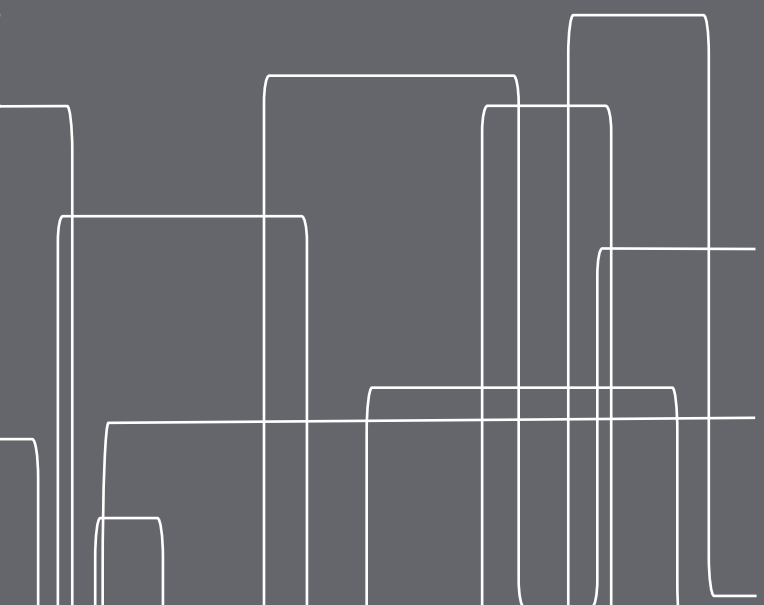
Objektorienterad Programkonstruktion

Föreläsning 1

2 nov 2015



Kontaktuppgifter & Info:

Kurskod:	DD1346, 6hp
Kursomgång:	oopk15
Kursansvarig:	Christian Smith, ccs@kth.se
Hemsida:	https://www.kth.se/social/course/DD1346/
Kursanmälan:	https://rapp.csc.kth.se/rapp

Får ej ingå i examen tillsammans med:

DD1332, DD1340, DD1341, DD1342, DD1343, DD1344, DD2385

Kursbeskrivning (Studiehandboken)

“Kursen behandlar principer för objektorienterad programmering, objektorienterad programmering i Java, objektorienterad modellering med UML samt designmönster.

Kursens huvudsakliga innehåll:

Programmering i Java inklusive biblioteksanvändning, modellering med UML, principer för god objektorienterad design, designmönster. Laborationer i Java, uppgifter i UML.”

Lärandemål (Studiehandboken)

Efter kursen ska studenten kunna:

• förklara och använda begrepp inom objektorienterad programmering

• använda objektorienteringens principer vid programmeringsarbete

• använda UML-klassdiagram för att på ett överskådligt sätt planera och dokumentera eget programmeringsarbete

• läsa och förstå UML-klassdiagram

• redogöra för några vanliga designmönster samt känna igen situationer där de bör användas

• utveckla objektorienterade program i Java

• använda Javas bibliotheksklasser och ramverk.

Kursupplägg

- Föreläsningar
  - (14+8) x 1h (45 min)
  - tar upp teori, principer, terminologi
- Övningar
  - (4+4) x 2h
  - tar upp Javasyntax, praktiska exempel
- Labbar
  - (7+5) x 2h x 2 grupper (+ stor del egen tid)
  - Mindre objektorienterade programmeringsuppgifter
  - Ett lite större projekt

Labbar

- 5 labbar för betyg E eller C, en extra labb för A
- Labbarbetet utförs parvis, båda ansvarar för resultatet
- Labbtid ca 3x2 h per vecka, för handledning & redovisning
- Eget arbete troligtvis ca 10 h per vecka
- Utgör det största kursmomentet, sett till nedlagd tid
- Varje labb redovisad i tid ger 1 bonuspoäng på E-delen på tentan, maximalt 5 bonuspoäng, giltiga till nästa kursomgång (dvs för en tenta + omtenta).

