IDE gebruik en introductie Java

Programmeertechnieken, Tim Cocx

Universiteit
Leiden
The Netherlands

IDE Gebruik

Universiteit
Leiden
The Netherlands

Discover the world at Leiden University

Wat we al gezien hebben

- Het runnen van code
- Meerdere ‘project’ files
- Libraries
- Make files
- Linking
- Debugging

Wat we misschien gezien hebben

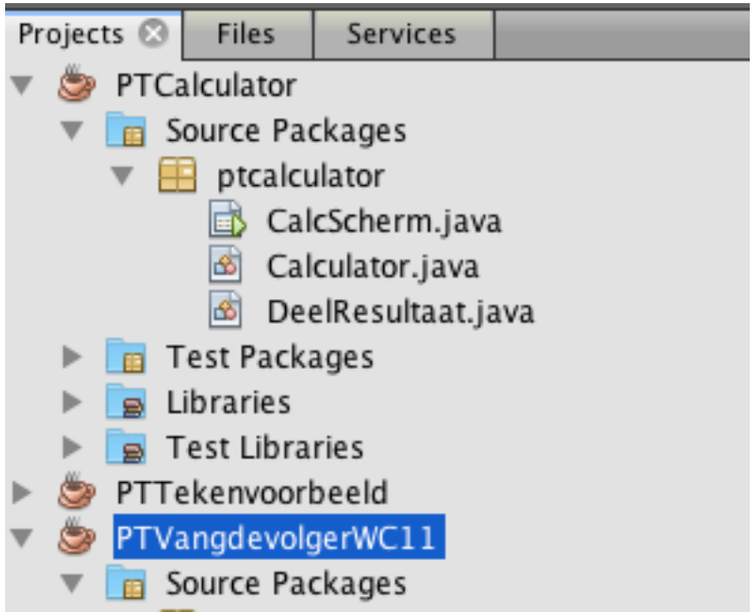
- Syntax highlighting
- Een compile/play-knop

Wat we gaan gebruiken

- Dit alles geïntegreerd
  - Plus extra's
- Integrated Development Environment (IDE)
  - Bevat alle (de meeste) tools die je gebruikt bij het maken van een applicatie.
  - Er zijn veel verschillende soorten en maten (merken, licensies, programmeer-/scripting-talen)

IDE: Project File structure

- Een IDE biedt je overzicht over je hele project door de file-tree voor je weer te geven:

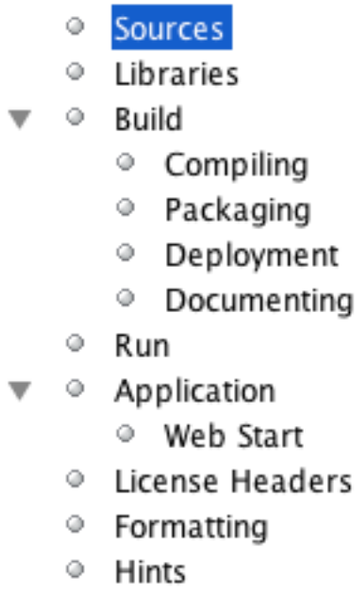


IDE: File operations

- In een IDE kan je makkelijk
  - nieuwe projecten maken
  - Files aan je project toevoegen
  - Files saven etc.

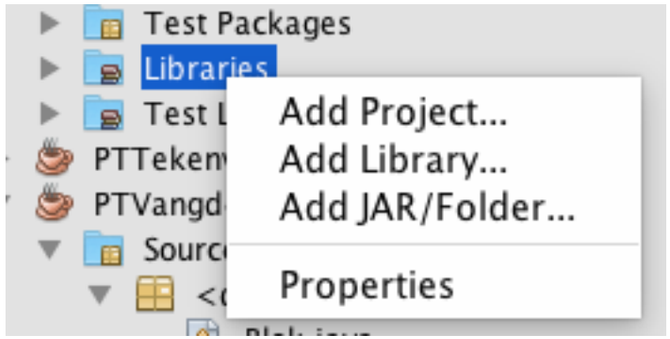
IDE: build settings

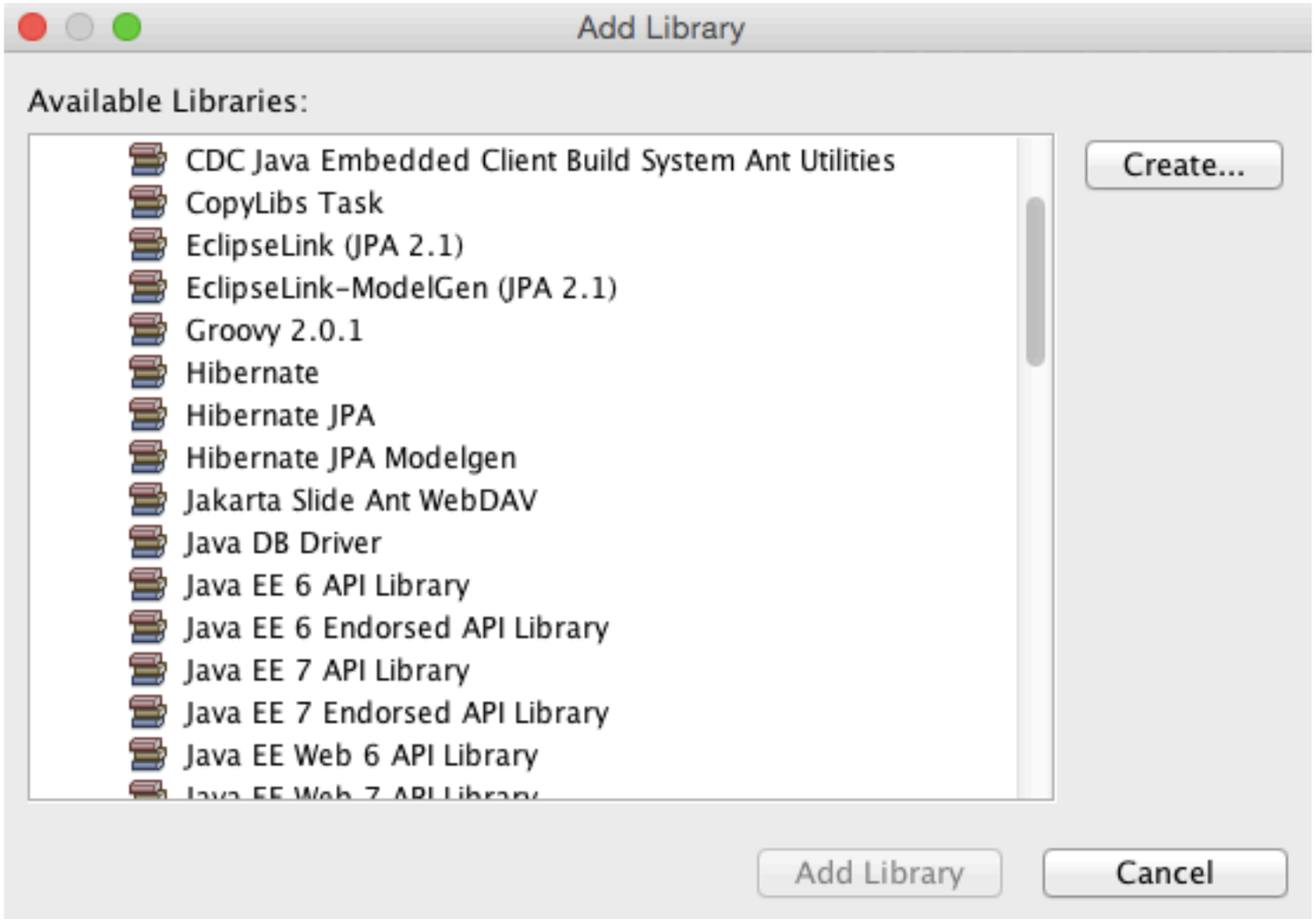
- Een IDE bepaalt automatisch welke files, hoe ssamenwerken in je project
  - Dus: automatische make-file
  - Dus: automatische linking
  - Meestal: support voor verschillende build formats (debug/ release (optimized))



