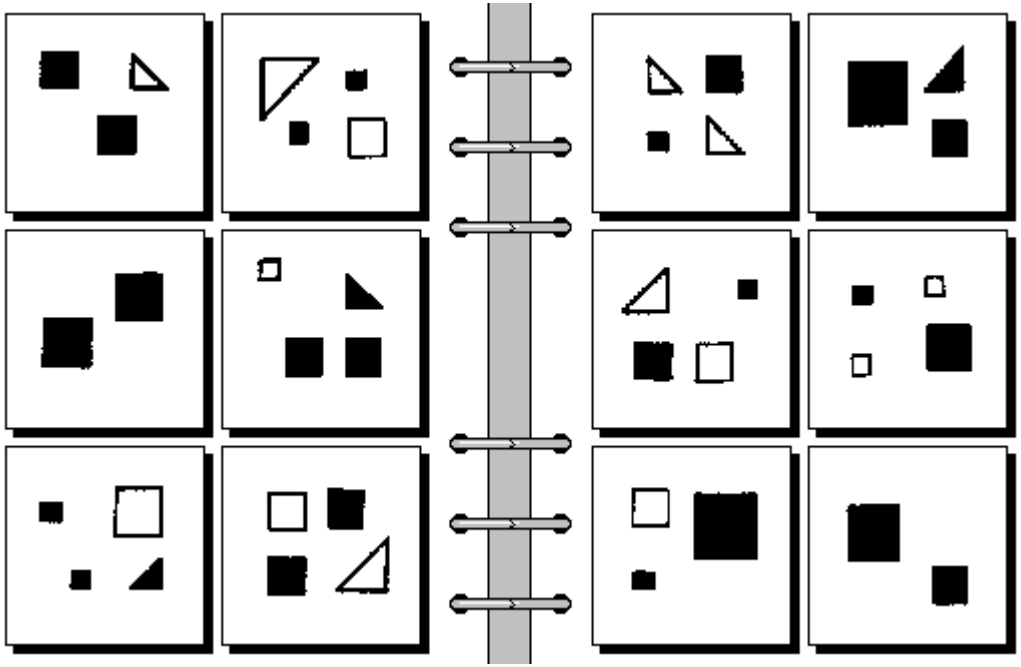
Abstraktion, Mer om Undantag, Filhantering, Rekursion och mer GUI

Bildserie 7

Mer om Abstraktion

Kommer ni ihåg?



Varför Abstraktion?

Abstraktion ger oss en möjlighet att hantera många (till synes) olika fenomen/objekt på ett enhetligt sätt

- För alla …. gäller
- Inga … har/kan…
- Samma …

Förenklar problemlösning, vi kan bortse från specialfall

- Gör tillvaron greppbar

Abstraktioner i Kursen

Överlagring (overload), kan använda samma namn för olika metoder (även flera konstruktorer)

Implements, alla objekt som implementerar gränssnittet kan behandlas på samma sätt (alla objekt har de metoder som anges i gränssnittet), EventBus

Extends, alla subklasser kan behandlas som superklassens, de kan minst lika mycket (ärver kod är en rent praktisk sak), StandardYatzy

Överskuggning (override), alla objekten vet själva hur de skall bete sig, vilken metod som anropas, avgörs av typen för objektet, RobotYatzy

Samlingar, alla samling kan hantera vilka objekt som helst, vi anger typen med t.ex. List<String> eller List<Tile>, DiceCup

