TDA 545: Objektorienterad programmering

Föreläsning 5:
Att använda klasser & objekt

Magnus Myréen

Chalmers, läsperiod 1, 2015-2016

I denna (och nästa) föreläsning

Läsanvisning: kap 2 & 13

- meddelanden och metoder
- informationsdöljande och inkapsling
- skapa och använda färdiga objekt !
- primitiva variabler kontra objektvariabler
- 3 tester på likhet
- metoder
- fält (arrays)

Meddelande och metoder

Man kopplar ihop funktionalitet genom att sända meddelanden:

```
fiasButik.skickaBukett(värde, adress)
```

ett objekt - mottagare av meddelandet

meddelandet vad objektet skall göra

argument dvs indata

Om objektet kan ”**`skickaBukett`**” så är det dess ansvar att utföra åtgärden – klienten vet inte hur objektet utför sysslan (informationsdöljande!).

Observera också att olika objekt kan tolka meddelandet olika:

```
minFru.skickaBukett(värde, adress)
```
- ringer sin vän

```
minBror.skickaBukett(värde, adress)
```
- ringer en butik

Informationsdöljande och inkapsling

Informationsdöljande och inkapsling stödjer 2 sätt att se på samma komponent och bygger på separation av gränssnitt (interface) och implementation:

Interface vyn	⟷	**Implementations vyn**
vad kan utföras	⟷	hur görs det

Beteendet är ”inkapslat” i klassen.

Informationsdöljandet sker med ”`private`”.

- man kan enkelt dela upp arbetet
- man kan byta ut innehållet utan att det märks utifrån (om man fortfarande uppfyller specifikationen)
- återanvändning förenklas

Hur vet man att det finns en skickaBukett? Man tittar i specifikationen.

Att deklarera objekt

Man deklarerar (dvs skapar) ett objekt(" -handtag") genom att associera ett variabelnamn med en klass.

```
Rectangle rect;
```

klassnamn dvs typnamn

variabelnamn

En referensvariabel skapas med innehållet "null" som kan innehålla adressen till en instans av klassen.

variabel, dvs
en minnesplats

typ: `Rectangle`
namn: `rect`

Att skapa objekt

Man skapar själva objektet (en instans av klassen-man ”instansierar”) genom att anropa **new** som anropar klassens konstruktor (vars namn är klassens namn).

```
ect = new Rectangle(…);
```