IDE: library linking

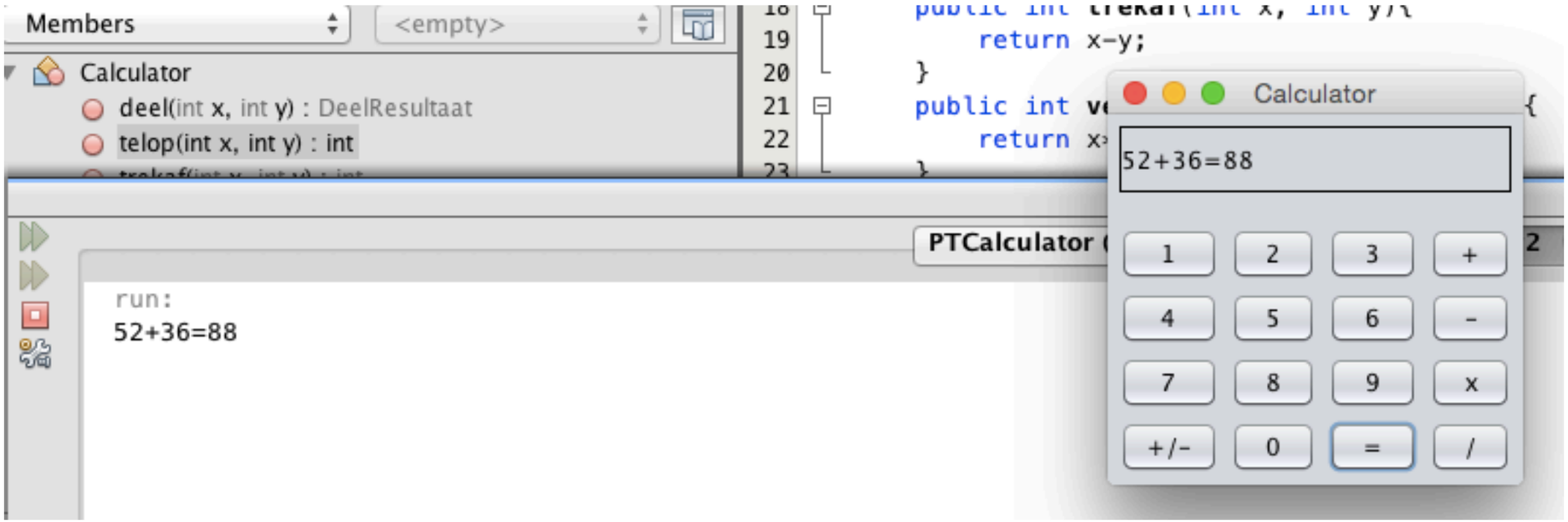
- In een IDE kan je meestal simpel tegen libraries linken door ze uit een lijst te selecteren
  - De ‘makefile’ wordt aangepast





IDE: integrated console & GUI

- In een IDE kan je je project runnen en de output bekijken BINNEN de IDE
  - Ook de gebruikers input geef je in de IDE
  - Je hebt dus maar 1 programma open



IDE: Source control: automatic indentation

- In een IDE spring je meestal automatisch in
- Accolades worden meestal ook op de goede plek gezet
  - Je ziet vaak ook de sluitaccolades!
  - Sluitaccolades worden vaak automatisch toegevoegd
- Vaak worden spaties toegevoegd om de code leesbaarder te maken

```
public class temp {

}
```

```
public class temp {
    public void temp(){

}
```

```
public class temp {

}
```

```
public class temp {
    public void temp(){

    }

}
```

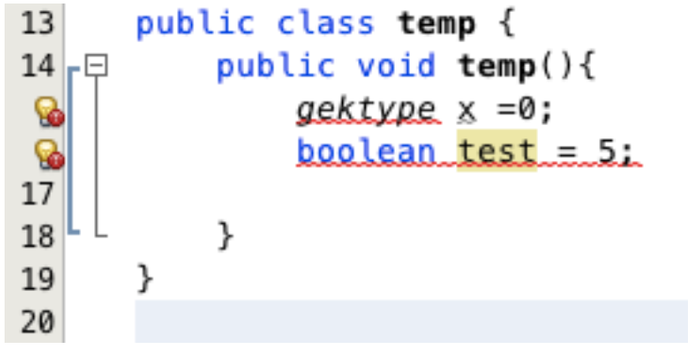
IDE: Source control: Syntax highlighting

- In een IDE zijn je codes herkenbaar gekleurd

```
public class Calculator {
    public int telop(int x, int y){
        System.out.println("Test");
        return x+y;
    }
```

IDE: Source control: error detection

- Een IDE laat zien wanneer een bepaald stuk code (waarschijnlijk) fout is.
  - Een echte fout
  - Warning
  - Niet gebruikt
  - Iets vergeten (exception handling etc).



IDE: Source control: group tabbing, group comment

- In een IDE kan je een heel stuk code ineens inspringen of (on-)becommentariëren