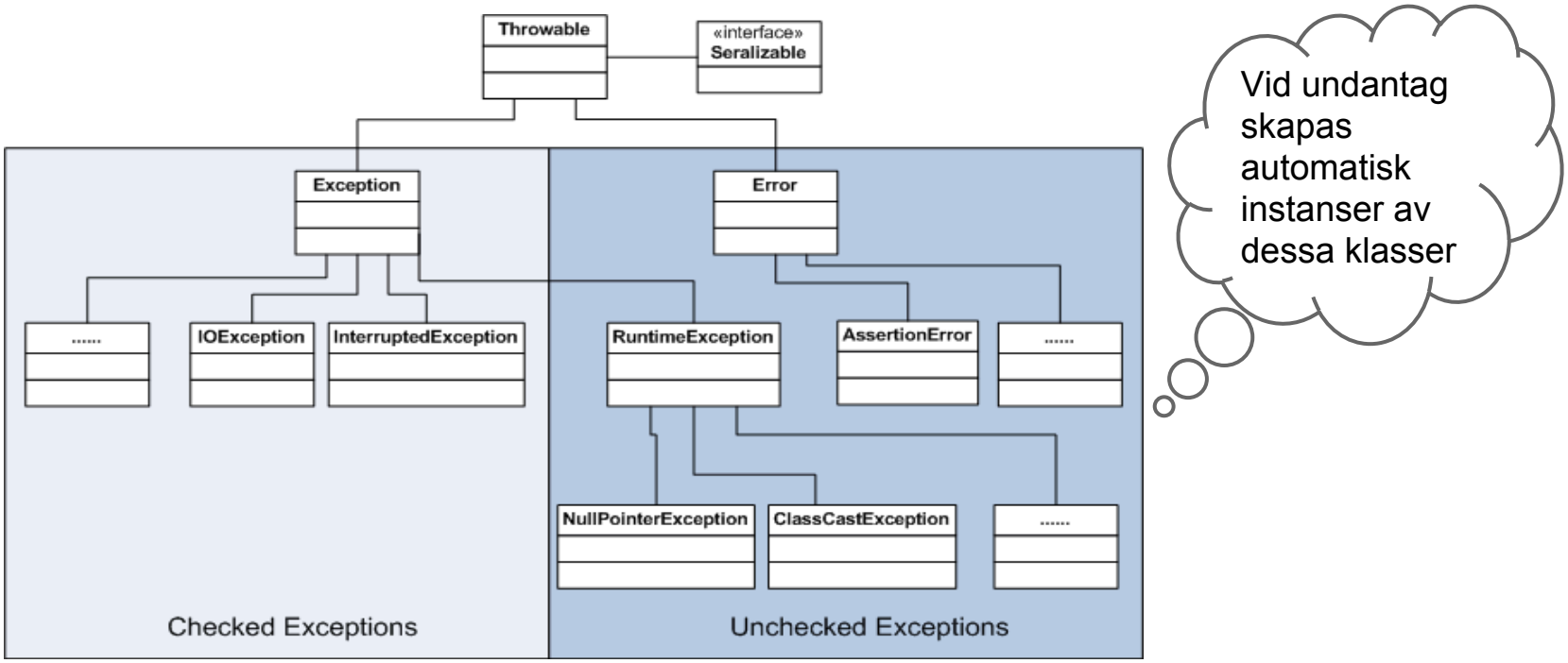
Polymorfism

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

Vi syftar på det i föregående bild, 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 (en textfil behövs)

Lab 1: Filformat

Vi sparar spelets tillstånd i en textfil (valfritt namn)

Filen innehåller en rad för varje spelare

Raden består av; Namn, ordningstal, rad, poäng, satt, rad, poäng, satt, ...

```
// Content of yatzy.txt. Anna is player 0, Urban player 1
Anna,0,0,3,true,1,4,true,2,0,false,3,0,...
Urban,1,0,3,true,1,4,true,2,0,false,3,0,...
```

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

Lab 1: Filhantering

Vi vill spara aktuell ställning (ifall vi avslutar och vill fortsätta senare)

- Vi abstraherar bort detaljerna hur detta sker genom att använda ett gränssnitt (IYatzyReader)
- Vi skapar en hjälpklass (en fabrik) som kan leverera objekt som implementerar IYatsyReader
- Vi skapar en implementation av IYatzyReader (YatzyPlainFileReader, som skriver/läser textfiler från disk). OBS! Att vi senare kan byta ut denna mot en klass som t.ex. går via nätet till en databas

Lab 1: Filhantering, forts.

Omvandling mellan text och objekt sköts enligt tidigare bild (omvandling sker i read/write-metoder)

Undantag skickas vidare till GUI (throws i metodhuvudet), i GUI:et fångas undantaget och en dialogruta visas (JOptionPane)

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:

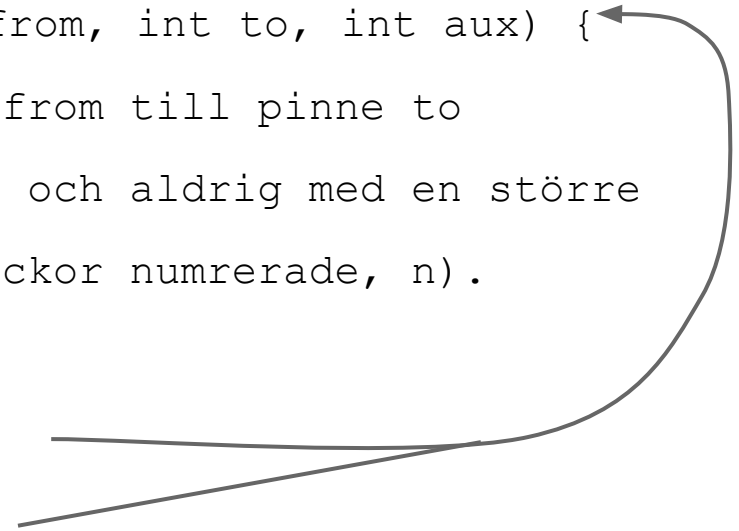
1. Only one disk may be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
3. No disk may be placed on top of a smaller disk

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

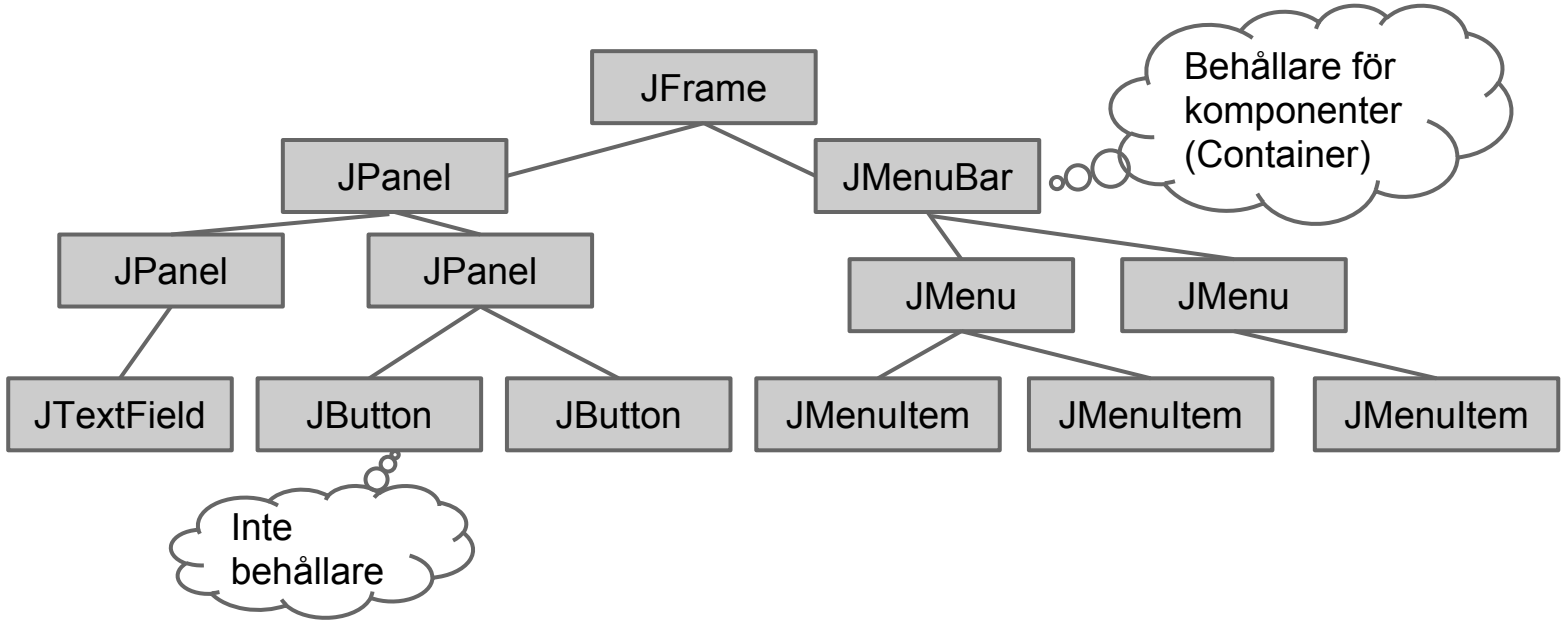
```
public static int move(int n, int from, int to, int aux) {
    // Flytta n brickor från pinne from till pinne to
    // med pinne aux som hjälppinne och aldrig med en större
    // bricka ovanpå en mindre (brickor numrerade, n).
    if (n > 0) {
        move(n - 1, from, aux, to);
        move(n - 1, aux, to, from);
    }
}

move(nrOfDiscs, 1, 2, 3);  // Första anrop
```