referensvärde som pekar på det nya objektet

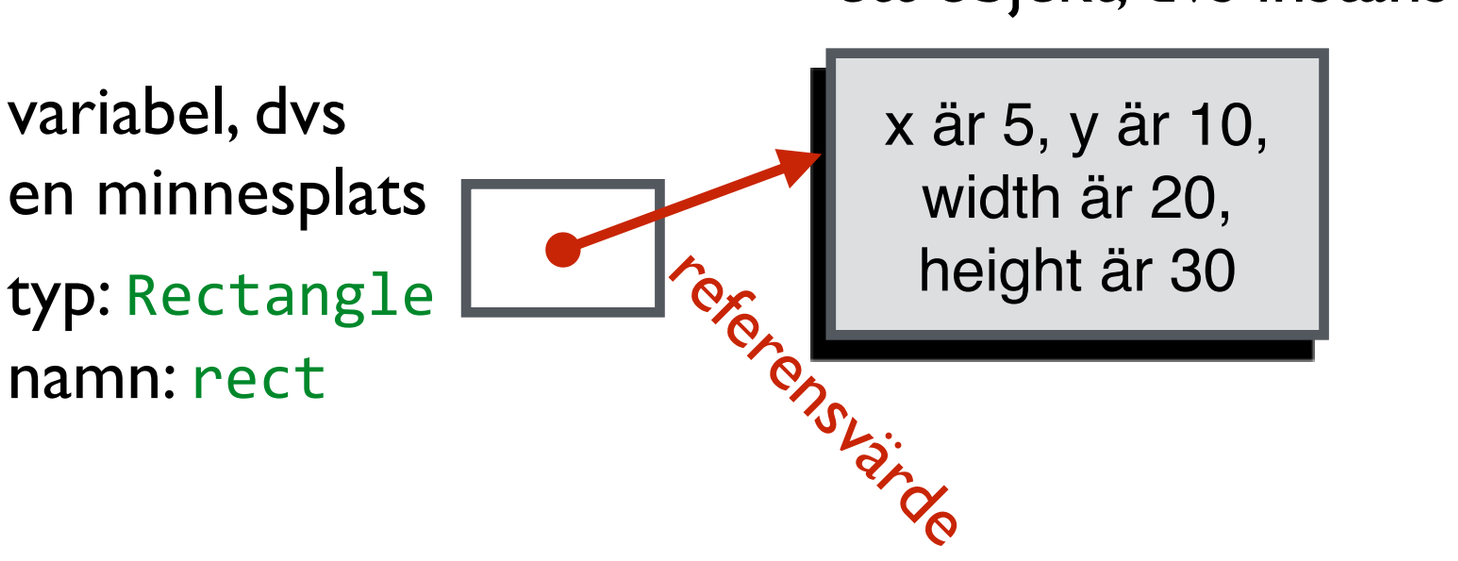
konstruktor skapar objektet

indata, hurdan Rect skall vi ha?

Exempel:

```
rect = new Rectangle(5,10,20,30);
```

new visar att vi skapar nytt objekt



Abstraktionen

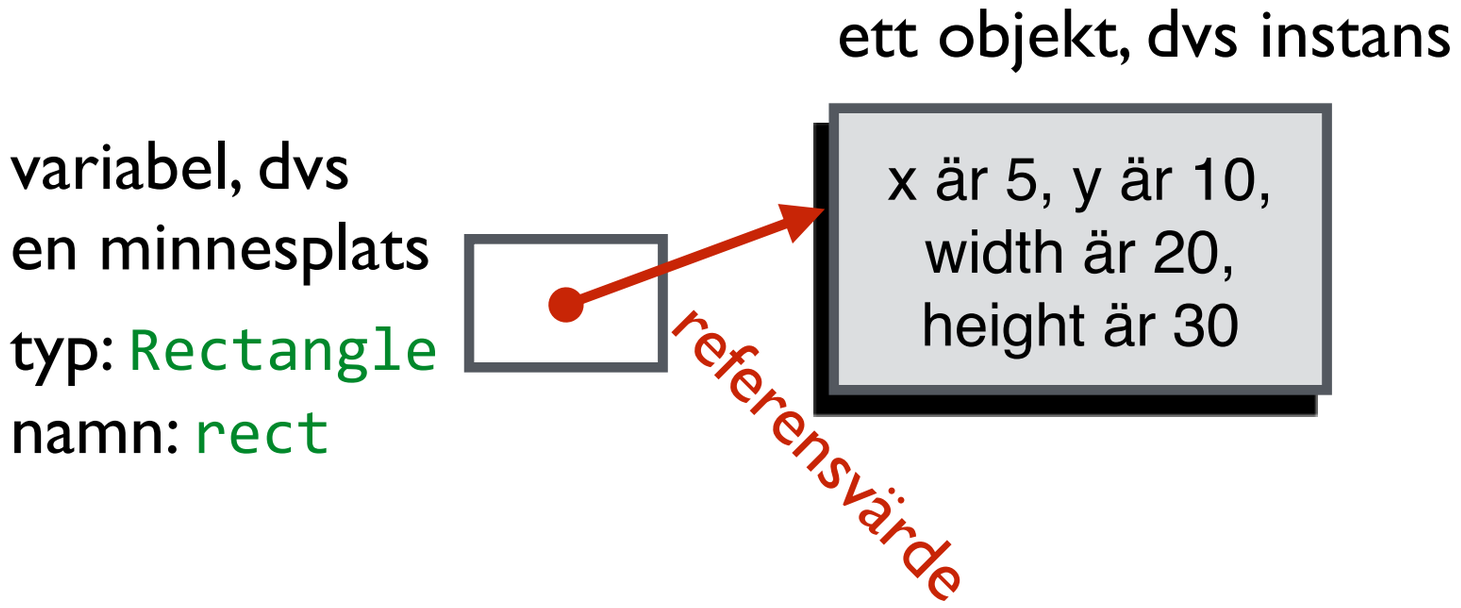
Ett rektangelobjekt är alltså inte ett rektangulärt föremål!

