Modeller, Objekt och klasser

Förelasning 10

TDA540 – Objektorienterad Programmering

CHALMERS

Objekt Orienterad Programmering

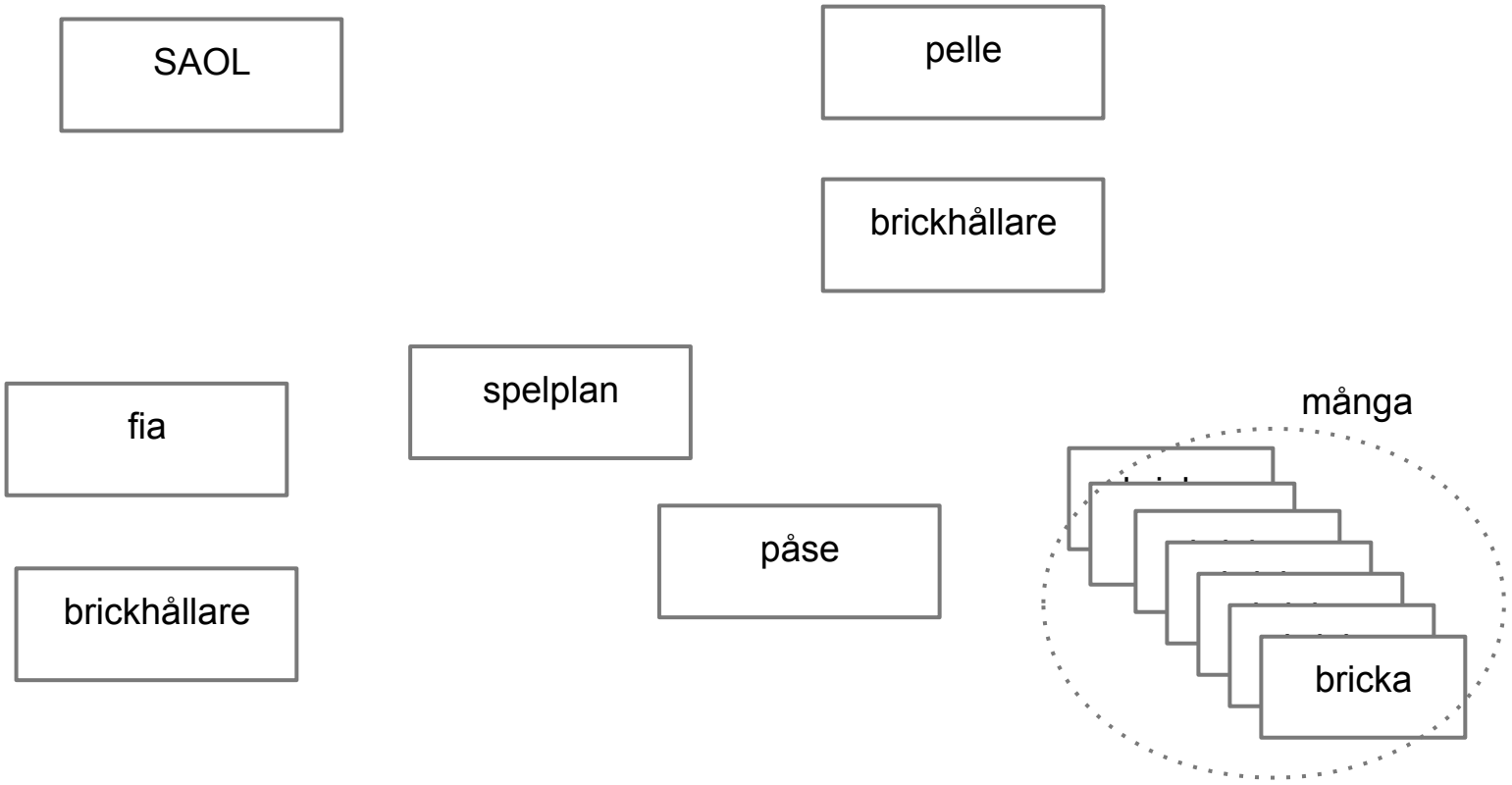
OO-programmering bygger på att vi som människor uppfattar tillvaron i termer av objekt

- Bastu, pizza, öl, ...