Labbar

- Introduktionslabb
- Knappar och Applets
- XML
- Simulering, Grafik
- Webbläsare
- Designmönster

Projektuppgift

- En lite större labbuppgift, egen planering
- IM, skickar meddelanden mellan datorer
- Basuppgift för E, välj till extrauppgifter för C, B eller A
- Labbarbetet utförs parvis, båda ansvarar för resultatet
- Labbtid ca 2 h per vecka, för handledning
- Först redovisas en projektplan med UML, vilken kan ge en bonuspoäng på tentan (ca v.5)
- kod + funktionalitet redovisas senast v. 8 för en bonuspoäng på tentan

Hederskodex

Regel 1: Alla ansvarar vid arbete i grupp

Vid grupparbete har alla i gruppen ett gemensamt ansvar för arbetet.

Många uppgifter görs i grupper om två eller fler studenter.

Vid arbete i grupp är samarbete inom gruppen naturligtvis tillåtet. Varje gruppmedlem ska bidra till arbetet på ett rättvist sätt. Alla gruppmedlemmar ska, var för sig, kunna redogöra för hela uppgiften och hela lösningen.

Hederskodex

Regel 2: Redovisa ärligt hjälp som erhållits och källor som använts

Om det finns delar av lösningen som studenten inte gjort själv, så måste studenten göra examinerande lärare uppmärksam på detta.

I många sammanhang är det naturligt att använda sådant andra har gjort. Vid programmeringsuppgifter kan det vara naturligt att använda färdiga exempel som finns i kurslitteraturen eller som kursledningen tillhandahållit. Detta ska tydligt redovisas, till exempel i form av kommentarer i koden. När man skriver rapporter/uppsatser är det naturligt att använda olika typer av källor och dessa ska då redovisas i form av referenser och en källförteckning (direkta citat måste explicit anges). Den som utnyttjar en idé som härstammar från en annan person ska tydligt redovisa varifrån idén kommer. Detta gäller även idéer som förmedlats muntligt, exempelvis vid diskussion med andra studenter.

När man kör fast på en (programmerings-)uppgift kan man behöva be en handledare eller en kamrat om hjälp med felsökning eller tips. Det är tillåtet, men hjälpen ska tydligt redovisas, till exempel i form av kommentarer i koden eller i labbrapporten om det gäller mer än någon enstaka kodrad eller mening. Den som söker hjälp med att lösa sin uppgift ska göra det i syftet att öka sin förståelse, inte i syftet att snabbt och enkelt klara av uppgiften.

Tanken är naturligtvis inte att studenterna ska leva i ett vakuum och inte få diskutera sina uppgifter med sina kamrater. Diskussioner kamrater emellan uppmuntras, men efter diskussionen ska var och en göra sin egen lösning. En student som, enligt examinerande lärares bedömning, gjort alltför liten del av lösningen själv har inte presterat tillräckligt för att kunna godkännas på det aktuella kursmomentet

Hederskodex

Regel 3: Kopiera inte från andras lösningar

Varje student ska skriva sin egen text (egen programkod).

Kopiering av text (eller programkod) från andras lösningar är förbjuden oberoende av källa. Kopiering av text (eller programkod) är inte tillåten även om texten (eller programkoden) skrivs om så att ytstrukturen är olika men innehållet är detsamma. I vissa kurser används ett program som beräknar likheten mellan olika lösningar till samma uppgift.

Hederskodex

Regel 4: Var beredd att redogöra för hela lösningen

Varje student ska vid muntlig examination kunna redogöra för hela uppgiften.

Varje student ska vid muntlig examination kunna redogöra för hela uppgiften och hela lösningen (även de delar som studenten eller gruppen inte gjort själv) det gäller alltså att vara påläst vid redovisningen.

Övningar

- Ger praktiska demonstrationer i Javaprogrammering och UML-diagram
- Möjlighet att ställa frågor av praktisk natur
- Kommer att gå igenom moment som sedan kommer på labbarna
- Tre övningsgrupper (Christian & Mateusz)

Examination

- Lab1 (2hp)
  - 5 labbar som görs antingen till E- eller C-nivå
  - Labbar gjorda i tid ger bonuspoäng till tentan
  - 6:e labb för A, tillsammans med 5 C-labbar
- Ten1 (2hp)
  - Tenta, med en E-del som kräver hög andel rätta svar, samt problemdel för högre betyg (mars 2016)
- Lab2 (2hp)
  - Självständigare programmeringsuppgift,
  - löst grunduppgift ger E, extrauppgifter för högre betyg.