Det är ett objekt som innehåller
fyra tal som *beskriver* rektangeln.

Vår modell av verkligheten!

x, y är övre vänstra hörnet

Klassen **`Rectangle`** finns i paketet **java.awt** i Javas API.



Använda objekt

Det gör man genom att anropa deras metoder:

- ”skicka ett meddelande”,
- ”ställ en fråga”, eller
- ”ge ett kommando”

En Rectangle har tex metoden

metodens namn

typen av returvärdet

```
boolean contains(Rectangle r)
```

typen och namnet på parametern

Checks whether or not this Rectangle entirely contains the specified Rectangle `r`.

Använda objekt

```
Rectangle a = new Rectangle(...);
Rectangle b = new Rectangle(...);
boolean con = a.contains(b);
```

implicit parameter

explicit parameter

En Rectangle har tex metoden

`boolean contains(Rectangle r)`

Checks whether or not this Rectangle
entirely contains the specified Rectangle **`r`**.

Använda objekt

Hur vet man att det finns en **contains** metod?

Svar: man tittar i Javas API (Application Programmers Interface) där alla specifikationer finns.

Vad är egentligen en specifikation?

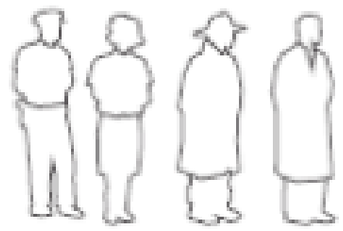
En Rectangle har tex metoden

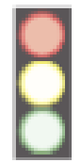
`boolean contains(Rectangle r)`

Checks whether or not this Rectangle entirely contains the specified Rectangle **`r`**.

En kö

Abstrakt datatyp
(abstrakt idé)





Specifikation
(gränssnitt)

Implementation

Syntax

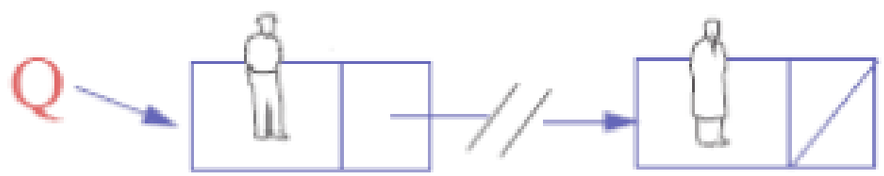
Semantik

Representation
(tillståndet)
Dvs variabler och
datastrukturer

Algoritmer
(beteende)
Dvs metoder

```
public someObject first( ) {
```

```
före: Kön är inte tom.
efter: Metoden returnerar värdet
som finns i "Current position". Anropas
operationen för en tom kö inträffar
den exceptionella händelsen
QueueIsEmpty.
```




```
public someObject first( ) {
    .....
    return .....
}
```

Specifikation / Implementation

specifikation av gränssnittet (interfacet) =

syntax och semantik för objektet

dvs en definition av ett objekts funktionalitet i termer av **hur** man skriver när man vill använda funktionaliteten och **vad** det betyder när man skriver så.

En precis beskrivning av vad en klient kan använda sig av hos objektet alltså.

implementation =

representation och algoritmen

dvs de interna variabler och metoder som möjliggör att objektet kan utföra sina ”löften”

Specifikation / Implementation

och en påminnelse:

- "The interface of a black box should be fairly straightforward, well defined, and easy to understand."
- "To use a black box, you shouldn't need to know anything about its implementation; all you need to know is its interface."
- "The implementor of a black box should not need to know anything about the larger systems in which the box will be used."

