

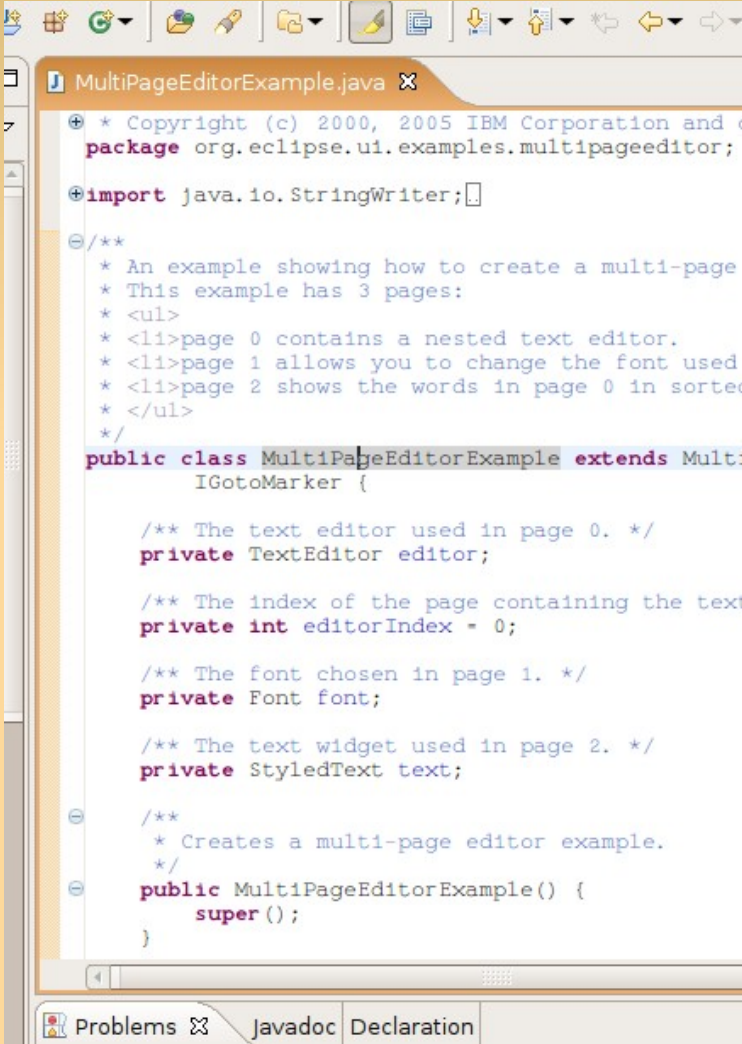
# Programming in Java: lecture 2

- Program Structure
- Variables
- Types
- The String Class
- Expressions
- TextIO

Slides made for use with "Introduction to Programming Using Java, Version 5.0" by David J. Eck  
Some figures are taken from "Introduction to Programming Using Java, Version 5.0" by David J. Eck  
Lecture 2 covers Section 2.1 to 2.5

# Program Structure

- Source Code – Kilde kode
- Syntax – Syntaks
  - Define what is a “correct” program
  - Syntax coloring
  - <notation>
- Semantics – Betydning
  - Does the program do what it is meant to?



```
MultiPageEditorExample.java x
+ * Copyright (c) 2000, 2005 IBM Corporation and o
package org.eclipse.ui.examples.multipageeditor;

+import java.io.StringWriter;

-/**
 * An example showing how to create a multi-page
 * This example has 3 pages:
 * <ul>
 * <li>page 0 contains a nested text editor.
 * <li>page 1 allows you to change the font used
 * <li>page 2 shows the words in page 0 in sorted
 * </ul>
 */
public class MultiPageEditorExample extends Multi
    IGoToMarker {

    /** The text editor used in page 0. */
    private TextEditor editor;

    /** The index of the page containing the text
    private int editorIndex = 0;

    /** The font chosen in page 1. */
    private Font font;

    /** The text widget used in page 2. */
    private StyledText text;

    /**
     * Creates a multi-page editor example.
     */
    public MultiPageEditorExample() {
        super();
    }
}
```

# Pragmatics

- Traditions
- Conventions
- camelCase
  - Classes: capital first letter
  - variables: lower case first letter
- JavaDoc
  - autogenerated documentation

```
* <li>page 2 shows the words in page 0 in sorted order.
* </li>
* </ul>
*/
public class MultiPageEditorExample extends MultiPageEditorPart implements
    IGotoMarker {

    /** The text editor used in page 0. */
    private TextEditor editor;
```

```
/** The index of the page containing the current editor. */
private int editorIndex = 0;

/** The font chosen in page 1. */
```

```
/**
 * An example showing how to create a multi-page editor.
 * This example has 3 pages:
 * <ul>
 * <li>page 0 contains a nested text editor.
 * <li>page 1 allows you to change the font used in page 2
 * <li>page 2 shows the words in page 0 in sorted order
 * </li>
 * </ul>
 */
public class MultiPageEditorExample extends MultiPageEditorPart implements
    IGotoMarker {

    /** The text editor used in page 0. */
    private TextEditor editor;
```