Betyg

Betygen på varje delmoment kan ge A, (B), C, (D) eller E. Slutbetyget sätts som ett aritmetiskt medelbetyg.

E ges värdet 1, D ges 2, C ges 3, B ges 4 och A ges 5.

Medelvärdet avrundas till heltal, och ett avrundat medel på 1 ger betyg E, 2 ger betyg D, 3 ger C, 4 ger B och 5 ger A.

Litteratur

- Labbanvisningar: Alla finns på kurshemsidan under Laborationer
- Föreläsningar och övningar: I princip allt material från undervisningen kommer att göras tillgängligt på kurshemsidan.
- Webben: Utöver föreläsningar och labbanvisningar kommer varje avsnitt att förses med länkar till relevant läsning på nätet. I första hand kommer vi att hänvisa till Oracles Java Turorial, samt Java SE 7 API. För den som kommer ihåg Python från ettans kurs och vill ha Java förklarat ur ett pythonperspektiv föreslås Java for Python Programmers av Bradley Miller.

Programmeringsparadigmer

Lågnivåspråk:

Maskinkod (binärkod)

Instruktioner som kan utföras direkt av processorn

Exempel: `b0 61` // Lagrar värdet 97 i AL-registret

Assembler (50-talet)

Nära maskinkod, men lättare för människor att läsa

Exempel: `MOV EAX, EBX` // lagrar värdet från register EBX i register EAX

Programeringsparadigmer

Procedurella språk (sent 60-tal, 70-tal)

Kommandon kan packas i procedurer, man skriver hur man vill att maskinens tillstånd skall ändras. Programmet kompileras till maskinkod innan det kan köras.

Exempel: Fortran, Cobol, (Python), C

```
float mean(float a, float b){
    return (a + b)/2;
}
int main(){
    float c;
    c = mean(17, 55.3);
}
```

Programmeringsparadigmer

Funktionella Språk (70-tal)

Program uttrycks som matematiska funktioner som ska utvärderas

Exempel: Scheme, Haskell, (Python), Common Lisp:

```
(defun mean (A B)
    "Compute the mean of A and B."
    (/ (+ A B) 2))
(mean 17 55.3)
```

Programmeringsparadigmer

Objektorienterade språk (70-tal)

Programdelar kapslas in i objekt, som innehåller datafält och metoder som verkar på objekten

Exempel: Simula67, SmallTalk, C++, (Python), Java:

```
public class 2DVector extends AbstractVector{
  private double x,y;
  public double getMean(){
      return (x+y)/2;
  }
}

double mean = My2DVector.getMean();
```

Programmeringsparadigmer

Funktionella Språk (70-tal)

Program uttrycks som matematiska funktioner som ska utvärderas

Exempel: Scheme, Haskell, (Python), Common Lisp:

```
(defun mean (A B)
    "Compute the mean of A and B."
    (/ (+ A B) 2))
(mean 17 55.3)
```

Objektorientering

Motiverat av allt mer komplexa programmeringsprojekt.

Programkomponenter lagras i “objekt”. Hur dessa ser ut inuti döljs från resten av programmet. Man kan interagera med objekt med hjälp av publika metoder.

Underlättar modulär design, och utbytbara programkomponenter.

Exempel: Ett objekt kan inehålla en bild, och som programmerare behöver man inte veta vilket format bilden är lagrad i för att kunna skala om den till en annan storlek, utan man anropar bara t.ex:

```
minBild.getScaledInstance(width, height, hints)
```

Terminologi

Mallar för hur **objekt** ska skapas kallas **klasser**

Man säger att **objekt** är **instanser** av **klasser**

Ex: objektet ZGF 123 är en **instans** av **klassen** Bil

Klasser (och Objekt) har **fält** som rymmer data (egenskaper), och **metoder** som kan anropas för att utföra något.

Ex: Klassen Matris har

- fälten: antal rader, antal kolumner, elementen

- metoderna: transponera, invertera, singulärvärdesuppdela, ...

Terminologi

Om en klass **B** är en förfining eller delmängd av en annan klass **A**, så säger man att **B ärver** från **A**.