Det borde vara lättare att skriva program om programmen är uppbyggda på liknande sätt

- Programmet är en abstraktion (en objektmodell) av en vald del av vår verklighet
- Abstraktionen/modellen måste vi själva ta fram
- Kan du abstrahera?

En Objektmodell av vad?



(svar)

Lösa Komplexa Problem

Det klassiska sättet att lösa komplexa problem är att dela problemet i mindre “delproblem”

- Ev. dela dessa i ännu mindre
- När de minsta problemen är lösta, kombinera ihop (del)lösningarna till en lösning för hela problemet
- Objektorientering ger oss ett sätt att dela upp problemet, nämligen i objekt

Objekt

Ett objekt karakteriseras av

- Identitet, det som gör det möjligt att särskilja objektet från andra objekt (pelle, fia)
- Tillstånd, den data som finns i objektet (namn, poäng)
- Beteende, hur man kan använda objektet, vad man kan göra med det (pelle skall ta bricka ur påse)

Objekt Modeller och Java

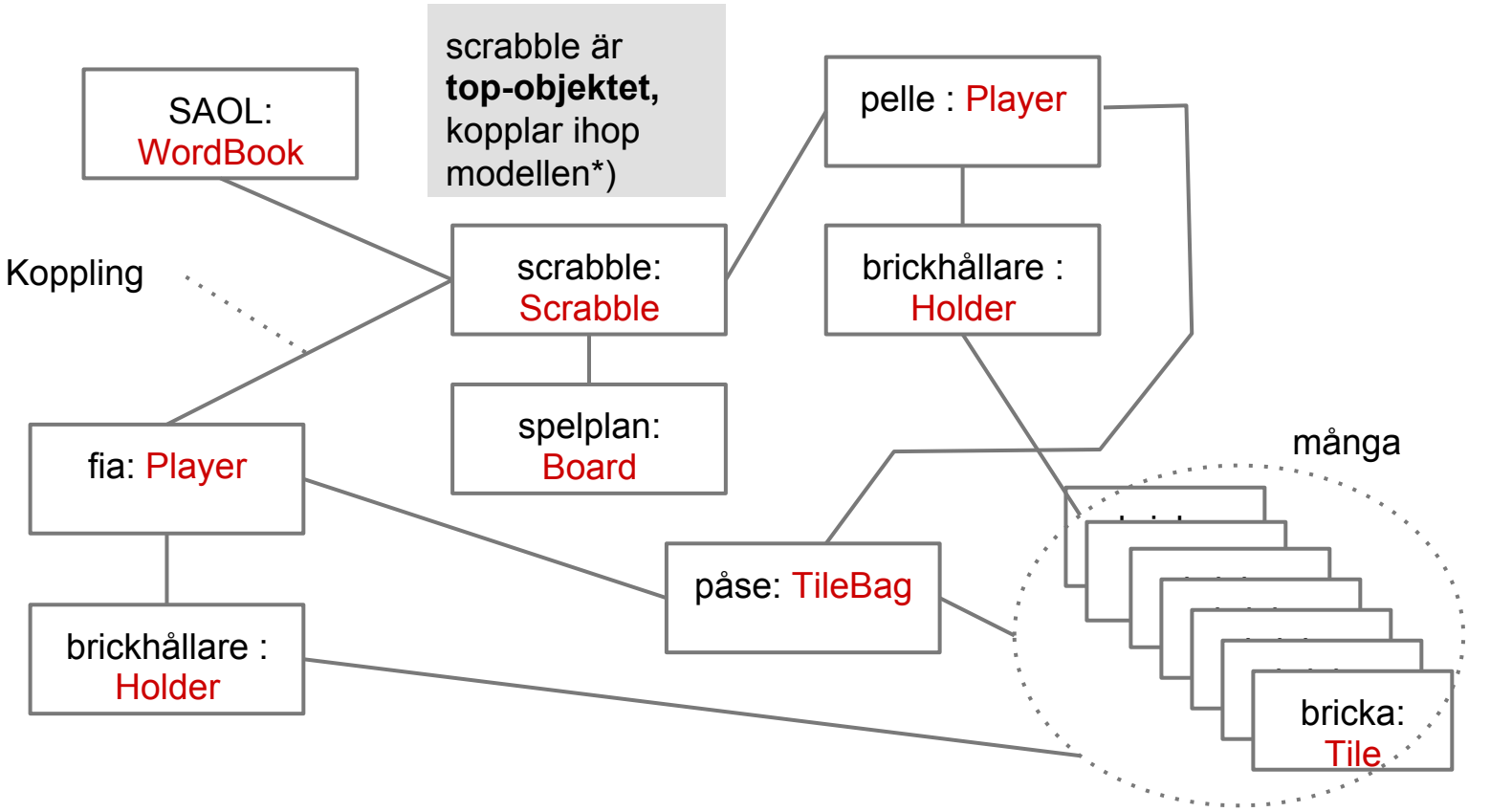
Objektmodeller består av interagerande objekt