Rectangle (Java Platform SE ×

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

java.awt

Class Rectangle

java.lang.Object
  java.awt.geom.RectangularShape
    java.awt.geom.Rectangle2D
      java.awt.Rectangle

All Implemented Interfaces:

Shape, Serializable, Cloneable

Direct Known Subclasses:

DefaultCaret

```
public class Rectangle
extends Rectangle2D
implements Shape, Serializable
```

A `Rectangle` specifies an area in a coordinate space that is enclosed by the `Rectangle` object's upper-left point `(x,y)` in the coordinate space, its width, and its height.

A `Rectangle` object's `width` and `height` are `public` fields. The constructors that create a `Rectangle`, and the methods that can modify one, do not prevent setting a negative value for width or height.

A `Rectangle` whose width or height is exactly zero has location along those axes with zero dimension, but is otherwise considered empty. The `isEmpty()` method will return true for such a `Rectangle`. Methods which test if an empty `Rectangle` contains or intersects a point or rectangle will always return false if either dimension is zero. Methods which combine such a `Rectangle` with a point or rectangle will include the location of the `Rectangle` on that axis in the result as if the `add(Point)` method were being called.

A `Rectangle` whose width or height is negative has neither location nor dimension along those axes with negative

Java™ Platform Standard Ed. 7

All Classes

Packages

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

MenuBar
MenuComponent
MenuItem
MenuShortcut
MouseInfo
MultipleGradientPaint
PageAttributes
PageAttributes.ColorType
PageAttributes.MediaType
PageAttributes.OrientationRe
PageAttributes.OriginType
PageAttributes.PrintQualityT
Panel
Point
PointerInfo
Polygon
PopupMenu
PrintJob
RadialGradientPaint
Rectangle
RenderingHints
RenderingHints.Key
Robot
Scrollbar
ScrollPane
ScrollPaneAdjustable
SplashScreen
SystemColor
SystemTray

Fields

Modifier and Type	Field and Description
`int`	**`height`** The height of the `Rectangle`.
`int`	**`width`** The width of the `Rectangle`.
`int`	**`x`** The X coordinate of the upper-left corner of the `Rectangle`.
`int`	**`y`** The Y coordinate of the upper-left corner of the `Rectangle`.

Fields inherited from class java.awt.geom.Rectangle2D

`OUT_BOTTOM, OUT_LEFT, OUT_RIGHT, OUT_TOP`

Constructor Summary

Constructors

Constructor and Description
`Rectangle()` Constructs a new `Rectangle` whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are both zero.
`Rectangle(Dimension d)` Constructs a new `Rectangle` whose top left corner is (0, 0) and whose width and height are specified by the `Dimension` argument.
`Rectangle(int width, int height)` Constructs a new `Rectangle` whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are specified by the arguments of the same name.
`Rectangle(int x, int y, int width, int height)`

Rectangle (Java Platform SE ×

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

Java™ Platform Standard Ed. 7

All Classes

Packages

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

MenuBar
MenuComponent
MenuItem
MenuShortcut
MouseInfo
MultipleGradientPaint
PageAttributes
PageAttributes.ColorType
PageAttributes.MediaType
PageAttributes.OrientationRe
PageAttributes.OriginType
PageAttributes.PrintQualityT
Panel
Point
PointerInfo
Polygon
PopupMenu
PrintJob
RadialGradientPaint
Rectangle
RenderingHints
RenderingHints.Key
Robot
Scrollbar
ScrollPane
ScrollPaneAdjustable
SplashScreen
SystemColor
SystemTray