Ex: klassen `Rotation` ärver från klassen `Matris`, vilket betyder att den delar vissa egenskaper med alla andra matriser.

Om en klass **B** har ett antal egenskaper som beskrivs i ett gränssnitt **C**, så säger man att **B implementerar C**.

Ex: klassen `Rotation` implementerar `Inverterbar`, vilket innebär att den tillhandahåller en metod för att generera en invers.

Terminologi

Vi hänvisar till fält i ett objekt med punktnotation:

Ex: `song.composer`

Ett fält i ett objekt, kan i sig vara ett objekt, då kan vi skriva längre uttryck för att hänvisa till något.

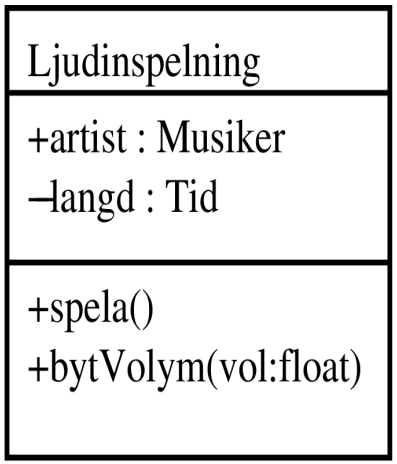
Ex: `song.composer.name`

Metoder anropas på liknande vis, men tar (eventuellt tomma) argument också:

Ex: `song.convertToMP3(bitrate)`

UML

För bättre överskådlighet kan man använda UML för att producera en grafisk representation av klasser och objekt.



+ Publik (Public)

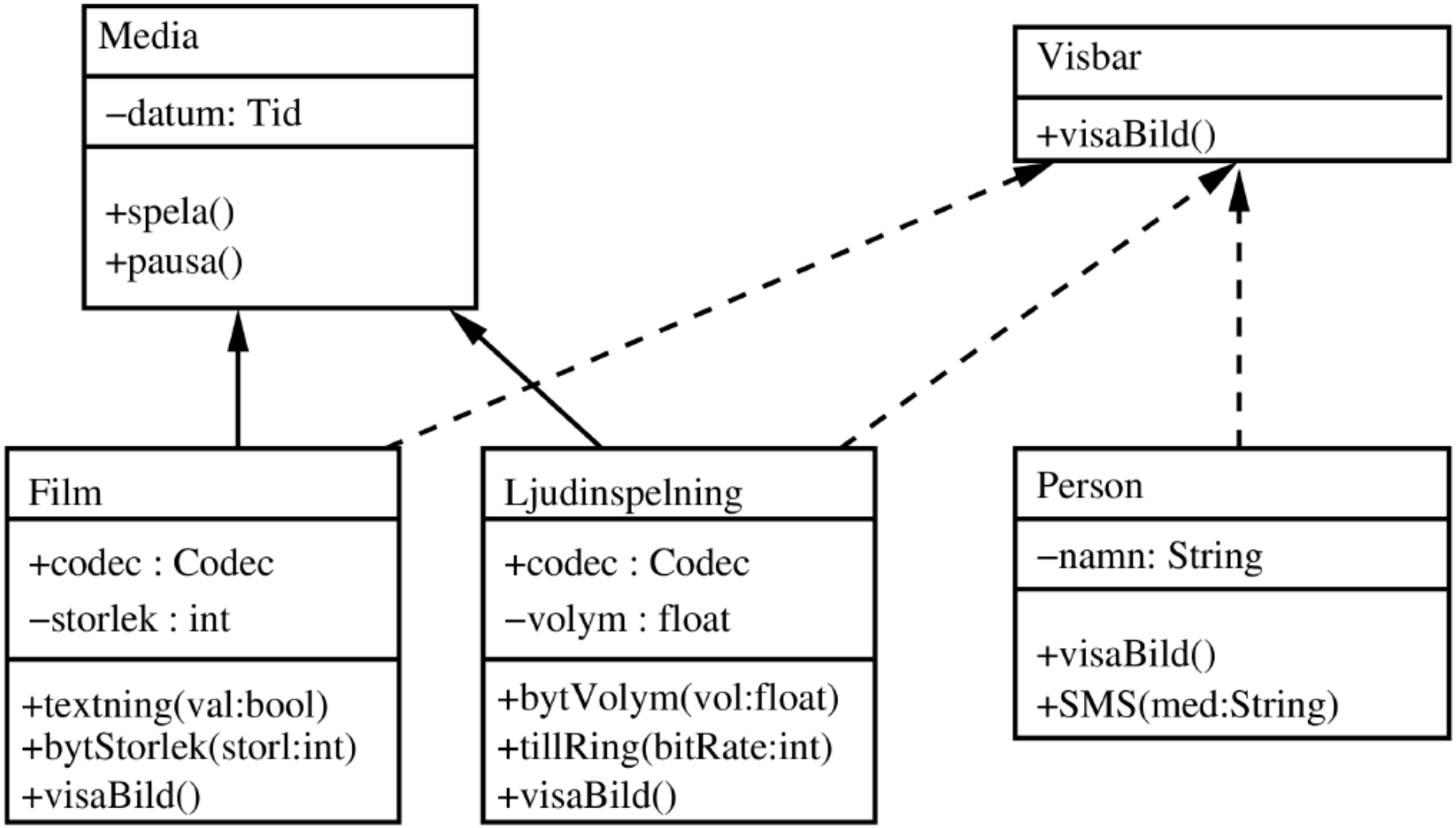
- Privat (Private)

