Enum, mer om Arv, Abstrakta klasser, Polymorfism, Recursion

Förelasning 14

TDA540 – Objektorienterad Programmering

CHALMERS

Enum

Uppräkningstyp (**enum** = enumeration)

- Används då en typ har ett fåtal (oföränderliga) objekt t. ex. veckodagar, färger, kön, … (varje objekt fungerar som ett värde)

- Typen deklareras som en klass men med enum istf class

- I enum:en räknas de ingående objekten upp (objekten skapas automatiskt vid körning)

- enum; kan inte ärva/ärvas, kan inte använda new, kan ha konstruktor och metoder, m.m.

Varför Enum?

Antag att vi använde heltal eller strängar för veckodagar

```
// Using int or String for weekdays
public final int MONDAY = 0       // Using int ...
public final String FRIDAY = “friday”;  // .. or String

// Compiler can’t warn us for misstakes ...
private int today = MONDAY;     // Ok!
private int today = -1;          // Oh, oh,

// Safe compiler reject anything but a Weekday
private Weekday today = ... must have type Weekday here
...
```

Arv och Överskuggning

Om vi låter en subklass överskugga en metod från superklassen gäller följande vid anrop av metoden;

- Om objektet är av superklasstyp anropas metoden i superklassen
- Om objektet är av subklasstyp anropas metoden i subklassen
- Vad som sker bestäms inte av den deklarerade typen (typen på referensvariabeln)

Kallas för **sen bindning (late binding)**. Vad som sker bestäms under körning (inte vid kompilering)

getClass()

Om man använder överskuggning (override) så bestäms vilken metod som skall köras utifrån objektets typ* (inte referensvariabelns)

Man kan kontrollera vilken typ ett objekt har m.h.a. metoden getClass()

- Finns i alla klasser (ärvd från objekt)
- Som svar får man ett Class-objekt (som alltså är klassen för objektet (kan ej ändra detta))

*) Kallas ofta **runtime typ** (för variabeln, förvirrande...)

instanceof

Ibland måste vi använda explicit typomvandling

- Gör vi fel blir det ClassCastException

Det går att kontrollera om typomvandlingen kommer att fungera m.h.a. **instanceof** operatorn. Ger sant om en referens är typkompatible med en given klass

```
// Using instanceof
Object o = …        // o could be of any type
// If true o typcompatible with String (or any subclass
of)
if( o instanceof String ){
```

Mer om final

Sammanfattning av **final**

- Anges för variabler som inta skall kunna ändras
- Anges för metoder som inte kan överskuggas
- Om det anges framför klassen (final class) kan man inte ärva klassen

De två sista används för att förhindra arv (arv kan vara farligt ibland, ingår inte denna kurs ...)

super

Vi har tidigare sett **super** (anrop av superklassens konstruktor)

super kan även användas för att komma åt superklassens metoder ifall dessa är överskuggade

```
// Overridden method in subklass
@Override
public void doIt(){
    super.doIt();  // Call method in superklass
}
```

Abstrakta Metoder

En abstrakt metod saknar metodkropp (som i ett gränssnitt, i gränssnitt är metoderna automatisk abstrakta, i klasser måste vi ange...)

```
// Abstract method in a class
public abstract void doIt();
```

Tanken är att någon subklass skall överskugga metoden och förse den med en kropp (körbar kod)

Abstrakta Klasser

En klass är abstrakt om den innehåller någon abstrakt metod

- Måste anges med public **abstract** class
- Abstrakta klasser kan inte instansieras (inte **new**)
- Används för att **faktorisera** d.v.s. flytta kod som är gemensam för flera subklasser till en gemensam superklass (den abstrakta klassen). Mycket dåligt att ha samma kod på flera ställen …

