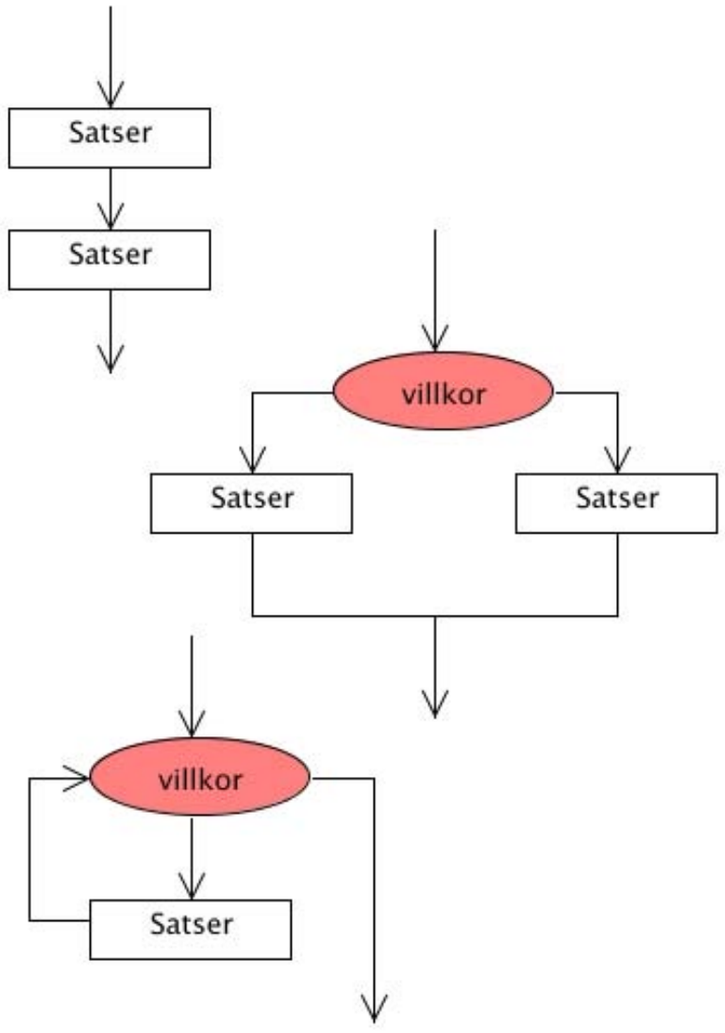
Tre sorters kommentarer i Java.

```
// allt till slutet av raden


/*
allt som står mellan
*/


/**
********** ********** **********
* Javadoc kommentar
* allt som står mellan
********** ********** **********
*/
```

Nästa gång: Primitiver

Det behövs bara ett fåtal väldefinerade primitiver för att kunna skriva alla program som går att skriva:

- sekvens
- val (selektion)
- iteration



Allt annat i språken finns för att underlätta och öka sannolikheten för att man skriver korrekta program.

Ordlistan växer...

klasser, objekt, identitet, tillstånd, beteende, funktionalitet, instans, typ, instansmetoder, immutable, string, klassnamn, filnamn, instansiera,

identifierare (namn på saker), litteraler (konstanta värden), konstanter, typer, variabler och tilldelning

primitiva typer och klasstyper, referenstyper, sammansatta typer, uttryck, tilldelning, operatorer, prioritet, typomvandling,

Javas API, java.lang, strängar, wrappers, reserverade ord, operationer, modellera, bitmönster, kompilator, minnesplats,

boolean, char, byte, short, int, long, float, double, deklaration, static, final, typecast, charAt, compareTo, equals, indexOf, length, substring, valueOf, parseInt, kommentar, sekvens, iteration,

För nästa gång: läs kapitel 3 (val/beslut) och 4 (iteration)