Lab 1: Rekursion

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

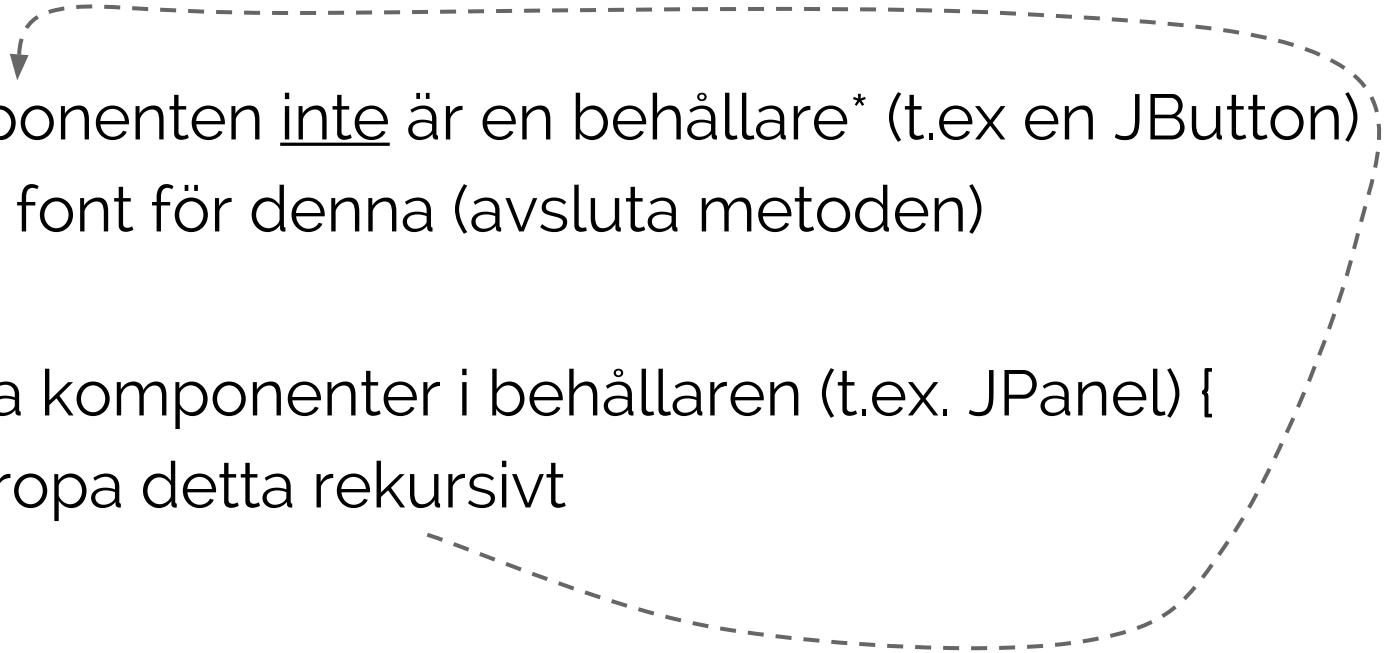
- OBS! Att ett GUI:et har en struktur som ett (upp och nedvänt) träd



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)

paintComponent()*

Om vi använder färdiga Swing-komponenter kommer de att ritas ut på ett förbestämt sätt (finns några att välja på Pluggable look'n' feel)

Är vi inte nöjda kan vi ändra hur komponenten skall ritas ut

- Görs genom att överskugga metoden paintComponent()
- I metoden skriver man kod som ritar efter våra önskemål (kan även visa ikoner, bilder)

*) Detta är frivillig överkurs

Graphics

Som inparameter till paintComponent får vi ett objekt av typen java.awt.Graphics (en abstrakt basklass)

- Klassen gör det möjligt för en komponent att rita
- Måste explicit typomvandlas till Graphics2D
- Innehåller info om vilken komponent det gäller, färg, font, olika pixel-operationer, olika objekt att rita ut, linje, rektangel (iofs bättre med klassen Shape)

java.awt.ToolKit är en klass som ger tillgång till implementationen av AWT (methoden getDefaultToolkit()), kan användas t.ex. för att få information om storlek och upplösning på skärmen

Animering*

Man kan styra när en komponent skall ritas m.h.a. en timer (javax.swing.Timer)

- Timern skickar händelser med visst intervall, en lyssnarmetod kan ta hand om händelsen …
- … och anropa methoden repaint(), som i sin tur automatiskt anropar paintComponent()

*) Detta är frivillig överkurs