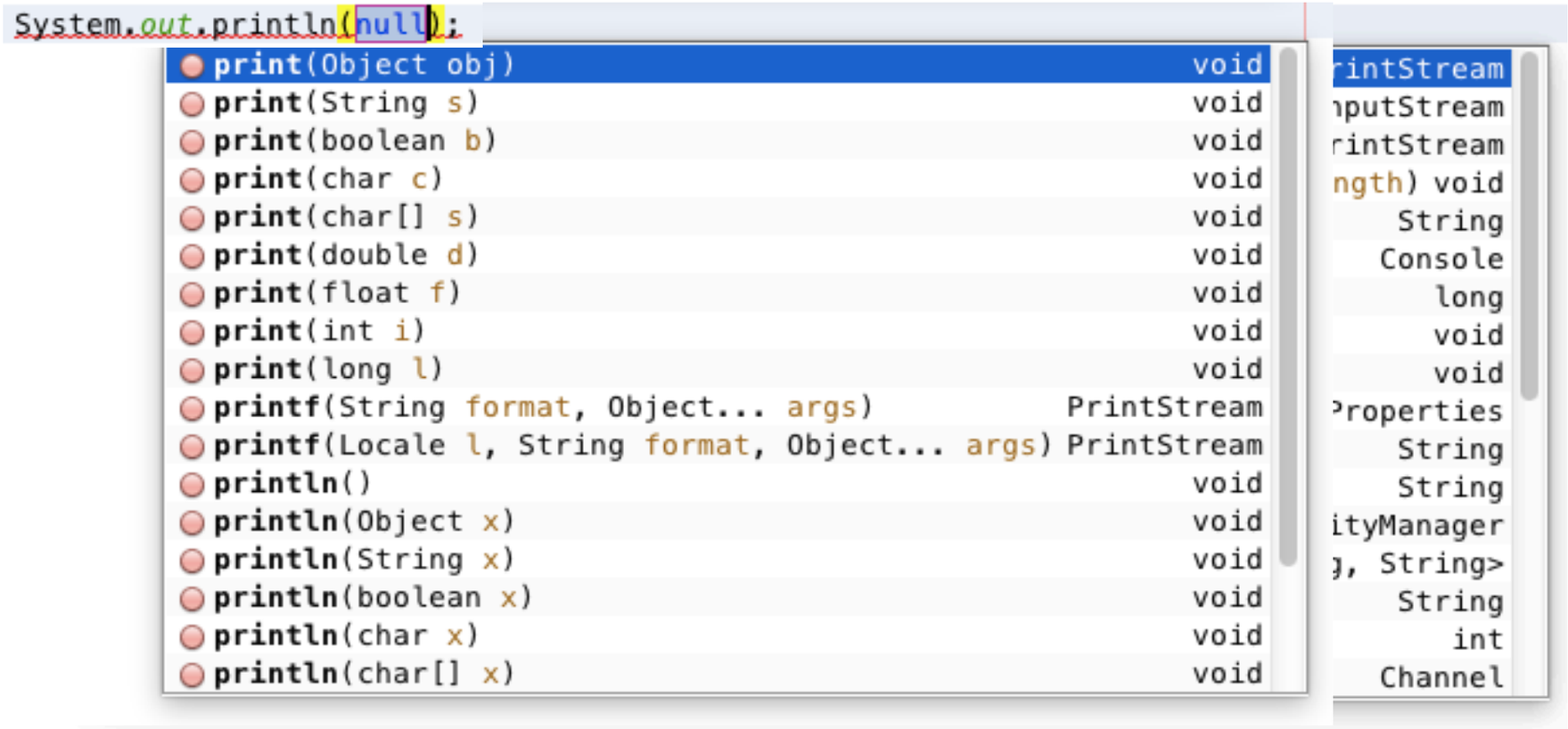
```
public class temp {
    public void temp(){
    int x =0;
    boolean test = true;
    if (test){
        x=6;
    }
    System.out.println(x);
    }
}
```

```
public class temp {
    public void temp(){
        int x =0;
        boolean test = true;
        if (test){
            x=6;
        }
        System.out.println(x);
    }
}
```

```
public class temp {
    public void temp(){
//          int x =0;
//          boolean test = true;
//          if (test){
//              x=6;
//          }
//          System.out.println(x);
    }
}
```

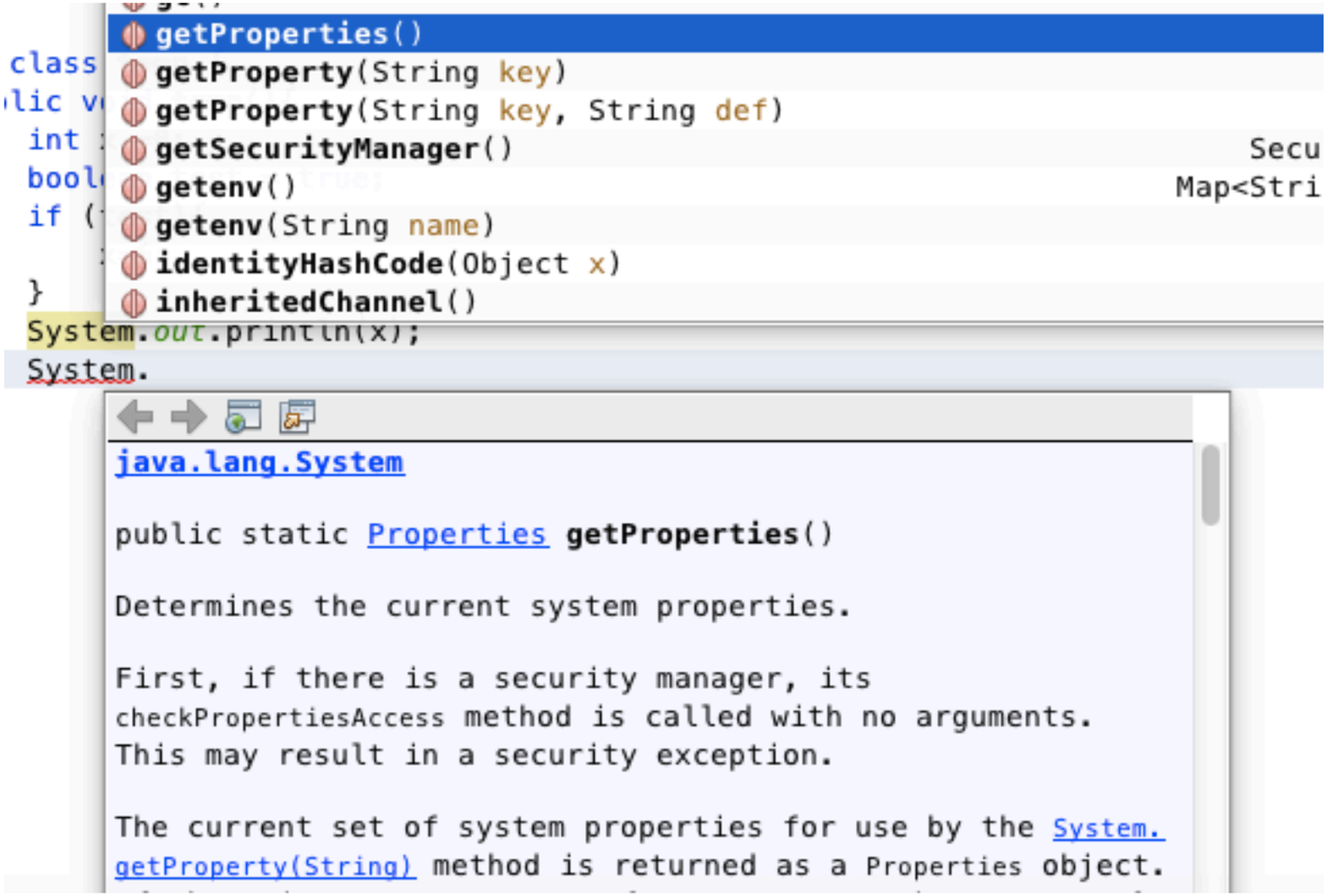
IDE: Source control: Code Completion

- Een IDE kan je suggesties geven over wat je wellicht bedoelt:
  - En kan vervolgens je code ‘compleet maken’