- Objekt måste vara kopplade till varann (annars hänger de i luften, de kan inte interagera)

Alla värden i Java måste ha en typ

- … objekten är faktisk “värden”, de måste ha typer!
- Hmm, fia och pelle är värden! … Vilken är typen? Abstrahera (flera möjligheter)!

Java-anpassad Modell



*) Kan också ses som objektet för "hela spelet"

Typer i rött (finns andra tänkabar typer)

Objekttyper och Kopplingar

Hur skapar man modellens objekttyper och kopplingar?

Svar: Detta görs m.h.a. klasser

- Klassen anger objekttypen
- Klassen deklarerar vilka kopplingar (referenser) till andra objekt ett viss typ av objekt kan ha (m.h.a. instansvariabler = attribut)

Klass

För att skapa objekt behöver vi **klasser** (**class**)

- Klassen skapas först och m.h.a denna ett eller flera objekt
- En klass är en beskrivning (ritning) för objekten
- Alla objekt som har samma uppbyggnad/struktur skapas utifrån samma klass
- pelle och fia har samma struktur, de tillhör (har typen) Player
- Ett objekt är en **instans** (specialfall) av en klass. Pelle är en instans av Player
- .. som sagt en klass introducerar en ny referenstyp

Skapa Klasser

I Java skapas klasser genom att man

- Skapar en text fil, med klassens namn. Filändelsen måste vara .java
- I filen skriver man en klassdeklaration där man anger klassens namn (samma som filen) samt kropp (class body)
- Vid kompilering av *.java filer skapas *.class filer (de innehåller Java byte code för klassen

Tile: Klassdeklaration

```
// Class declaration in file Tile.java
class Tile {

    // Class body, delimited by { and }

}
```

Använder klasser från scrabble (Alfapet) som genomgående exempel

Klassmedlemmar

I klassdeklarationen anger man klassens **medlemmar**

Vi anger följande medlemmar

- Instansvariabler (= attribut)
- Konstanter (konstanta instansvariabler)
- Metoder
- Konstruktor(er)
- mer senare...

Instansvariabler

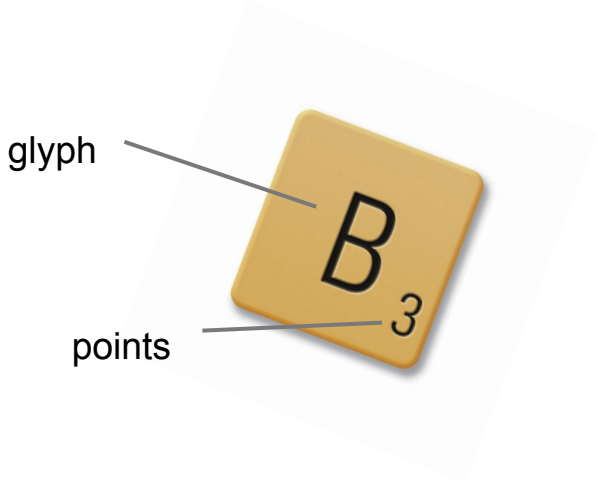
Instansvariabler ger objektets tillstånd