Method Summary

Methods

Modifier and Type	Method and Description
`void`	**`add`**`(int newx, int newy)` Adds a point, specified by the integer arguments `newx,newy` to the bounds of this `Rectangle`.
`void`	**`add`**`(`**`Point`**` pt)` Adds the specified `Point` to the bounds of this `Rectangle`.
`void`	**`add`**`(`**`Rectangle`**` r)` Adds a `Rectangle` to this `Rectangle`.
`boolean`	**`contains`**`(int x, int y)` Checks whether or not this `Rectangle` contains the point at the specified location `(x,y)`.
`boolean`	**`contains`**`(int X, int Y, int W, int H)` Checks whether this `Rectangle` entirely contains the `Rectangle` at the specified location `(X,Y)` with the specified dimensions `(W,H)`.
`boolean`	**`contains`**`(`**`Point`**` p)` Checks whether or not this `Rectangle` contains the specified `Point`.
`boolean`	**`contains`**`(`**`Rectangle`**` r)` Checks whether or not this `Rectangle` entirely contains the specified `Rectangle`.
`Rectangle2D`	**`createIntersection`**`(`**`Rectangle2D`**` r)` Returns a new `Rectangle2D` object representing the intersection of this `Rectangle2D` with the specified `Rectangle2D`.
`Rectangle2D`	**`createUnion`**`(`**`Rectangle2D`**` r)` Returns a new `Rectangle2D` object representing the union of this `Rectangle2D` with the specified `Rectangle2D`.
`boolean`	**`equals`**`(`**`Object`**` obj)` Checks whether two rectangles are equal.
`Rectangle`	**`getBounds`**`()` Gets the bounding `Rectangle` of this `Rectangle`.

Exempel: att flytta en rektangel

```
void translate(int x, int y)
```

Translates this Rectangle the indicated distance, to the right along the x coordinate axis, and downward along the y coordinate axis.

Ska vi skriva ett program som använder **translate**?

Hur gör vi det?

Exempel: att flytta en rektangel

```
import java.awt.Rectangle;
public class MoveTester {
   public static void main(String[] args){
      Rectangle box = new Rectangle(5, 10, 20, 30);
      // Move the rectangle
      box.translate(15, 25);
      // Print information about the
      // moved rectangle
      System.out.println("After moving,the top-left corner is:");
      System.out.println(box.getX());
      System.out.println(box.getY());
   }
}
```

skapar en ny rektangel

anropar **translate** metoden

skriver ut var den flyttade rektangeln finns

```
After moving, the top-left corner is:
20
35
```

Import satsen

```
import java.awt.Rectangle;
```

En import sats representerar en eller flera filer/foldrar där kompilatorn skall söka efter externa klasser om den inte hittar dem i samma folder som programmet som kompileras.

Dessa filer/foldrar måste anges relativt system variablerna path eller classpath.

Utan import måste vi skriva

java.awt.Rectangle box = new java.awt.Rect...

Varje gång vi refererar till Rectangle

Skall vi använda många klasser i paketet java.awt så skriver vi **import java.awt.*;**

objektvariabler och primitiva variabler

Det är skillnad på

```
int n = 25;
int m = n;
```

variabel, dvs
en minnesplats

typ: `int` **25**
namn: `n`

typ: `int` **25**
namn: `m`

och

```
Rectangle box = new Rectangle(5,10,20,30);
Rectangle box2 = box;
```

variabel, dvs
en minnesplats

typ: `Rectangle`
namn: `box`

typ: `Rectangle`
namn: `box2`

ett objekt, dvs instans

x är 5, y är 10,
width är 20,
height är 30

3 tester på likhet i Java

Två objekt är lika om dom innehåller samma "tillstånd" dvs om deras instansvariabler är lika.

Två objekt är identiska om dom är samma objekt dvs om dom pekar på samma minnesutrymme.

```
String str1 = "Kalle";
String str2 = "Hobbe";
```

Test på identitet:

```
if (str1 == str2) { ...
```

Test på likhet:

```
if (str1.equals(str2)) { ...
```

Test på 'storlek':

```
if (str1.compareTo(str2) == 0) { ...
```

kommer den ena strängen före (**<0**) eller efter (**>0**) i en ordbok? Eller är de precis på samma ställe (**0**).