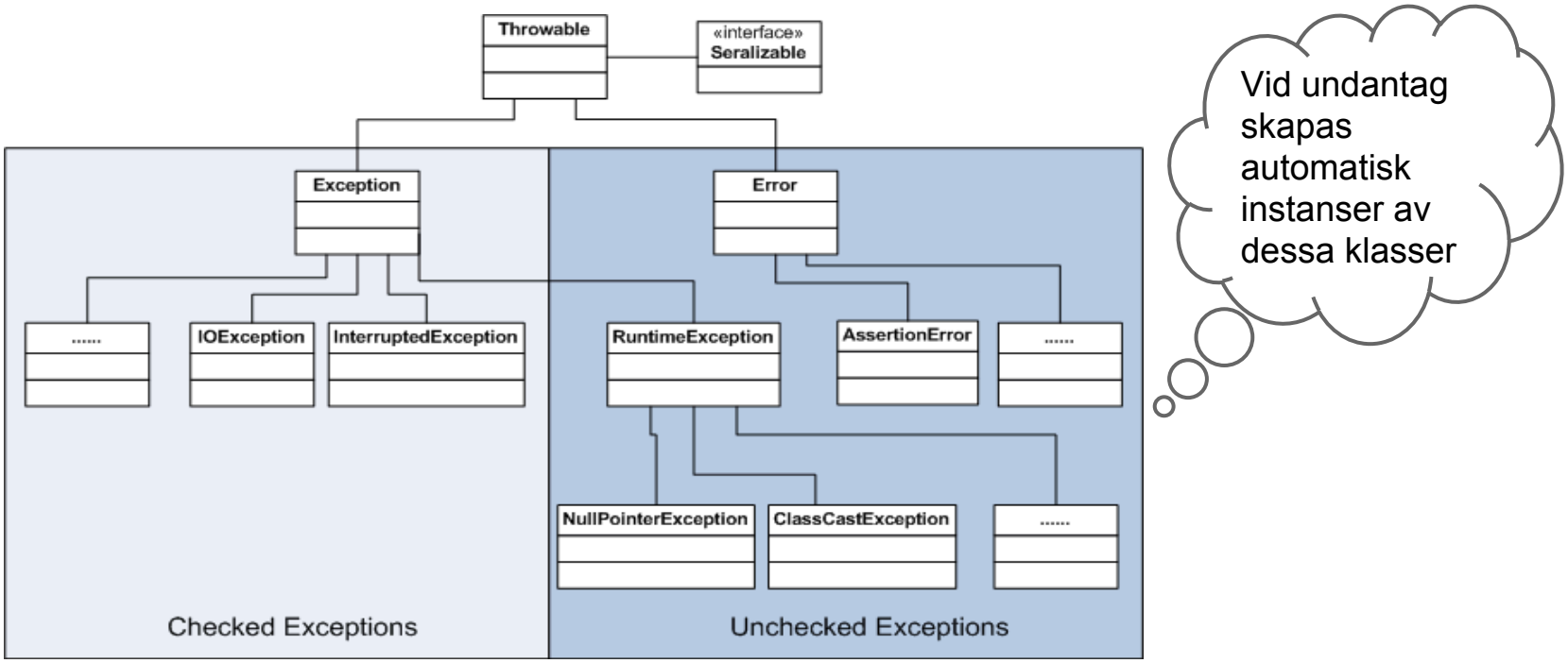
Polymorfism

Är ett centralt men svårfångat begrepp inom objekt orienterad programmering (på svenska "mångformighet")

saker beror på vilka typer som är inblandade (beteende utifrån typ)

Mer om Undantag

I Java finns två sorters undantag, unchecked exceptions och checked exceptions



Mer om Undantag, forts

Checked exceptions används för situationer programmet måste hantera (t.ex. nätet nere)

- Checked exception måste "fångas" (hanteras), kontrolleras att så sker vid kompilering
- Om man inte vill fånga felet direkt där det kan uppstå kan man skicka det vidare, man anger **throws** + typen för felet metodhuvudet)

Unchecked exceptions används för fel som programmeraren kan göra

- Unchecked fångas vanligen inte, har vi gjort fel så skall det "smälla" så snabbt och högt som möjligt (programmet skall krascha så att vi märker felet direkt)

Fånga Undantag

Checked exception fångas med **try..catch**..

- Satsen där undantaget kan uppstå finns i **try**-grenen
- I **catch**-grenen fångas (och ev åtgärdas) felet. Man deklarera vilken typ av felobjekt man fångar och namn på felobjektet (kan ha flera catch grenar för olika undantag)

Ibland låter man undantaget vandra hela vägen upp till GUI:et där man fångar och visar en dialogruta

Filhantering

Program behöver normal komma ihåg data mellan körningarna (användarinställningar, high-score:listor , m. m.)

- Då ett program avslutas töms minnet på allt som tillhör programmet
- För att kunna användas vid nästa körning sparas data på datorns hårddisk (i vårt fall), en fil skapas
- Vid nästa körning kan programmet läsa in datan i filen (eller delar av) och spara i minnet
- Vi kan inspektera filen i något filhanteringsprogram

Filhantering i Java

Finns färdig klasser för allt

- Vi använder de färdiga Scanner, PrintWriter och File (finns i boken)
- Filhantering innebär att vi kan få IOExceptions (filen kanske saknas), en Checked exception, måste fångas mer senare ...