- Deklareras i kroppen (alltså mellan { och })
- Var man deklarerar spelar ingen roll, ofta först i kroppen
- Synlighetsområdet är hela klassen (klasskroppen)
- Deklaration anger typ, namn och ev. modifierare
- Normalt använder vi **private**, som modifierare, mer senare …
- Instansvariabler har förbestämda värden, int är 0, referenser är null
- Alla objekt har en egen uppsättning av instansvariablerna
- Vissa av instansvariablerna utgör kopplingar till andra objekt (det är så vi får en sammanhängande modell)

Tile : Instansvariabler

```
// Tile.java
public class Tile {
    private String glyph; // Instance variables
    private int points;
    ...
}
```

Tile objekt kan kopplas till objekt av typen String



Metoder

Metoder ger beteendet, det vi kan göra med objekten

- Metoder deklareras också direkt i klasskroppen enligt;

```
modifierare returtyp* namn ( parameterlista** ){   //Metodhuvud
        // Metod kropp
}
```

- Vi använder **public** och **private** som modifierare. Public kan anropas av andra objekt, private kan bara användas internt i objektet (hjälpmetoder)
- Returtypen är typen för det värde metoden returnerar (om något). Om returvärdet inte sparas (tilldelas en variabel) försvinner det.
- Ordningen (var i kroppen) spelar ingen roll

*) Om returtyp saknas anges void (= metoden är inte ett uttryck)
**) Parameterlistan kan vara tom

Tile: Metoder

```
// Tile.java
public class Tile {
     private final String glyph;
     private final int points;
     private boolean frozen;
     public String getGlyph() { return glyph; }
     public int getPoints() { return points; }
     public boolean isFrozen(){ return frozen; }
     public void freeze(){ frozen = true;}
}
```

Inget **static** används, mer senare ...

Konstruktor

En konstruktor är en speciell metod som exekveras då objektet skapas

- Används för initiering (ge instansvariablerna värden, t.ex. koppla ihop objektet med andra objekt)
- Har samma namn som klassen
- Noll eller flera parametrar
- Kan inte anropas som en vanlig metod
- Finns alltid en parameterlös **"default"-konstruktor** (även om den inte syns i koden). Skapas automatiskt men ...
- … om vi skapar en egen konstruktor med parametrar skapas inte default-konstruktorn (måste själva skriva dit den om vi vill ha en)
- Ingen returtyp skall anges. Om returtyp anges blir det en vanlig method, isf körs metoden inte då objektet skapas! Varning!

this

"this" står i vårt fall för en referens till det aktuella objektet

- this-referensen är tillgänglig i konstruktorn och syftar då på det objekt vi håller på att konstruera
- this referensen kan även användas i andra metoder (objektet vars metod vi exekverar)
- Används ofta för att kunna använda samma namn för en metodparameter och en instansvariabel (man kan skilja dem åt genom att använda "this" framför instansvariabeln)

Konstanta Instansvariabler

Mycket vanligt med konstanta instansvariabler, instansvariabler som inte kan ändras

- Variabler deklarerade som **final** kan inte ändras
- Variablerna måste ges värden i samband med att objektet skapas och ändras därefter aldrig (exempel: koppingen skall aldrig ändras)
- Konstanta variabler kan ges värden i konstruktorn
- Använd final generöst (det vi inte kan ändra på kan inte bli fel)

