

Föreläsning 2

- Presentation av kursen
- Att programmera
- Java och utvecklingsmiljöer
- Ett program

JF: Kap 1

Programmering 1, DA211T

Kursansvarig och lärare: Rolf Axelsson
040-6657681, rolf.axelsson@mah.se, E145

Lärare: Kristina Alder
040-6657132, kristina.alder@mah.se, E131

Studieadministratör: Bodil Sterner
040-6657620, bodil.sterner@mah.se

Kursens hemsida:

- It's learning:

mah.itslearning.com/elogin/

Här hittar du bl.a. kursmaterial, kursplan, schema och kursplanering

- Vecka 36 och 37:

<http://edu.mah.se/sv/Course/DA211T#Syllabus>

Välj "Innehåll"



Om Programmering 1

- **Laborationer** ska redovisas. Du ska visa labhandledaren att du gjort laborationen.
Mot slutet av kursen är det fyra projektförberedande laborationer.
- Du ska lösa och redovisa **5 inlämningsuppgifter**. Du ska samtidigt granska andra studenters lösningar.
- Kursen avslutas med **skriftlig tentamen**. Tentamen är betygsgrundande.

Att göra ett program

- **Uppgiftsformulering**, vad är det för uppgift som ska lösas? Formulera uppgiften i termer av vad en dator kan utföra. Avgränsa problemet, vad är en del av uppgiften? Vad ingår inte?
- **Algoritmkonstruktion**, vilka algoritmer är de mest lämpliga för detta problem? Konstruera strukturen på programmet och skriv ner så kallad pseudokod. Detta är kreativ problemlösning.
- **Kodning**, översätt pseudokoden till ett programmeringsspråk t.ex. Java
- **Dokumentation**, beskriva din lösning både i löpande text, med hjälp av UML och som kommentarer i programmet (t.ex. javadoc).
- **Testning**, är programmet byggt på ett bra sätt så att det löser uppgiften utan att fel uppstår?

Programmeringspråk

Ett programmeringsspråk är ett antal regler (syntax) och dess betydelser (semantik) i vilket en programmerare kan uttrycka sig och ge instruktioner till en dator.

Programmerare skriver instruktioner (text) i programmeringsspråket. Instruktionerna översätts av en kompilator till kod som kan exekveras på en dator.

Det finns ett stort antal olika programmeringsspråk som man kan använda. Ett par vanliga är:

- C++
- C#
- Java

Java lite historia

- Java är ett programspråk som utvecklats av *Sun Microsystem* under ledning av *James Gosling*. Det blev allmänt tillgängligt 1995. Från början var tanken med Java att det skulle användas i olika typer av elektronik som brödrostar eller diskmaskiner.
- 1995: Från början var Java mest känt som ett slags programspråk som man använder på Internet för att skapa häftiga effekter på webbsidor. Javaprogrammet kan t.ex. generera ljud och rörliga bilder eller låta användaren kommunicera med programmet med hjälp av mus och tangentbord.
- Java är ett fullfjädrat programspråk. Med Java kan man, liksom t.ex. C++, skapa fullständiga applikationsprogram.

Egenskaper hos Java

- **Java är plattformsoberoende.**
Med *plattform* menas ett visst operativsystem som kör på en viss typ av dator. Windows XP på en PC är t. ex. en plattform. Linux på en PC är en annan plattform.
- **Java innehåller verktyg för att generera grafiska användargränssnitt.**
Man kan alltså med hjälp av Java skriva grafiska program, dvs sådana program som använder fönster, menyer, knappar etc, för att kommunicera med användaren.
- **Java är objektorienterat.**
Java bygger helt på de objektorienterade principen för att konstruera program.
- **Java gör det möjligt att skriva parallella program.**
Java stödjer nämligen s.k. multitrådar. Detta innebär att javaprogrammet kan beskriva flera aktiviteter som pågår samtidigt.
- **Java kan användas på Internet**
- **Java används vid programutveckling på Android-plattformen**

Java (IDE)