Filformat

Filformatet bestämmer hur datan skall lagras

- **Binärfiler**, sparar data som ren "dump" av minnet, ej läsbart för människor, "konstiga" tecken då man tittar på innehållet m.h.a. av något program, inga rader (Java:s class-filer är binärfiler)
- **Textfiler**, innehåller text, ordnad i rader, läsbar för människor (filnamn ofta *.txt)

Vi använder bara textfiler

Lab 1: Omvandling av Text

När vi sparar instansvariabler kommer värdena att automatiskt att omvandlas till String (sköts av PrintWriter)

När vi läser in värden (strängar) så måste vi själva omvandla från String till t.ex. int eller boolean, görs enligt nedan

```
// From String to int
int i = Integer.valueOf(“123”);  // If “abc” exception
boolean b = Boolean.valueOf(“false”); // “false” or “true”
                                      // else exception
```

Swing och Filhantering

För att välja filer att öppna eller spara (namn och placering i filsystemet) finns en färdig komponent i Swing; JFileChooser

Rekursion

"**Recursion** is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until you get to a small enough problem that it can be solved trivially. Usually recursion involves a function [method] calling itself. While it may not seem like much on the surface, recursion allows us to write elegant solutions to problems that may otherwise be very difficult to program."

- Ett enkelt problem: Summera alla tal i en lista (enklare med loop, exemplet bara för demo)
- Ett svårare problem: Towers of Hanoi!

Tower of Hanoi

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk may be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
3. No disk may be placed on top of a smaller disk

Min antal flyttningar : $2^n - 1$

(64 diskar, flytta en per sek. 2^{64}-1 = 18,446,744,073,709,551,615 s = 585 miljarder år, 127 gånger solens nuvarande ålder)

Lösning Tower of Hanoi

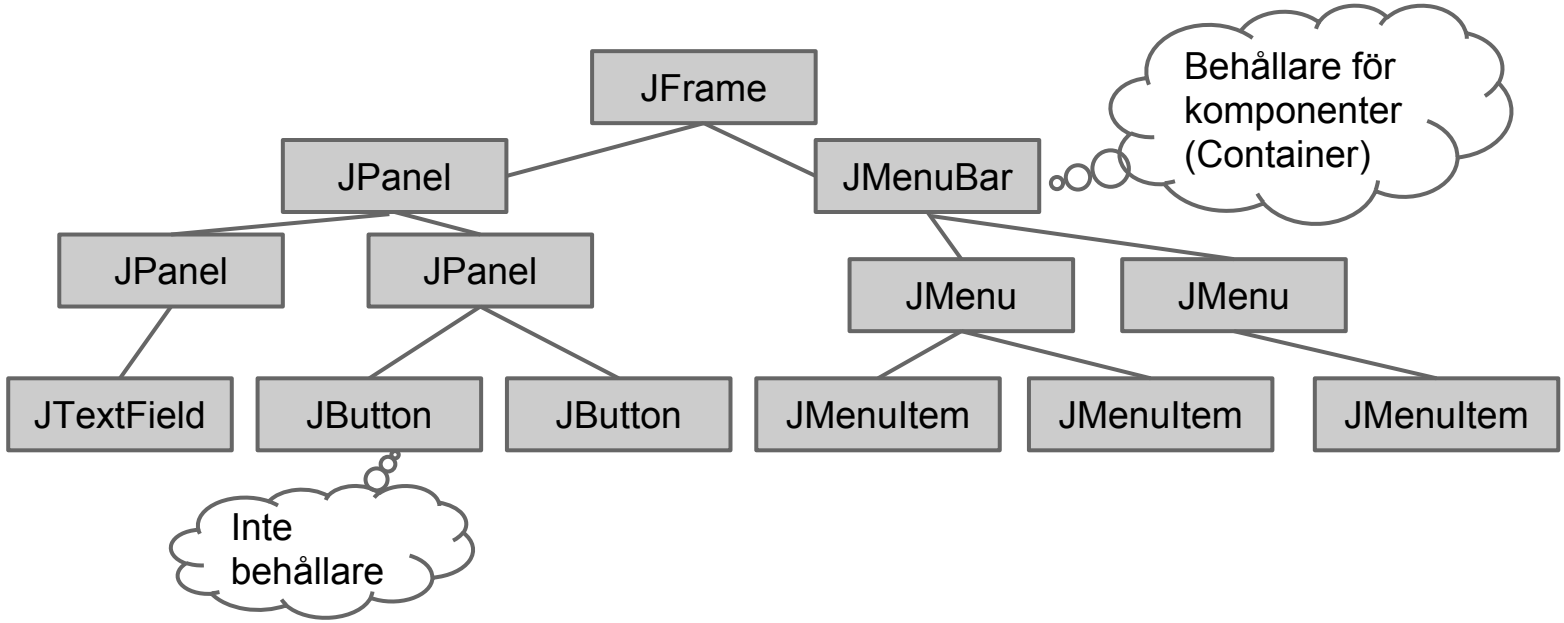
```
class Hanoi {
   public static void main(String[] args) {
      move(3, 1, 2, 3);
   }

   public static void move(int n, int from, int to, int via) {
      if (n == 1) {
         System.out.println("Move disk from pole " + from + " to pole " + to);
      } else {
         move(n - 1, from, via, to);
         move(1, from, to, via);
         move(n - 1, via, to, from);
      }
   }
}
```