IDE: Source control: Documentatie

- In een IDE kan je zien wat standaard functies precies doen



IDE: Source control: Code snippets

- Voor veel voorkomende opdrachten heb je code snippets: manieren om snel het gewenste commando tevoorschijn te toveren

IDE: Source Control: Code formatting

- Mocht je code toch een rotzooi zijn geworden…
  - Of heb je rotzooi code van een ander gekregen !!
  - Dan kun je in een keer je code ‘fixen’

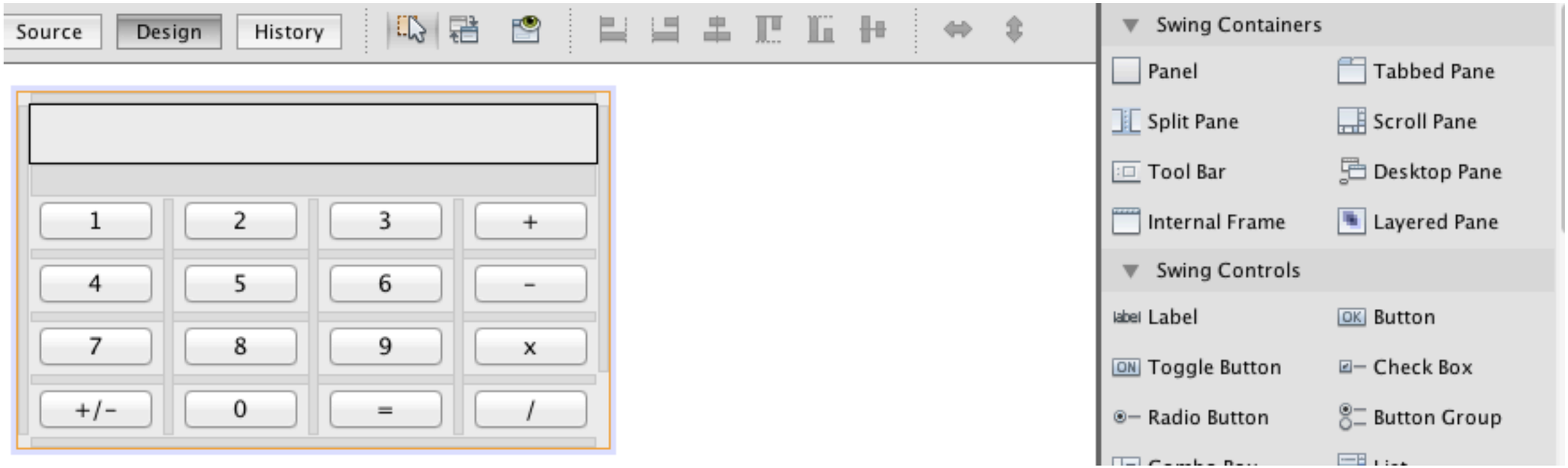
```
public class temp {public void temp(){
  int x=      0;
                boolean test=true;
        if(test)
        {
                        x=6;}
   System.out.println(x);
                System.out.println("");
            }}
```

```
public class temp {

    public void temp() {
        int x = 0;
        boolean test = true;
        if (test) {
            x = 6;
        }
        System.out.println(x);
        System.out.println("");
    }
}
```

IDE: WYSIWYG GUI editing

- Een IDE beschikt meestal over een wysiwyg editor voor het maken van een mooie GUI
  - GUI: Graphical User Interface
  - WYSIWYG: What You See Is What You Get

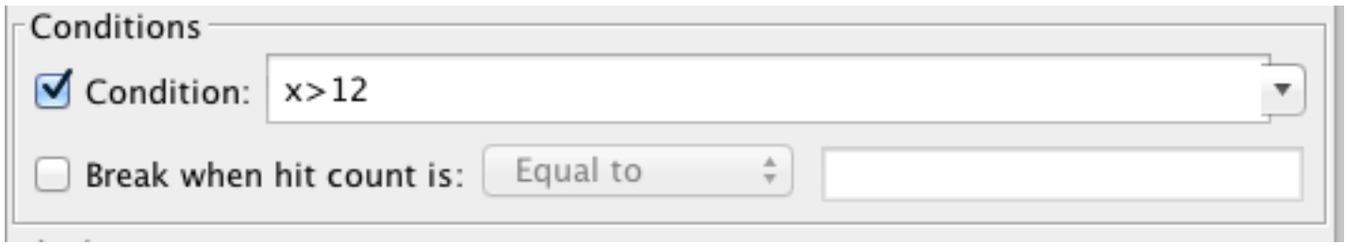


IDE: Debugging: Breakpoints

- In een IDE kan je makkelijk brakepoints aanzetten
  - En uitzetten tijdens het debuggen!

```
21      public int vermenigvuldig(int x, int y){
            return x*y;
23      }
```

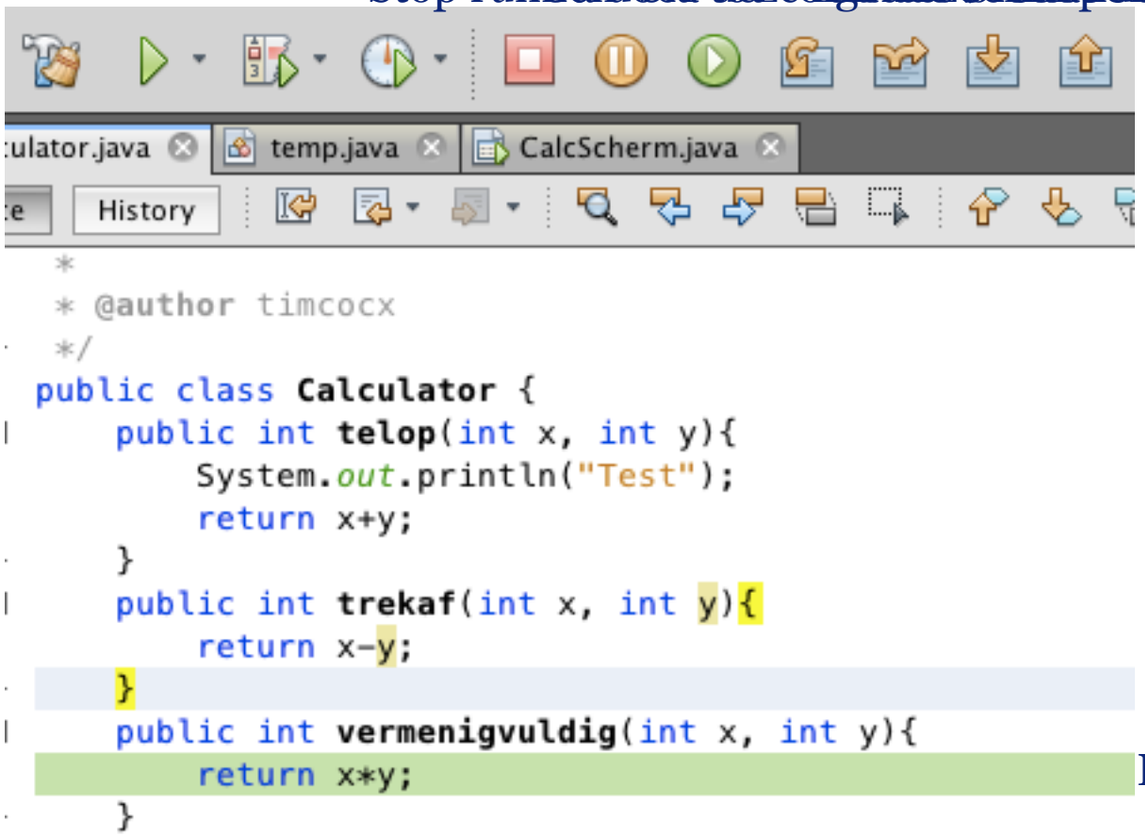
  - Deze breakpoints kan je ‘conditional’ maken