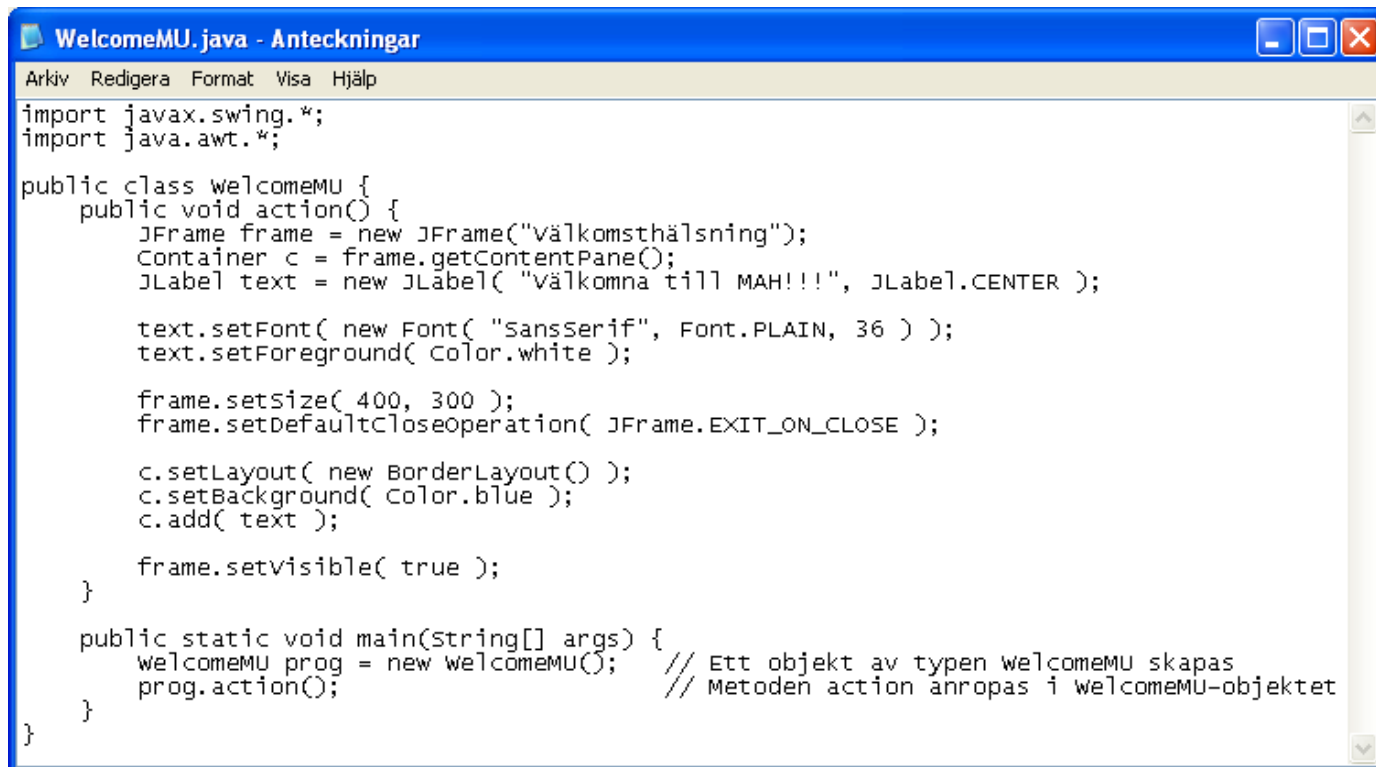
Ett modernt integrerat programutvecklingssystem, ett s.k. IDE (Integrated Development Environment) brukar erbjuda programmeraren en miljö där det finns alla hjälpmedel man behöver för att utveckla program. Det finns sådana system även för Java, t.ex.

- **JDK1.0, JDK1.0.2, JDK1.1, JDK1.2, JDK1.3, JDK1.4.2, JDK 1.5, JDK 6, JDK 7**
- **Eclipse Juno** (www.eclipse.org)
- **NetBeans 7.2** (www.netbeans.org)

På kursen kommer Eclipse att användas.

Hur programmerar man?

- 1) Man skriver in programkoden, **källkoden**, med hjälp av en **texteditor**.



```
Arkiv Redigera Format Visa Hjälp

import javax.swing.*;
import java.awt.*;

public class WelcomeMU {
    public void action() {
        JFrame frame = new JFrame("Välkomsthälsning");
        Container c = frame.getContentPane();
        JLabel text = new JLabel( "Välkomna till MAH!!!", JLabel.CENTER );

        text.setFont( new Font( "SansSerif", Font.PLAIN, 36 ) );
        text.setForeground( Color.white );

        frame.setSize( 400, 300 );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        c.setLayout( new BorderLayout() );
        c.setBackground( Color.blue );
        c.add( text );

        frame.setVisible( true );
    }

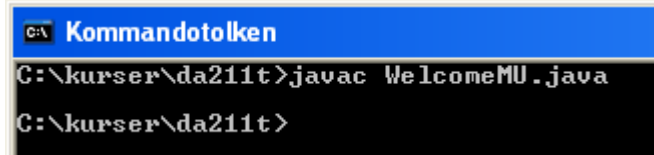
    public static void main(String[] args) {
        WelcomeMU prog = new WelcomeMU(); // Ett objekt av typen WelcomeMU skapas
        prog.action();                     // Metoden action anropas i WelcomeMU-objektet
    }
}
```