~ Paket (Package)

Skyddad (Protected)

Statisk (Static)

UML


Media
–datum: Tid
+spela()
+pausa()
Visbar
+visaBild()
Film
+codec : Codec
–storlek : int
+textning(val:bool)
+bytStorlek(storl:int)
+visaBild()
Ljudinspelning
+codec : Codec
–volym : float
+bytVolym(vol:float)
+tillRing(bitRate:int)
+visaBild()
Person
–namn: String
+visaBild()
+SMS(med:String)

Java

Varför lära sig Java?

Java är ett av världens vanligare programmeringsspråk, väl understött och dokumenterat.

Man kan fort skriva program som kan köras “var som helst”.

Java är ett relativt enkelt språk för att lära sig objektorientering, tillräckligt likt t.ex C++ eller C# för att underlätta inlärning av dessa.

Java

Skapades 1995 på Sun Microsystems, ägs och utvecklas numera av Oracle.

Java används idag främst i applikationer med grafiska användargränsnitt, som ska kunna köras på många olika platformar. Exempel:

Menysystemet på Blue Ray-skivor

Program till mobiltelefoner (Android)

Matlabs grafiska gränsnitt

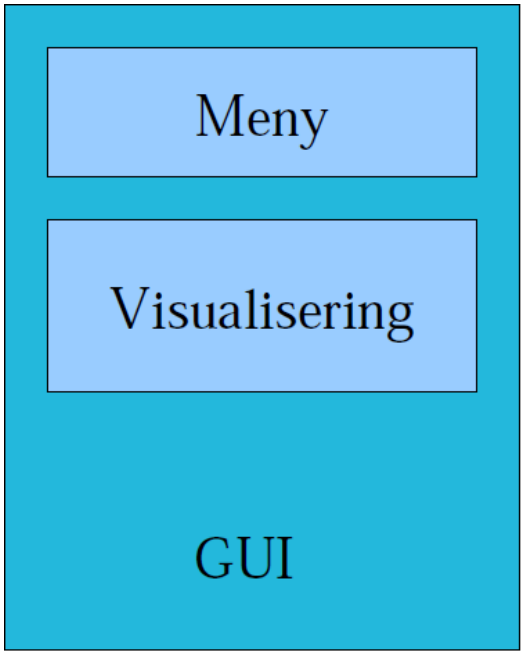
3D-kartan på hitta.se (numera “min stad”)

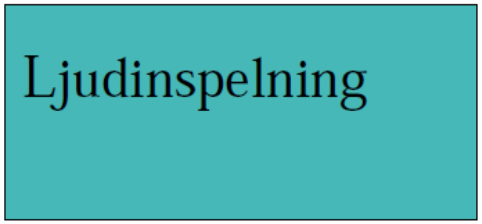
Java

Ett objektorienterat javaprogram består av ett antal objekt som ber varandra att utföra olika saker, genom att anropa varandras metoder