IDE: Debugging: Graphical Debug control

- In een IDE kan je grafisch je debugproces controleren en zien hoe je door de code loopt

Stop runnen

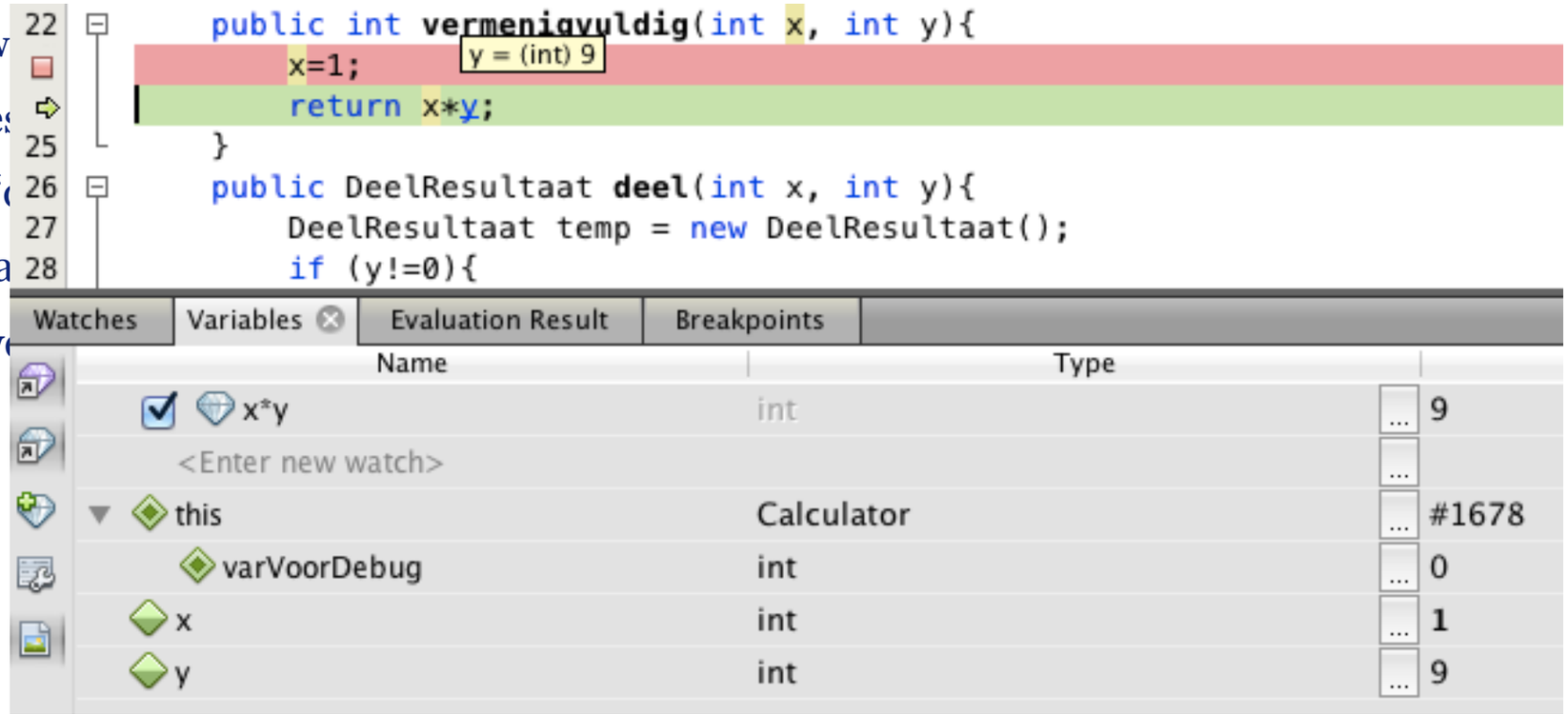


functie

Hier ben je

IDE: Debugging: Watches

- Tijdens debuggen kan je de huidige waarde van een variabele zien
  - Door eroverheen te hoveren
  - Door bij de w
- Bij de watches
- Je kan soms '
- Je kan zelf wa
- Je kan zien w