Man sparar källkoden (som enbart består av text) i en fil. När man arbetar med Java ska filen alltid ha suffixet java, t.ex.
WelcomeMU.java

Hur programmerar man?

- 2) Man kompilerar (översätter) källkodsfilen till ett format som kan exekveras på en dator. När du arbetar med Java heter kompilatorn **javac.exe**.

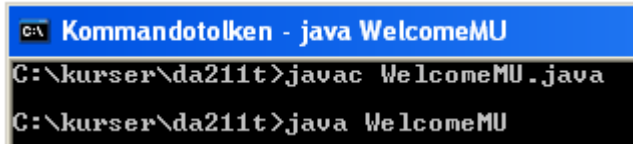
Resultatet av översättningen är ett format som kallas **bytekod** (eller javabytekod). Bytekoden sparas i en filen som ges suffixet class (t.ex. WelcomeMU.class).



```
C:\ Kommandotolken
C:\kursen\da211t>javac WelcomeMU.java
C:\kursen\da211t>
```

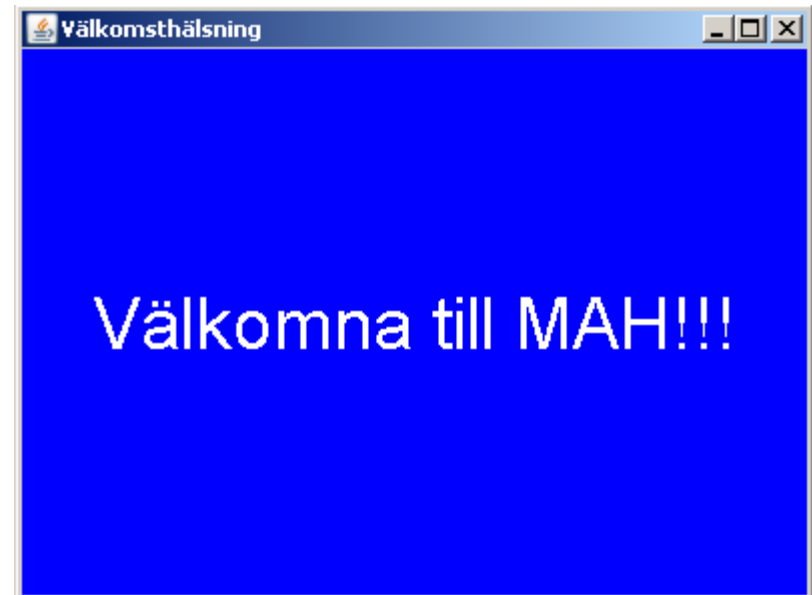
Hur programmerar man?

- 3) För att exekvera bytekodsfilen, eller bytekodsfilerna, krävs ett speciellt program som kallas **JVM** (**J**ava **V**irtual **M**achine). JVM tolkar bytekodsfilerna och utför instruktionerna som de innehåller. JVM ingår i JDK och heter **java.exe**.

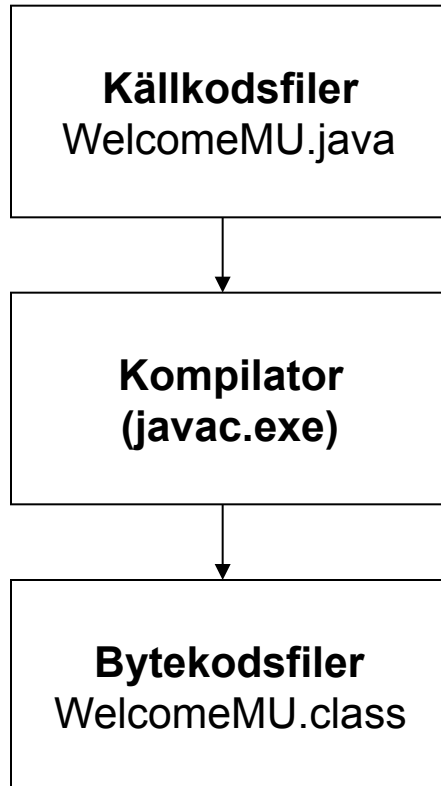


```
C:\> Kommandotolken - java WelcomeMU
C:\kurser\da211t>javac WelcomeMU.java
C:\kurser\da211t>java WelcomeMU
```

JVM finns för olika plattformar. Därför kan samma bytekodsfiler exekveras på olika plattformar.