Tile: Konstruktor och this

```
// Using a constructor to initialize constant values
public class Tile {
    private final String glyph; // When created never changes
    private final int points;
    // No use of constructor, shall always start out as false
    private boolean frozen = false;

    // Constructor to set start values
    public Tile(String glyph, int points) {
        this.glyph  = glyph;  // Using the this reference
        this.points = points;
    }
     ...
     // methods here ...
     ...
}
```

Klasser och Typer

Att deklarera en klass innebär att vi får en ny referenstyp (med samma namn)

Givet: Alla variabeldeklarationer måste ange en typ
Givet: Vi har en ny typ

Slutsats: Vi kan deklarera variabler av den nya typen

```
// Declaring a variable using our new type Tile
private Tile t;   // No object just a variable!
```

Klass kontra Typ

I fortsättningen används klass och typ som utbytbara begrepp

- Typen Dice …
- … eller klassen Dice, syftar på samma sak ...
- Objekt av klass = objekt av typ
- Instans av klass = instans av typ
- ...

Instansieringsuttryck

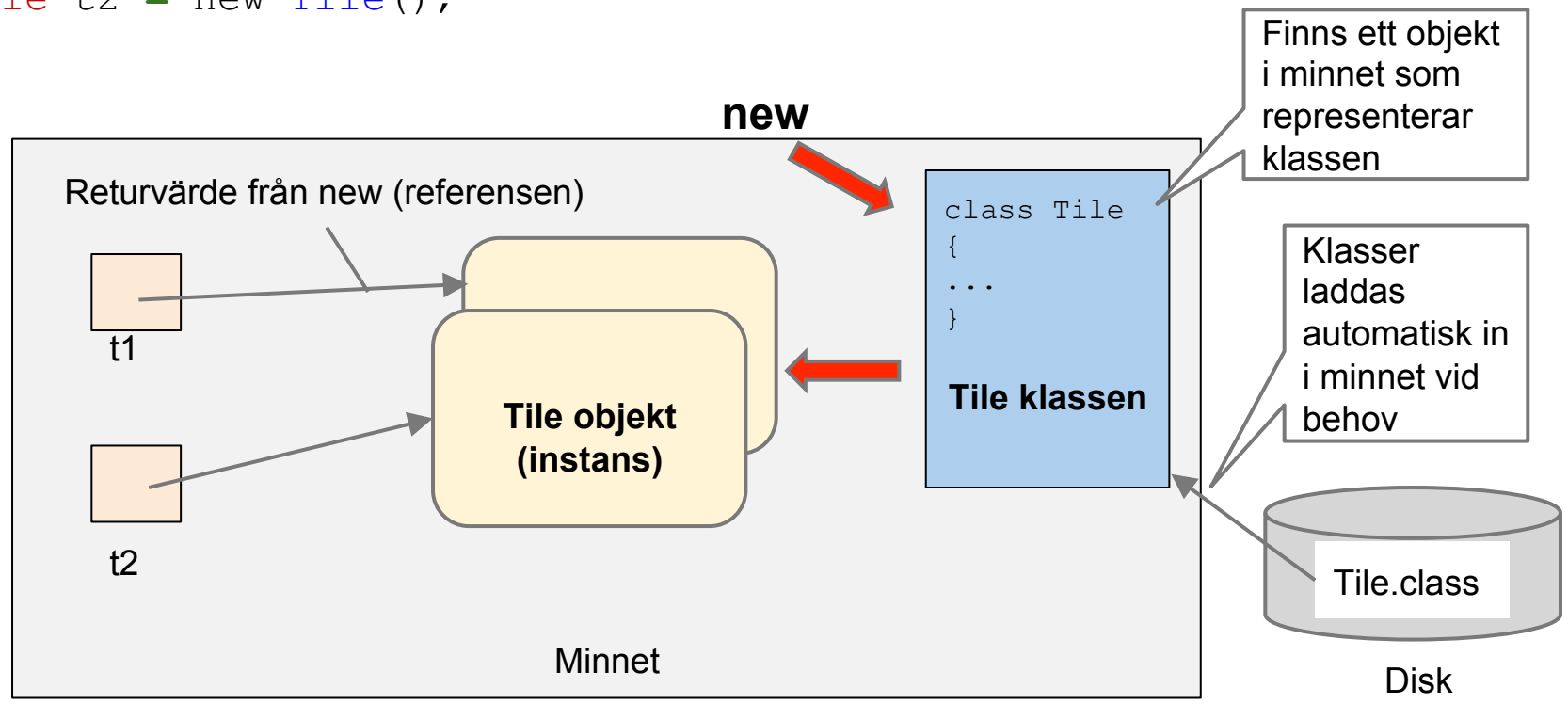
För att skapa ett objekt (en instans av klassen) används ett instansieringsuttryck (där **new** operatorn ingår)

```
// Class instance creation expression
new ClassName (arg, arg, ... )
```