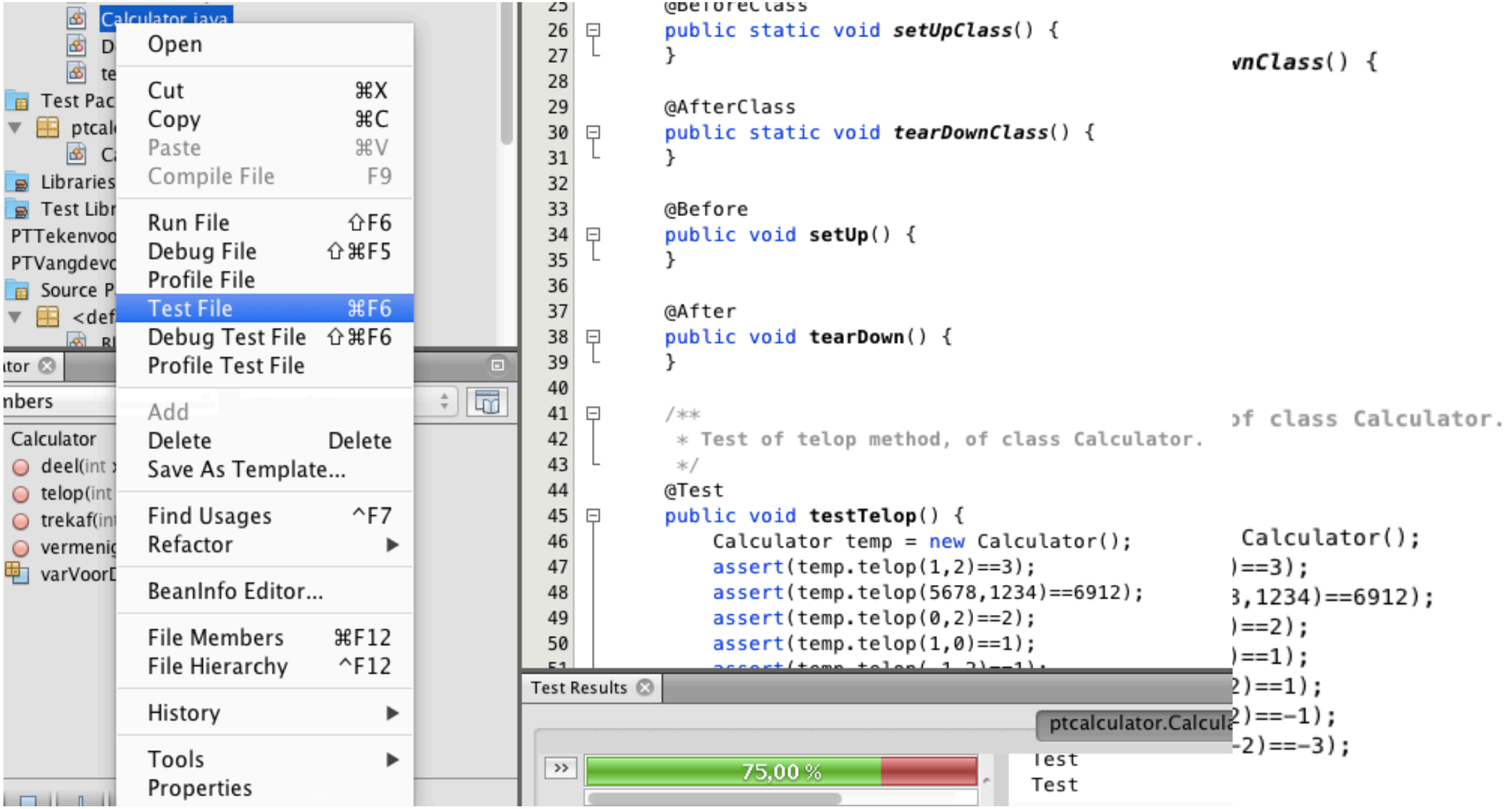
IDE: Debugging: run-time error

- Ook als je niet aan het debuggen bent, maar toch een crash hebt kan je doorklikken naar de plek van de error

```
Exception in thread "AWT-EventQueue-0" java.lang.ArithmeticException: / by zero
        at ptcalculator.Calculator.vermenigvuldig(Calculator.java:23)
        at ptcalculator.CalcScherm.knopIsActionPerformed(CalcScherm.java:354)
        at ptcalculator.CalcScherm.access$1400(CalcScherm.java:16)
        at ptcalculator.CalcScherm$15.actionPerformed(CalcScherm.java:207)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:2018)
```

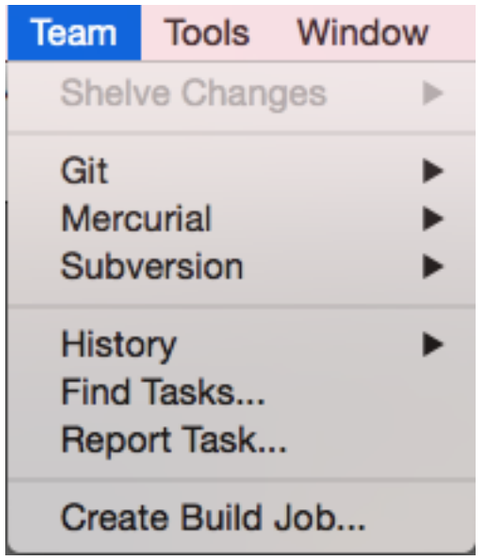
IDE: Unit testing

- In een IDE kan je makkelijk unit tests aanmaken en uitvoeren



IDE: Versioning

- In een IDE kan je makkelijk versiebeheer doen
  - Waarover later meer bij het vak Software Engineering