Ett enkelt program – kompilera källkoden

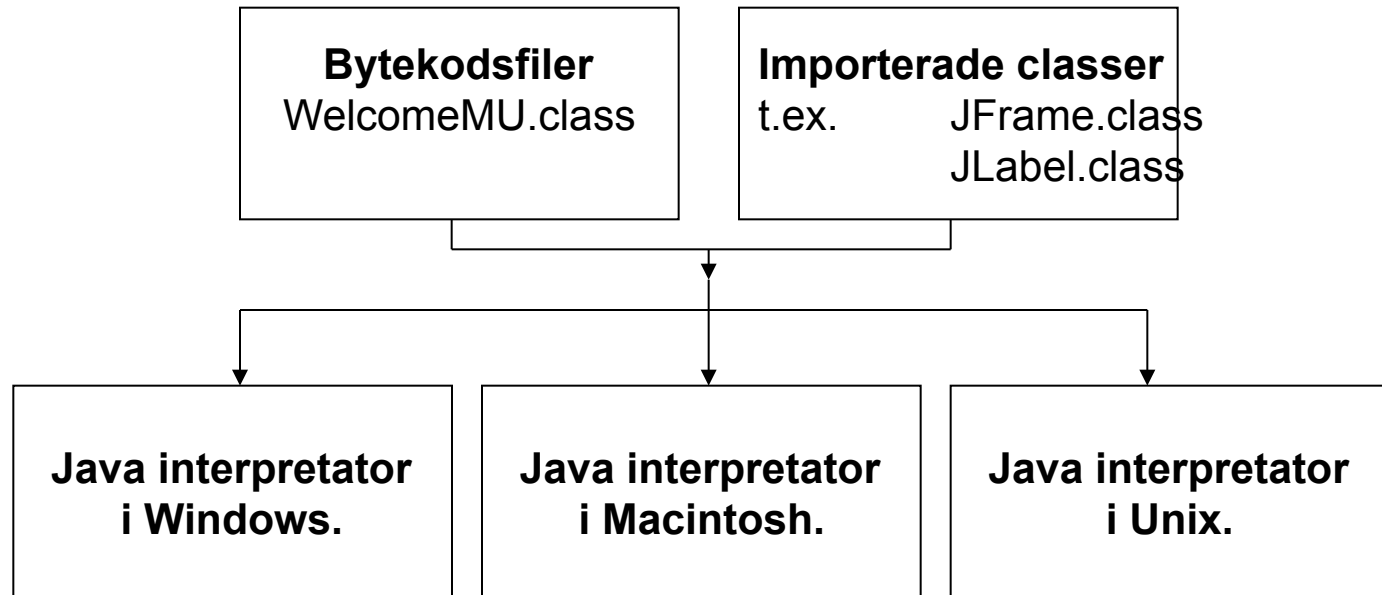


Källkodsfilerna läses av
programmet javac.exe

javac.exe översätter källkodsfilerna
till bytekodsfiler

Bytekodsfilerna används när
programmet ska exekveras (köras)

Ett enkelt program – exekvera bytekoden



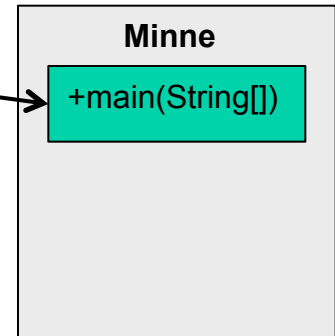
Bytekodsfilerna och filer som importeras tolkas av Java-interpretatorn (JVM – Java Virtual Machine, t.ex. java.exe) och ger ett körresultat.

Ungefär samma körresultat uppnås på olika plattformar.