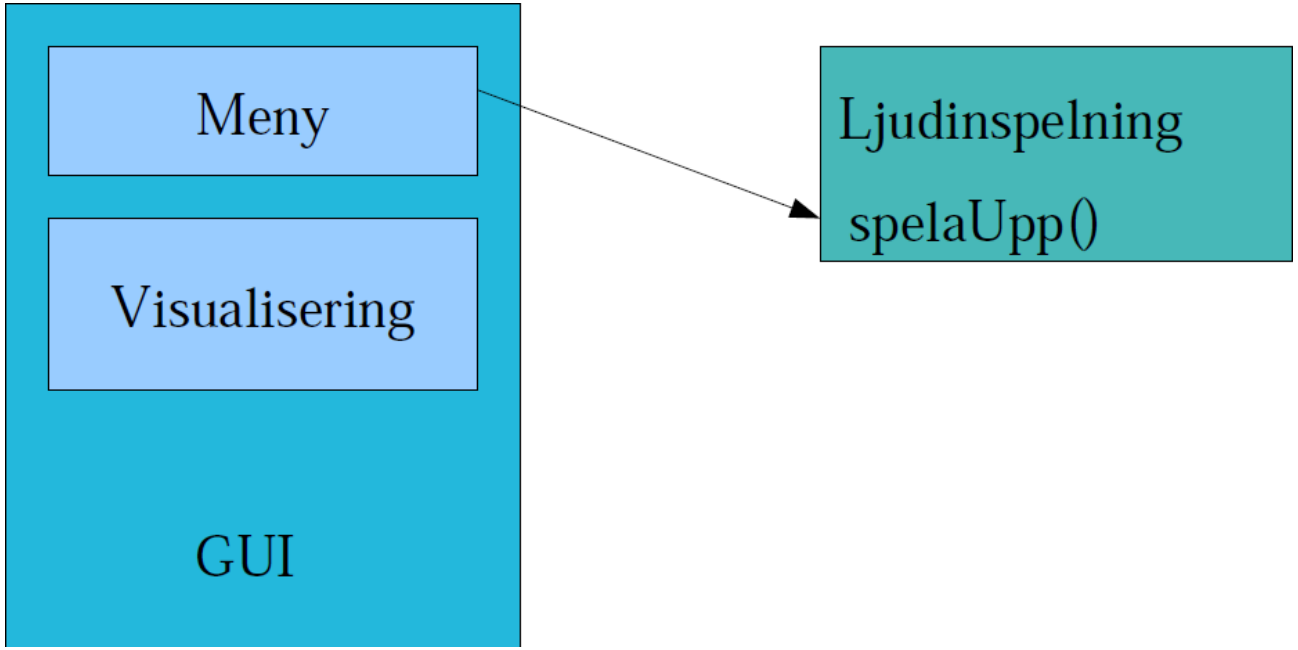

KTH
VETENSKAP
OCH KONST

Java

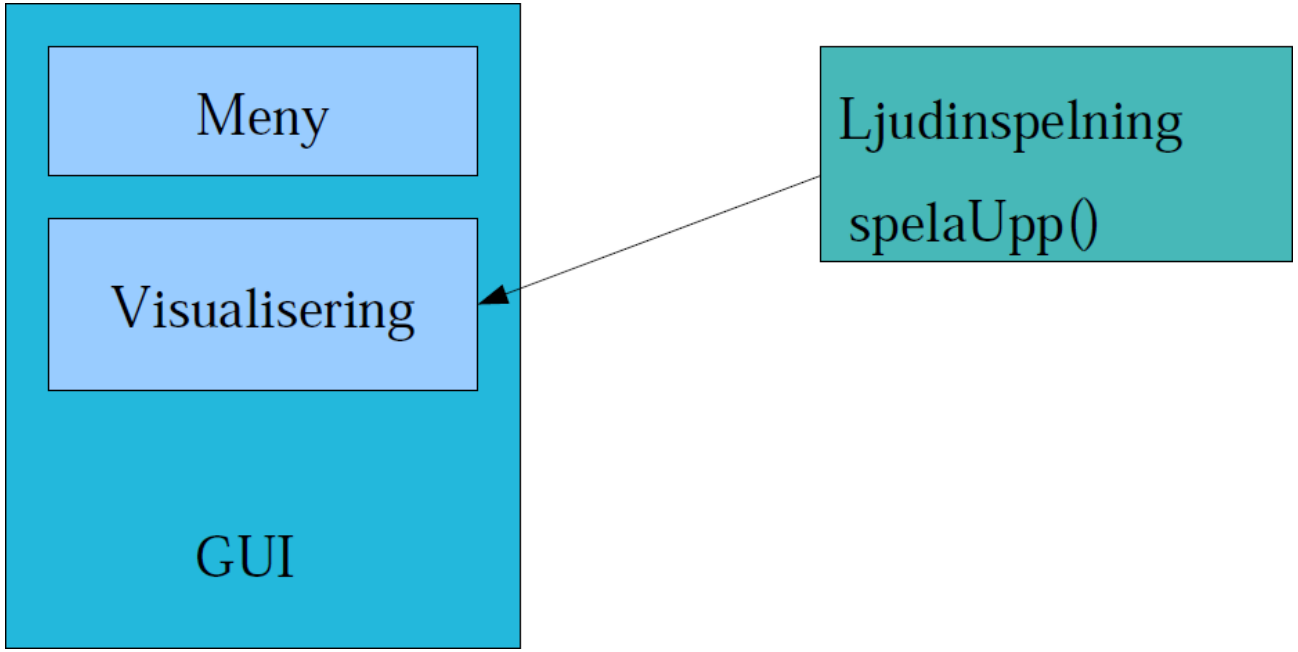

Meny
Visualisering
GUI


Ljudinspelning

Java



Java



Java

Ett javaprogram är antingen en **fristående applikation**, som kan köras som ett eget program givet att man har en **JVM** (Java Virtual Machine) installerad, eller en **applet**, som kan köras som en del av ett annat program, t.ex i en webbläsare som har stöd för detta, antingen direkt eller genom en plugin.

Javakod kompileras till s.k **bytekod** som kan köras i den virtuella maskinen, oberoende av platform. Det medgör att samma program kan köras på många olika system. Detta innebär att prestanda oftast är något sämre än för maskinspecifik kod, men att utvecklingskostnaderna blir lägre, särskilt om man vill att ens kod ska fungera överallt.

Java, variabeldeklarationer

Java är ett **typat** språk. Detta betyder att vi måste ange explicit vilken typ en viss variabel tillhör.

ex primitiva typer (dessa är inte objekt!):

double, float, boolean, int, char

```
double myNumber = 3.6;

int myInteger = 6;

boolean lovesMe = false;

char myLetter = 'c';
```

Java

Java är ett typat språk. Detta betyder att vi måste ange explicit vilken typ en viss variabel tillhör.

ex typer som är objekt:

String, Jbutton, Double, BufferedImage

```
String myText = “Min fina text!”;

JButton myButton = new JButton(“Tryck mig!”);

Double myDouble = new Double(17.5);
```

Java, arrayer

Man kan skapa **array**er av alla typer. Dessa kan behöva initieras flera gånger!

ex:

```
int[] myIntArray = {1,2,3};
```

ex:

```
JButton[] myButtonArray;

myButtonArray = new JButton[10];

myButtonArray[0] = new JButton(“En Knapp!”);
```

Java, exempel

Om vi skapar en fil som heter Complex.java, och som innehåller följande text:

```
public class Complex{
    private double re;
    private double im;

    public Complex(double real, double imag){
        re = real;
        im = imag;
    }

    public double getAbs(){
        return Math.sqrt(re*re + im*im);
    }
}
```

Så kan den kompileras med `'javac Complex.java'`, vilket

skapar filen `Complex.class`

Java

Vi kan nu använda objekt av klassen `Complex` i ett annat program såhär:

```
public class ComplexTest{

    public static void main(String[] arg){
        Complex c = new Complex(2.0, 3.1);
        double n = c.getAbs();
        System.out.println("Abs is: " + n);
    }
}
```

Där klassen ovan går att kompilera till ett körbart program eftersom den har en `main`-metod.

Slutligen

Titta på kurshemsidan:

`https://www.kth.se/social/course/DD1346/`