Lab 1: Rekursion

Det skall vara möjligt att ändra fontstorlek i hela GUI:et (meny View > Big Font). Hur göra?

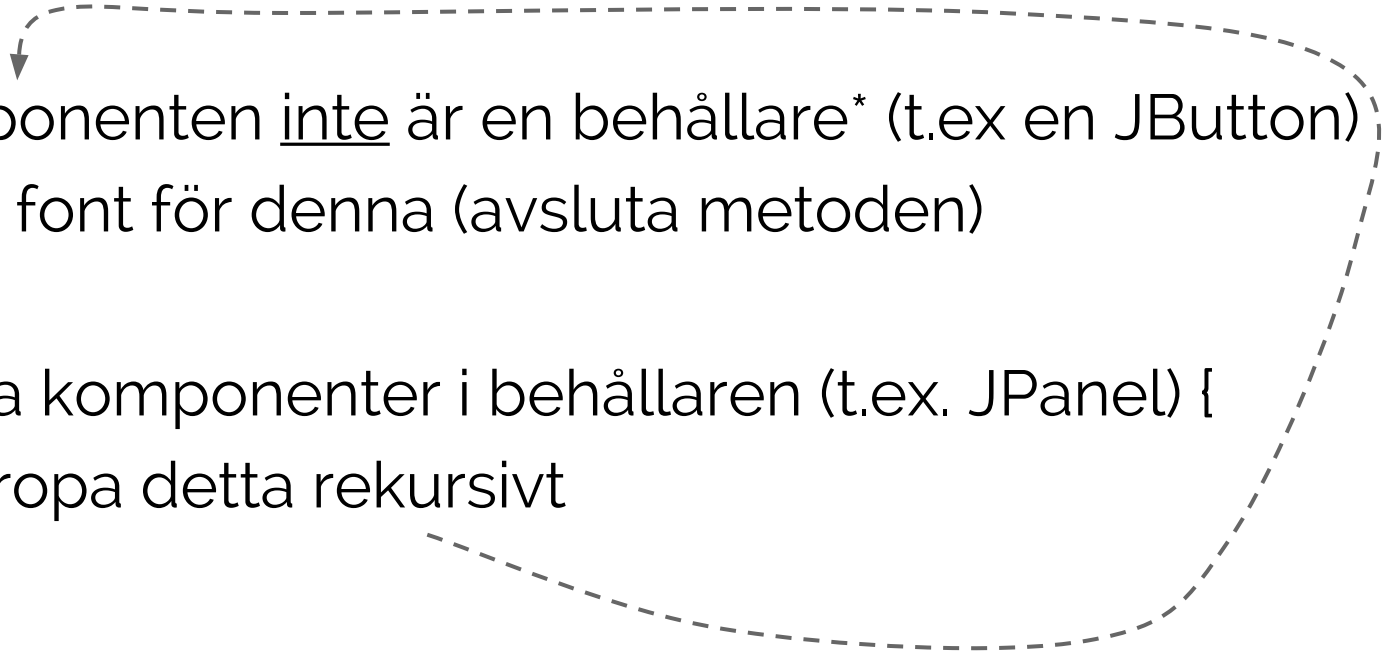
- OBS! Att ett GUI:et har en struktur som ett (upp och nedvänt) träd



Lab 1: Rekursion, forts

Indata: En (start) komponent

Om komponenten inte är en behållare* (t.ex en JButton)
    Ändra font för denna (avsluta metoden)
Annars
    för alla komponenter i behållaren (t.ex. JPanel) {
        anropa detta rekursivt
    }



*) Kan undersökas med: if(component instanceof Container)