Exekvering av programmet WelcomeMU

1. Exekveringen börjar med att *main*-metoden läses in (skapas) i datorns minne.

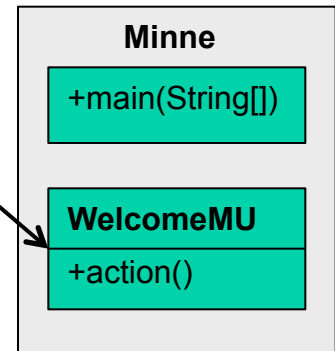
```
public static void main(String[] args)
    welcomeMU prog = new welcomeMU();
    prog.action();
}
```



2. Den första (översta) raden i *main*-metoden exekveras:

```
WelcomeMU prog = new WelcomeMU();
```

Raden innebär att ett objekt av typen *WelcomeMU* skapas i minnet.



3. Metoden *action* anropas i det nyss skapade objektet

```
prog.action();
```

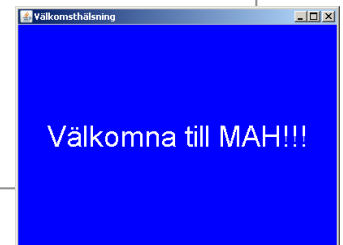
varvid exekveringen fortsätter i *action*-metoden

```
public void action() {
    JFrame frame = new JFrame("Välkomsthälsning");
    Container c = frame.getContentPane();
    JLabel text = new JLabel("Välkomna till MAH!!!", JLabel.CENTER);
    text.setFont(new Font("SansSerif", Font.PLAIN, 36));
    text.setForeground(Color.white);
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    c.setLayout(new BorderLayout());
    c.setBackground(Color.blue);
    c.add(text);
    frame.setVisible(true);
}
```

Exekvering av programmet WelcomeMU

4. Instruktionerna i metoden *action* exekveras en i tagen, uppifrån och ner

```
public void action() {  
    JFrame frame = new JFrame("Välkomsthälsning");  
    Container c = frame.getContentPane();  
    JLabel text = new JLabel( "Välkomna till MAH!!!", JLabel.CENTER );  
  
    text.setFont( new Font( "SansSerif", Font.PLAIN, 36 ) );  
    text.setForeground( Color.white );  
  
    frame.setSize( 400, 300 );  
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );  
  
    c.setLayout( new BorderLayout() );  
    c.setBackground( Color.blue );  
    c.add( text );  
  
    frame.setVisible( true );  
}
```



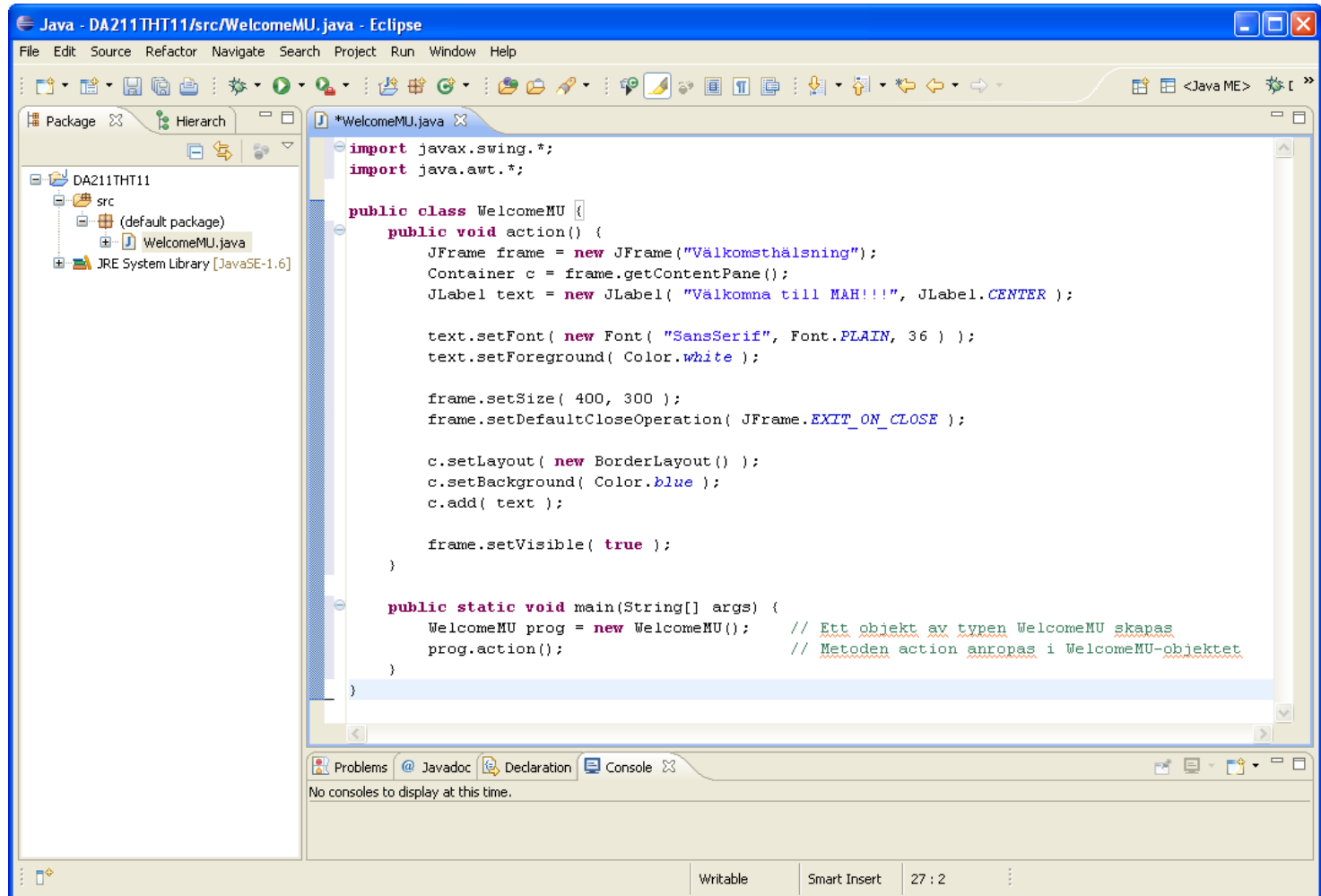
5. När den sista instruktionen i metoden *action* utförts

```
frame.setVisible( true );
```