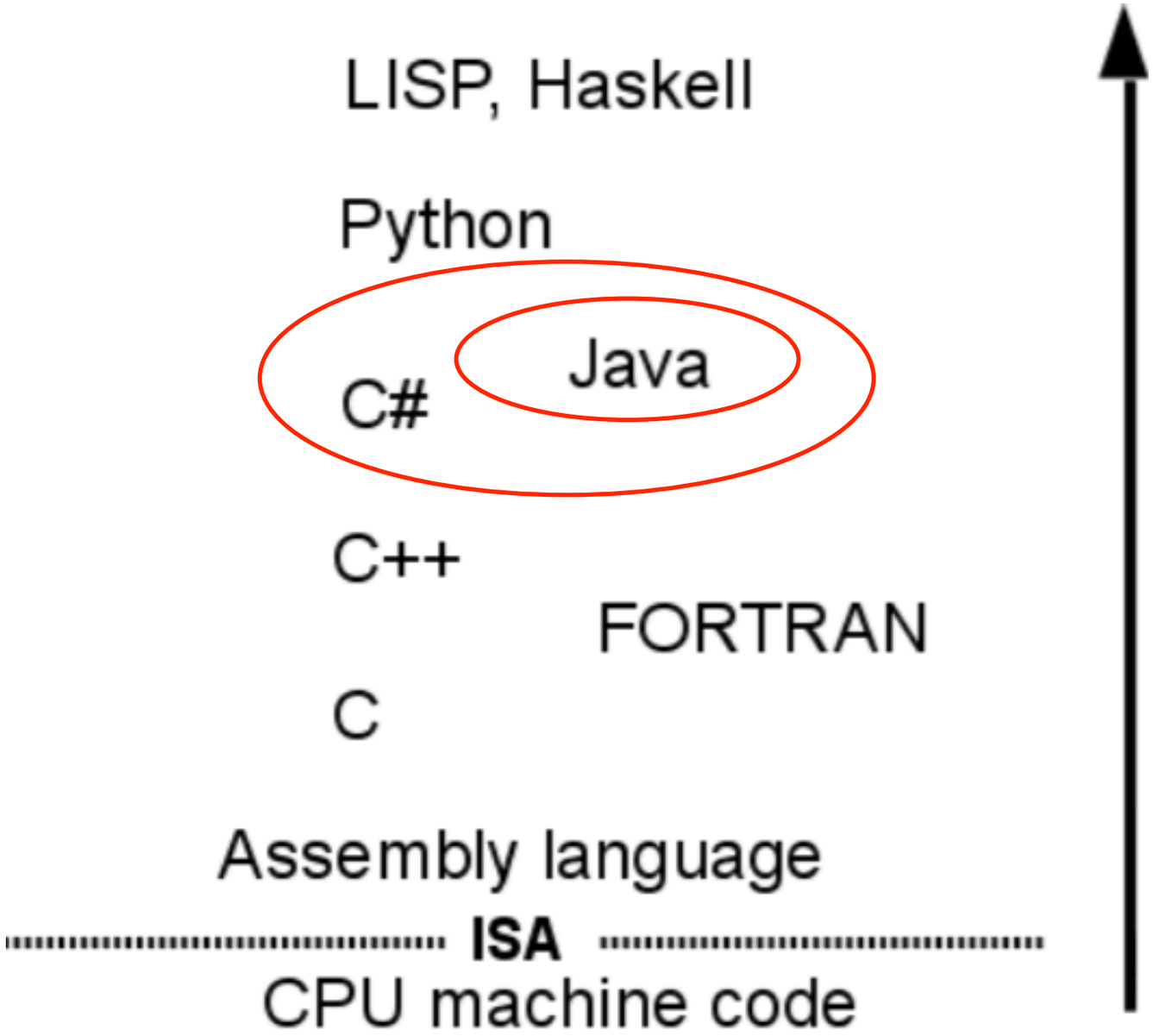
Introductie Java

Universiteit
Leiden
The Netherlands

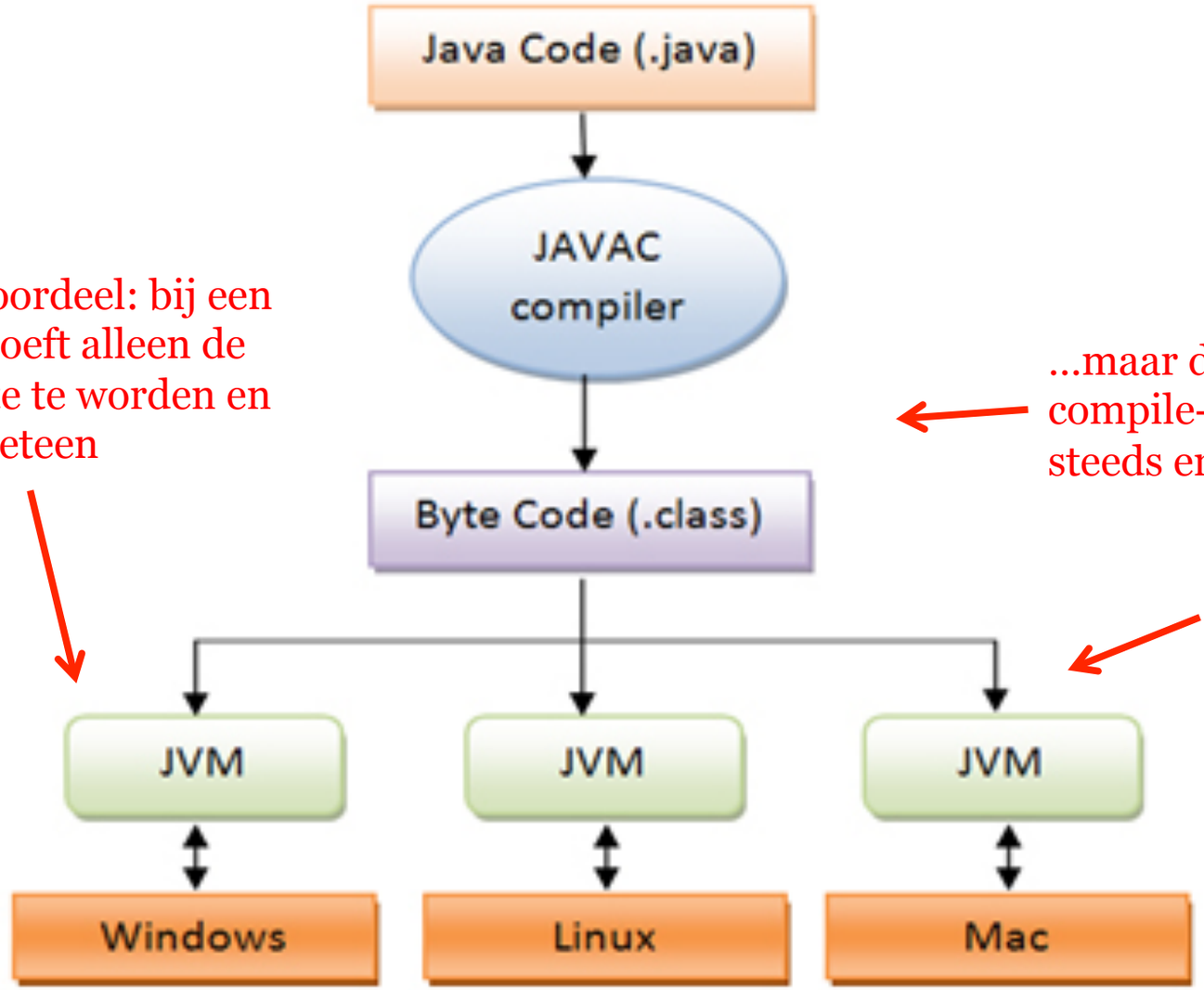
Discover the world at Leiden University

Programmeertalen

- Deze programmeertalen geven je genoeg kracht om je (operating) systeem te gebruiken
  - Je kan er GUI's mee maken
  - Je hebt precieze controle over je variabelen
  - Je kan er executables mee maken
  - -etc..
- Maar er wordt geabstraheerd van bepaalde 'details'
  - Geen memory management meer nodig
  - (of mogelijk!)
  - Geen OS afhankelijke executables meer
  - Alle code 'vindt plaats' binnen een klasse / framwork/ namespace
  - (geen losse functies meer)
  - Geen headerfiles
  - Etc...
- Kortom: makkelijker gebruik, maar iets minder flexibel / krachtig dan bijv. C++



Java Virtual Machine



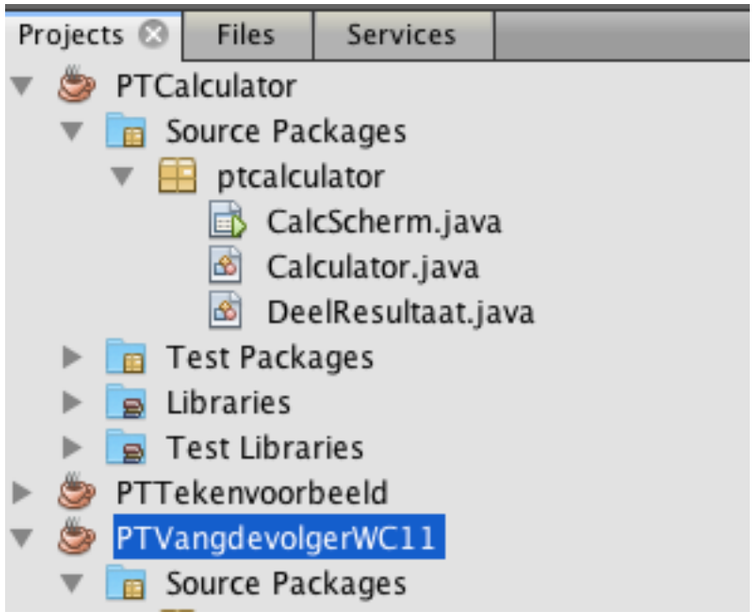
Bijkomend voordeel: bij een taal-update hoeft alleen de JVM geupdate te worden en alles werkt meteen

...maar door een efficiënte compile-slag blijft het nog steeds erg snel...

Er is een interpreter...

Packages

- In Java zijn source files opgedeeld in packages
  - Vergelijkbaar met C++ namespaces
  - Toegang (encapsulatie) is anders binnen en buiten het package.