- Värde för uttrycket är en referens till det nyss skapade objektet
- Alla som använder referensen refererar till samma objekt
- Sparas inte returvärdet (referensen) finns det senare inget sätt att komma åt objektet!

Skapa Objekt

```
// Instance creation using new, type, classname and assignment
Tile t1 = new Tile();
Tile t2 = new Tile();
```



Skräpsamling

De objekt vi behöver i programmet skapar vi. Men hur blir vi av med objekt vi inte behöver längre?

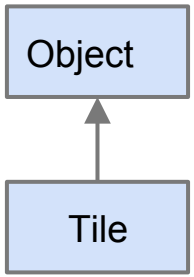
- Objekten skapas i minnet, många objekt -> mycket minne. Risk att minnet tar slut … ?
- Java hanterar detta genom s.k. **skräpsamling (garbage collection)** …
- … objekt som inte refereras förstörs automatiskt
- Vi behöver normalt inte bekymra oss

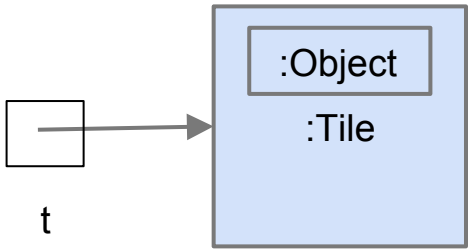
Klassen Object

Alla klasser vi skapar har en förälder

- Anges ingen speciellt sätts föräldern till den befintliga klassen Object (gäller för oss just nu). Föräldern kallas **super-klass** till vår klass. Object är superklass
- Våra klasser ärver därmed metoder från Object, t.ex. metoden toString()
- Om vi inte är nöjda med de metoder vi ärver kan vi skapa egna versioner i vår klass, kallas **överskugga**. Om så visar vi detta med @Override (en annotering)
- Objekt skapade utifrån vår klass kör vår (överskuggade) metod ist för den ärvda
- Den överskuggande metoden måste ha exakt samma namn, returtyp och parametrar som den ärvda
- De objekt vi skapar utifrån våra klasser har delobjekt av typen Object

Klasser
Objekt är super-klass till Tile





Objekt
Variabeln t refererar ett objekt av typ tile med ett delobjekt av typen Object

Tile: Överskuggning

Metoden toString() ärvs från Object, överskuggas i Tile

- Metoden anropas automatisk vid utskrifter, bra vid felsökning

```
// Tile.java
public class Tile {
     ...
     // attributes, methods here
     ...
     // Original inherited but we override with own version
     // (easier to read when written to output)
     @Override
     public String toString() {
         return "[" + glyph + "," + points + "]";    // We prefer this
     }
}
```

Array och egna Typer

Om vi skapat en klass kan vi deklarera en array med hjälp av den nya typen

- Klassen Holder (i Scrabble) använder en array av Tile (Tile[])

Namnkonventioner

Som vi sett ovan så används för

- Instansvariabler och metoder; inledande liten bokstav och därefter “camelCase”
- Klasser; inledande stor bokstav och därefter “CamelCase”

Läxa

Skapa objektmodell för spelet Gris (Pig)

- Anpassa modellen till Java