fortsätter exekveringen i *main*-metoden. *main*-metoden innehåller inga fler instruktioner och därmed avslutas programmet.

Eclipse

Eclipse är en integrerad utvecklingsmiljö som innehåller alla verktyg du behöver för att utveckla ett java-program.



Ett Java-program

Skriv ett program som ger körresultatet:

```
Mitt namn är Nina.
```

```
Jag är 16 år gammal och bor i Nässjö.
```

Aktuella instruktioner: `System.out.println()`

Använd Eclipse!

Ett Java-program

Börja med att skriva en klass. Tänk på att ge klassen ett bra namn och att klassens namn ska börja med stor bokstav.

```
public class Nina {  
}
```

Klassens huvud

public class Klassnamn

Klassen till vänster heter Nina

Blockparenteserna, { och },
markerar start och slut på **klassens kropp**.

JF skriver parenteserna så här:

```
public class Nina  
{  
  
}
```

Ett Java-program

Skriv de metoder som behövs i klassen. I det här fallet räcker det med en metod. Metoden ska ha ett lämpligt namn, t.ex. info.

```
public class Nina {  
    public void info() {  
  
    }  
}
```

Raden med metodens namn kallas för *metodens huvud*.

Blockparenteserna markerar start och slut på info-metoden. Instruktioner i blocket utgör *metodens kropp*. På nästa sida lägger vi till några instruktioner.

Ett Java-program

Lägg till instruktioner som ska utföras då metoden info anropas.

Instruktionerna i metoden info beskriver vad som ska uträttas då metoden anropas.

INSTRUKTION
Utskrift av tom rad
System.out.println()

```
public class Nina {  
    public void info() {  
        System.out.println( "Mitt namn är Nina." );  
        System.out.println();  
        System.out.println( "Jag är 16 år gammal ..." );  
    }  
}
```

Metoden innehåller tre satser med kod (tre statement).
Varje sats avslutas med semikolon ;.

Ett Java-program

Lägg till en main-metod i en klass. Exekveringen börjar alltid i en main-metod.

Vi lägger till en main-metod i klassen Nina. I main-metoden ska vi skapa det objekt som behövs. Sedan anropar vi metoden info.

Metoden main beskriver vad som ska uträttas av programmet.

```
public class Nina {  
    public void info() {  
        System.out.println( "Mitt namn är Nina." );  
        System.out.println();  
        System.out.println( "Jag är 16 år gammal ..." );  
    }  
  
    public static void main( String[] args ) {  
        Nina presentation = new Nina();  
        presentation.info();  
    }  
}
```

// skapar ett Nina-objekt
// anropar info-metoden

Blockparenteserna markerar start och slut på main-metoden.