Alles een class

- In Java is alles een class
  - Er zijn geen functies, globalen, etc. buiten een klasse
  - Er moet dus overal een object van gemaakt worden
  - Uitzondering: static methoden kunnen aangeroepen worden zonder dat er een object gemaakt wordt
  - Er kunnen dan geen attributen gebruikt worden
  - Voorbeeld: public static void main()
- Alle klassen staan in een eigen bestand
  - .java
- Ook een enum is een apart soort klasse
  - En dus in een eigen file

```
public enum Richting {
    NOORD, ZUID, WEST, OOST;
}
```

```
public class JavaApplication24 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

Geen header files

- In Java kent men geen header files
  - Dat is gewoon extra werk
  - De ‘interface’ wordt gescand tijdens compileren
- Declaratie en definitie vindt plaats ‘inline’:

- Syntax dingen:
  - public / private er altijd voor
  - bool → boolean
  - string→ String
  - String wel standaard ‘aanwezig’
  - Overerving→extends
  - Parent constructor→super() (evt met parameters)

```
public class Machinist extends Mens{
    private int lengte;
    private String naam;
    private double salaris;
    private boolean getrouwd;

    public Machinist(){
        //constructor (altijd public)
        super();
    }

    public void rijden(int snelheid){
        //tjoeke tjoeke
    }

    private void trouwen (Machinist partner){
        //feestje
    }

}
```

Objecten

- Objecten worden altijd aangemaakt met ‘new’ en ’constructor-haken’ (evt met parameters)
- Je kan een child-class aanmaken bij een parent class (statisch type).
- Ja kan het dynamische type (daadwerkelijke objecttype) checken met instanceof
- Je kan objecten makkelijk casten
  - Let op haakjes
- Syntax dingen:
  - const→final

```
public static void main(String[] args) {
    final int MAXMACHINIST = 12;

    Machinist tim=new Machinist(); // Machinist Tim; Machnist* tim = new Machinist()
    Mens kris = new Machinist();   // mag ook
    if (kris instanceof Machinist){
        ((Machinist)kris).rijden(40); // moet wel casten & kan mooier
    }

}
```

Virtualiteit

- In Java zijn alle methoden ‘virtual’
  - Wat betekent dat je ze kan overschrijven in een kind-klasse
  - Om te voorkomen dat dat per ongeluk gebeurt moet je dan @override toevoegen
  - Anders: warning
  - Bij runnen wordt altijd de laagst mogelijk methode uit de hiërarchie gepakt mbt dynamische objecten.

```
public class Mens {
    public void praten(){
        //blabla
    }
}
```

```
@Override
public void praten(){
    //tuuttuut
}
```

Parameter handling

- In Java worden alle parameters hetzelfde meegegeven
- Alle primitieve types (double, int, boolean, etc): call-by-value
  - Als je dat niet wil: ‘wrappen’ in klasse
- Alle andere types: call-by-reference
  - objecten
  - Arrays
  - String!
- Dit is vrijwel altijd ook wat je wil.

CBV CBR

```
public void salarisVerhoging(double verhoging, Machinist werknemer) {

}
```

CamelCase standaard

- Java hanteert (de facto) altijd de CamelCase standaard
  - Als woorden aan elkaar staan begint elk volgende woord met een hoofdletter
- Classes: Upper CamelCase: ook eerste letter is hoofletter
- Methoden / variabelen: lower CamelCase: eerste letter is klein
- Hiermee herken je makkelijk het verschil
  - Bv: CBV / CBR
  - Bv: is er extra functionaliteit
  - String!

```
public class ApenPark {
    public final int MAXKINDEREN = 4;
    public int aantalKinderen;
    public Machinist[] vriendenVanPark = new Machinist[50];

    public void krijgKinderen(int hoeveelKids){

    }

}
```

- Verder:
  - Een final schrijf je met ‘all-caps’
- Syntax dingen
  - De declaratie van een array is anders: (soort van object)

Input / output, import

- In Java gaat input met System.out.println();
  - out & println zijn static bij standaard Java klasse System!!
- In Java gaat input met System.in
  - Meestal ge-wrapped door Scanner om omgang met ruwe input te voorkomen
  - Deze library moet je wel importeren
  - Dat is makkelijk in een IDE☺ (rechtsklik op error→fix imports)

```
Scanner scan = new Scanner(System.in);
public void krijgKinderen(int hoeveelKids){
    System.out.println("krijg nou kinderen!");
    int hoeveel = scan.nextInt();
}
```

- Importeren (Java) is wat anders dan includeren (C++)
  - Het systeem is hiërarchisch (herkenbaar wat er gebeurt)
  - Alles importeren kan ook met bijv. import java.util.*
  - Niet alles wordt daadwerkelijk included
  - Alleen wat je gebruikt wordt meegelinkt; de rest neemt geen ruimte op in je executable

```
import java.util.Scanner;
```

Object class

- In Java erft elke klasse automatisch over van de klasse Object
- Deze heeft 1 methode: public String toString();
  - Je kan hierdoor elk object afdrukken: de methode toString() wordt automatisch aangeroepen
- Omdat Java natuurlijk niet precies weet welke info van een object moet worden afgedrukt maakt hij er het beste van
  - Op basis van de typen van variabelen of de bitstream (ofzo)
- Je kan dus het beste de methode toString() overriden
  - Bij afdrukken wordt de laagst mogelijke toString() (dynamisch type) automatisch aangeroepen
  - Wat is object oriëntatie toch mooi! ☺

toString()

```java
public class EensZien {

    public int een;
    public double twee;
    public boolean drie;

    public EensZien() {
        een = 42;
        twee = 3.14;
        drie = false;
    }
}
```

```java
EensZien test = new EensZien();
System.out.println(test);
```

run:

javaapplication24.EensZien@4f14e777

BUILD SUCCESSFUL (total time: 0 seconds)

toString()

```java
public class EensZien {

    public int een;
    public double twee;
    public boolean drie;

    public EensZien() {
        een = 42;
        twee = 3.14;
        drie = false;
    }

    @Override
    public String toString(){
        return ("een: "+ een + ", twee: " + twee + ", drie: "+ (drie?"Ja":"nee"));
    }
```

```java
EensZien test = new EensZien();
System.out.println(test);
```

Kenden jullie de short-hand-if al?

run:

een: 42, twee: 3.14, drie: nee

BUILD SUCCESSFUL (total time: 0 seconds)

Garbage collection

- Java heeft zijn eigen memory management: garbage collection
  - Geheugenlekken kunnen (in principe niet meer)
  - Alles wordt automatisch weggegooid als er niet meer aan gerefereerd wordt
  - Dat gaat ‘cascading’
  - Dus ook geen destructor nodig
  - Dus pointers (als zodanig begrip) bestaat niet in Java
  - (al zijn eigenlijk alle variabelen die een object of array bevatten stiekem een pointer, maar dat hoef je niet te weten)
- Voordeel: makkelijk!!
- Nadeel:
  - Je kan er niet meer expliciet voor kiezen iets weg te gooien (is soms wat onhandig)
  - Advanced/ direct memory manipulation is niet mogelijk (als je dat zou willen)

Nu:

- Maak als oefening de rekenmachine in NetBeans met behulp van de library Swing.
- Ga daarna (volgende week?)(zo snel mogelijk) aan de gang met opdracht 3: Vang de Volger

- Om “netbeans” op te starten:

$ source /vol/share/groups/liacs/scratch/pt2016/pt2016.env

$ netbeans