Test på likhet, forts.

```
String str1 = "Hej";
String str2 = str1;
String str3 = "Hej";
```

Svar:

```
System.out.println(str1 == str2);           true
System.out.println(str1 == str3);           false
System.out.println(str1.equals(str2));      true
System.out.println(str1.equals(str3));      true
System.out.println(str1.compareTo(str2));   0
System.out.println(str1.compareTo(str3));   0
```

Test på likhet, forts.

```
String str1 = "Hej";
String str2 = str1;
String str3 = "Hej";
```

Vad händer om vi skriver "**Kalle**" i stället?

```
System.out.println(str1 == str2);
System.out.println(str1 == str3);
System.out.println(str1.equals(str2));
System.out.println(str1.equals(str3));
System.out.println(str1.compareTo(str2));
System.out.println(str1.compareTo(str3));
```

Svar:

```
true
false
true
false
0
-3
```

Att använda String

Uppgift:

Skriv kod som vänder om en sträng.

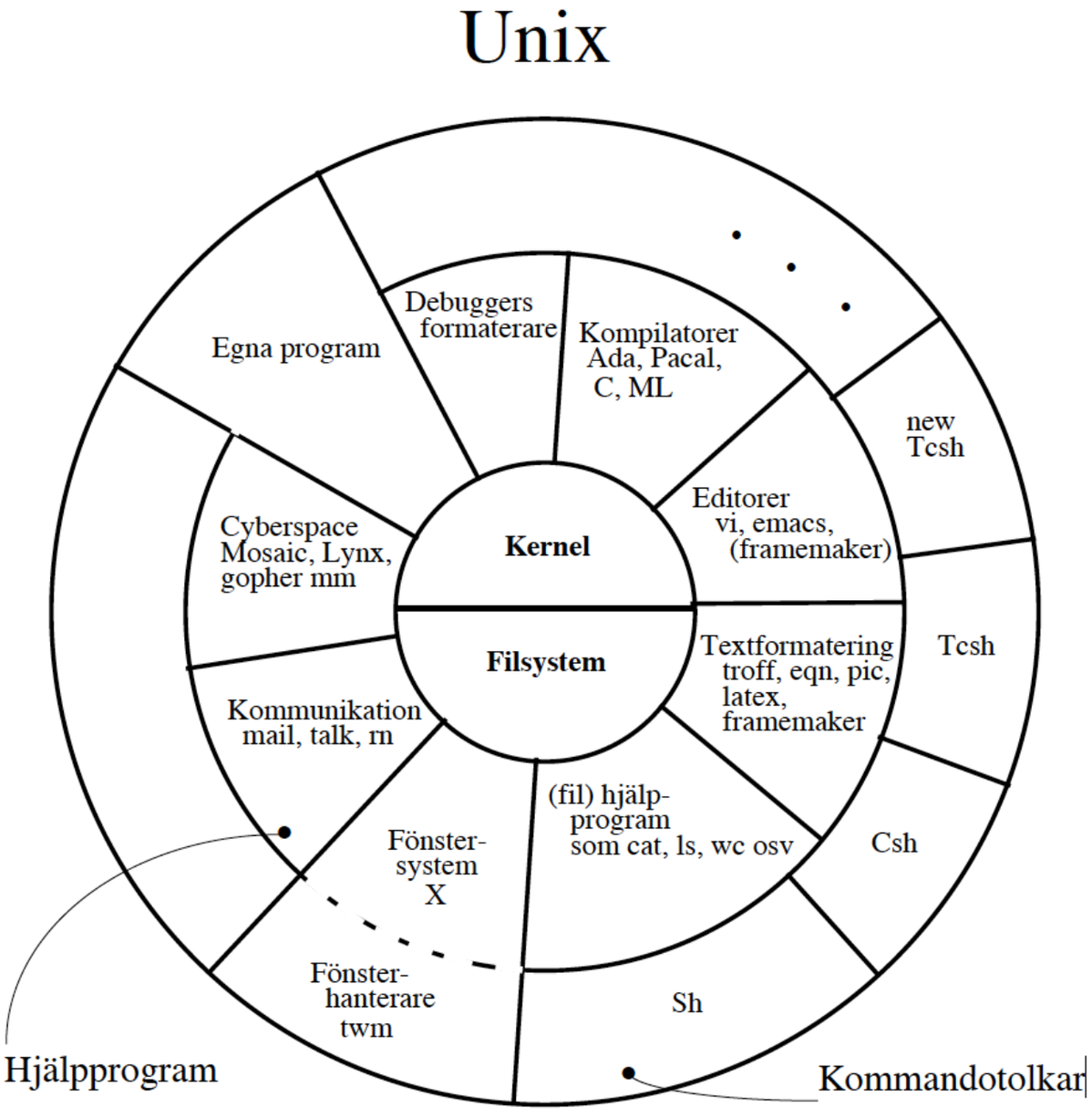
Exempel: “Hello!” bör bli “!olleH”