Ett Java-program

```
public class Nina {  
    public void info() {  
        System.out.println( "Mitt namn är Nina." );  
        System.out.println();  
        System.out.println( "Jag är 16 år gammal ..." );  
    }  
  
    public static void main( String[] args ) {  
        Nina presentation = new Nina();  
        presentation.info();  
    }  
}
```

Diagram illustrating line numbers for the code above:

- Line 3: `System.out.println("Mitt namn är Nina.");`
- Line 4: `System.out.println();`
- Line 5: `System.out.println("Jag är 16 år gammal ...");`
- Line 1: `Nina presentation = new Nina();`
- Line 2: `presentation.info();`

Satserna med kod utförs [uppfifrån och ned](#).

Nina.java

Vad händer om vi skriver raderna i info i annan ordning?

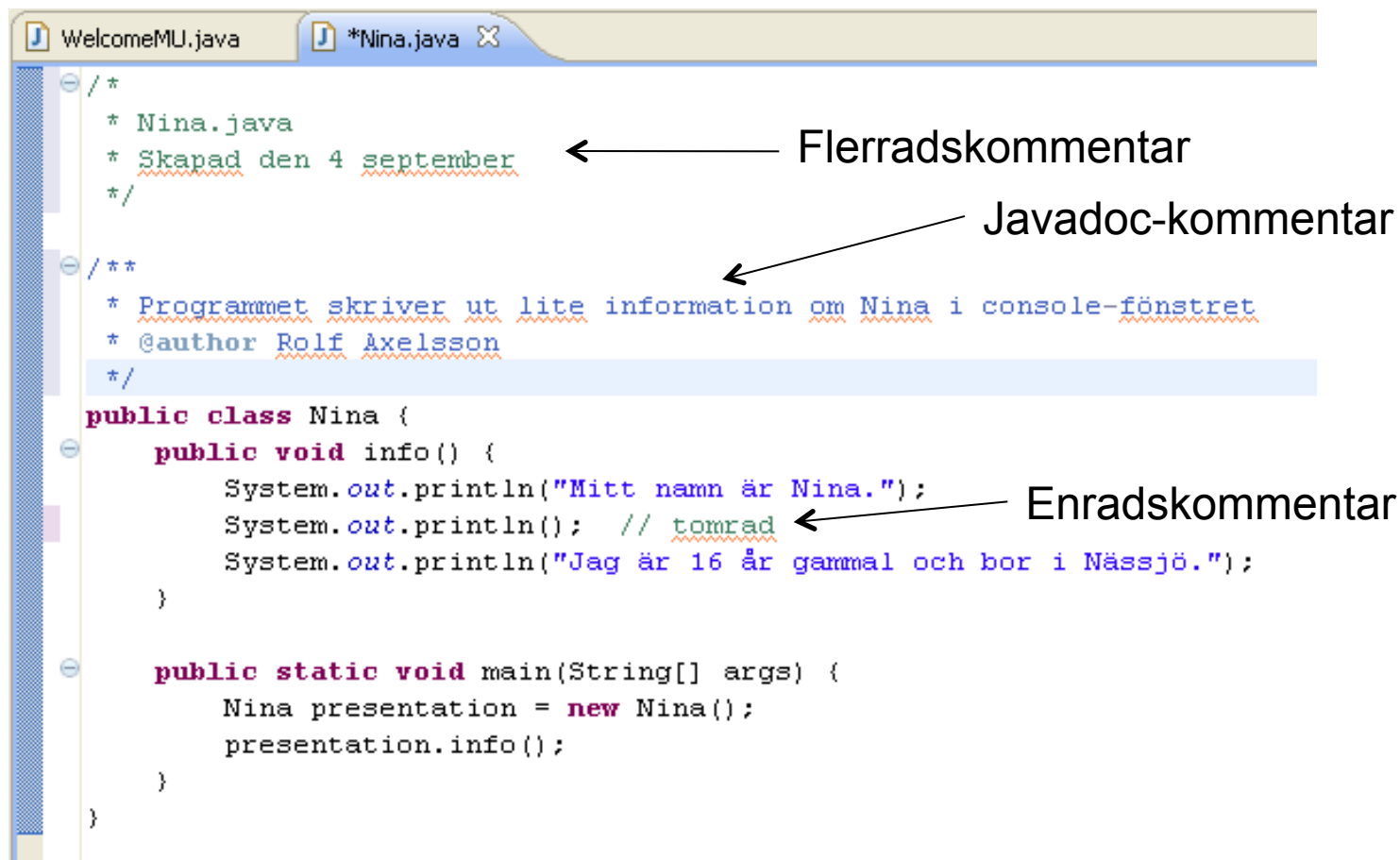
Vad händer om main-metoden avslutas med två identiska instruktioner?

```
presentation.info();  
presentation.info();
```

Kommentarer

Ett program ska innehålla kommentarer som beskriver vad det gör.

Kompilatorn bryr sig inte om kommentarerna. De påverkar inte exekveringen.



```
WelcomeMU.java *Nina.java X
/*
 * Nina.java
 * Skapad den 4 september
 */
/**
 * Programmet skriver ut lite information om Nina i console-fönstret
 * @author Rolf Axelsson
 */
public class Nina {
    public void info() {
        System.out.println("Mitt namn är Nina.");
        System.out.println(); // tomrad
        System.out.println("Jag är 16 år gammal och bor i Nässjö.");
    }

    public static void main(String[] args) {
        Nina presentation = new Nina();
        presentation.info();
    }
}
```

Flerradskommentar

Javadoc-kommentar

Enradskommentar

Identifiers - identifierare

- *Identifiers* are the words a programmer uses in a program
 - can be made up of letters, digits, the underscore character (`_`) and `$`
 - cannot begin with a digit
- Java is *case sensitive* - `Total`, `total`, and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers, such as
 - *title case* for class names - `Lincoln`
 - *upper case* for constants – `MAXIMUM`

Identifiers – identifiers are

- Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as `println`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language.
A reserved word cannot be used in any other way

Reserved words, reserved word

- The Java reserved words

<code>abstract</code>	<code>else</code>	<code>interface</code>	<code>switch</code>
<code>assert</code>	<code>enum</code>	<code>long</code>	<code>synchronized</code>
<code>boolean</code>	<code>extends</code>	<code>native</code>	<code>this</code>
<code>break</code>	<code>false</code>	<code>new</code>	<code>throw</code>
<code>byte</code>	<code>final</code>	<code>null</code>	<code>throws</code>
<code>case</code>	<code>finally</code>	<code>package</code>	<code>transient</code>
<code>catch</code>	<code>float</code>	<code>private</code>	<code>true</code>
<code>char</code>	<code>for</code>	<code>protected</code>	<code>try</code>
<code>class</code>	<code>goto</code>	<code>public</code>	<code>void</code>
<code>const</code>	<code>if</code>	<code>return</code>	<code>volatile</code>
<code>continue</code>	<code>implements</code>	<code>short</code>	<code>while</code>
<code>default</code>	<code>import</code>	<code>static</code>	
<code>do</code>	<code>instanceof</code>	<code>strictfp</code>	
<code>double</code>	<code>int</code>	<code>super</code>	

White space

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation

```
//*****  
//  Lincoln1.java          Java Foundations  
//  
//  Demonstrates a well formatted program.  
//*****  
  
public class Lincoln1 {  
    public static void main(String[]args){  
        System.out.println("A quote by Abraham Lincoln:");  
        System.out.println("Whatever you are, be a good one.");  
    }  
}
```

Lincoln2.java

```
/**
 * Lincoln2.java      Java Foundations
 *
 * Demonstrates a poorly formatted, though valid, program.
 */
```

```
public class Lincoln2{public static void main(String[]args){
    System.out.println("A quote by Abraham Lincoln:");
    System.out.println("Whatever you are, be a good one.");}}
```

```
/**
 * Lincoln3.java      Java Foundations
 *
 * Demonstrates another valid program that is poorly formatted.
 */
```

```

    public      class
Lincoln3      {      public      static
    void main      (
String
        []      args      )      {      System.out.println
"A quote by Abraham Lincoln:"      )
    ;      System.out.println
        (      "Whatever you are, be a good one."
    )      ;      }
```

Errors - fel

- A program can have three types of errors
 - The compiler will find syntax errors and other basic problems (*compile-time errors* - *kompileringsfel*)
 - If compile-time errors exist, an executable version of the program is not created
 - A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors* - *exekveringsfel*)
 - A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors* – *logiskt fel*)