Att använda Scanner klassen

Uppgift:

Räkna medelvärdet av heltals input med Scanner klassen.

Unix, Emacs, mm.



- Dator
  - **Hårdvaran**
  - **Kärna** – hårdvaruberoende kod
  - **Filsystem** – ett sätt att organisera filer
    - trädstruktur med kommandon
      ls, cd, mkdir, ...
    - absolut path /<...>/<...>
    - relativ path <...>/<...>/<...>
    - filändelser <...>.java

Kärnan, filsystemet och "skalen" är ett

- **Operativsystem** –
  Unix, Linux, MacOSX,
  Windows, MSDos, Amiga,
- Fönstersystem, hela skärmen, menyer
- Terminalfönster (kommandofönster)
  emulerar en datorterminal
- xterm –en terminal emulator,
  (≈vt100)
  i den körs en **kommandotolk** som
  sh, bash, csh, tcsh, ...

Kommandotolken tolkar kommandon du skriver.

Unix, Emacs, mm.

Kommandon som styr kommandotolken (bash)

bash, tcsh, ... (byt shell)

history, ↑, ↓, !hi (historylistan)

tabbe (komplettering)

Attempt to perform completion on the text before point. Bash attempts completion treating the text as
- a variable (if the text begins with $),
- username (if the text begins with ~),
- hostname (if the text begins with @),
- or command (including aliases and functions)
- finally, filename completion is attempted.

M-? (esc-shift-?) (möjliga kompletteringar)

^c, ^z, ^d, bg, fg (processhantering)

*, ? (text expansion)

(rm -i *.class)

\ (upphäv betydelsen av följande tecken)

Förstår även programspråkssyntax

och det finns variabler som kan listas med tex

echo, env och set (olika sorters variabler)

echo $SHELL

se spec SHELL, PATH, CLASSPATH, HOME

>, <, |, >> (omdirigering)

kommando > fil (ls -l > tmp)

kommando < fil (java Rse < indata.txt)

kommando | kommando (ls -l | wc > tmp)

sort

sort fil

sort < fil

sort f1 > f2

sort < f1 > f2

sort f1 >> f2

Unix, Emacs, mm.

Kommandon som påverkar filsystemet

ls,	(lista innehållet)
	(ls -l, ls -a, ls -R /)
cd, pwd	(flytta runt i filsystemet)
mv, cp	(flytta, kopiera filer)
rm	(ta bort filer
	(rm -i <fil>, rm -r <mapp>)
	(rm -i *.class)
	(jämför med rm * .class)
mkdir, rmdir	(skapa/ta bort mappar)

Kommandon för editering, kompilering mm

cat	(lista innehållet i filer)
	(cat -te <fil>)
	(cat -b <lång fil>.txt | grep java)
more, less	
man <program>	(manualsidor)
man man	(^u, ^d, (=space), q, ↑, ↓)
sort	(sortering av filer)
diff	(hitta skillnader i filer)
grep	(sök i filer)
lpr, enscript	(utskrifter)
emacs	(välkänd editor)

Moderna texteditorer för kod

eclipse

https://eclipse.org/home/index.php

Sublime Text 2

http://www.sublimetext.com/

GNU Emacs

http://www.gnu.org/software/emacs/

Notepad++

https://notepad-plus-plus.org/

http://www.vim.org/