# Comments and JavaDoc

```
// This is a single line comment
```

```
/*
```

```
 * This is a multi line comment
```

```
*/
```

```
/**  
 * An example showing how to create a multi-page editor.  
 * This example has 3 pages:  
 * <ul>  
 * <li>page 0 contains a nested text editor.  
 * <li>page 1 allows you to change the font used in page 2  
 * <li>page 2 shows the words in page 0 in sorted order  
 * </ul>  
 */  
public class MultiPageEditorExample extends MultiPageEditorPart implements  
    IGotoMarker {  
  
    /** The text editor used in page 0. */  
    private TextEditor editor;
```

# Identifiers

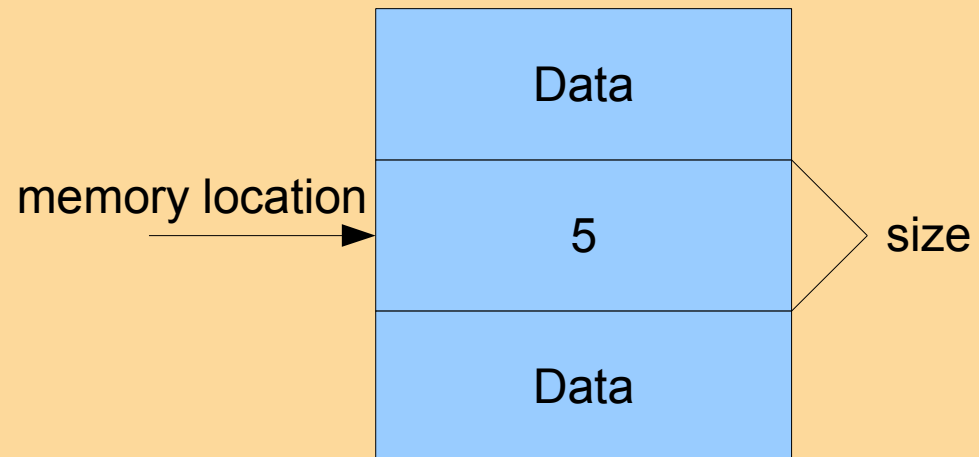
- Structure
  - Must start with letter or “\_”
  - Can contain numbers
  - Examples: `_local`, `x2`, `variableName`
- Simple identifiers – local
  - Contains no “.”
- Compound identifiers – “global”
  - `System.out.println`

# Reserved Words

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

# Variables

- A box that contains data
  - A location in memory
- The data inside the box
  - A value
- Example:
  - $x = x + 2$



# Types

- Java is strongly typed
  - Weakly typed: Hope for the best
- Apples and oranges
  - Automatic conversion or Compile error
- Types:
  - Primitive Types:
    - boolean, int, short, ...
  - Classes:
    - String, ...



# Primitive Types

- Not classes
- Describes a single value
- Integer:
  - `int x = 5`
  - `int y = 345`
- Double
  - `double y = 3.56` (Bemærk komma er .)

# Primitive Types 2

| Name  | bits | Range   |
|-------|------|---|
| byte  | 8    | -128 to 127   |
| short | 16   | -32,768 to 32,767   |
| int   | 32   | -2,147,483,648<br>to<br>2,147,483,647                         |
| long  | 64   | -9,223,372,036,854,775,808<br>to<br>9,223,372,036,854,775,807 |

# Primitive Types 3

- float      32-bit
  - 7 significant digits
- double    64-bit
  - 15 significant digits
- boolean   1 bit of information
  - true (1) or false (0)

# Variable Declarations

- Reserves space in memory
- Makes the name (identifier) usable after this point.
- `<type-name> <variable-name-or-names>`
  - `int numberOfStudents;`
  - `String name; // First, middle and last name`
  - `double x, y; // represents coordinates`
  - `boolean isFinished;`
  - `char firstInitial, middleInitial, lastInitial;`



Space for Data

# Assignment Statements

- Putting something into the box
- `<variable> = <expression>`
- `x = 2`
- `y = 4 * 5`
- `myVariable = x / y`
- More expression later

```
⊖/**
 * This class implements a simple program that
 * will compute the amount of interest that is
 * earned on $17,000 invested at an interest
 * rate of 0.07 for one year. The interest and
 * the value of the investment after one year are
 * printed to standard output.
 */

public class Interest {

    ⊖ public static void main(String[] args) {

        /* Declare the variables. */

        double principal;    // The value of the investment.
        double rate;         // The annual interest rate.
        double interest;     // Interest earned in one year.

        /* Do the computations. */

        principal = 17000;
        rate = 0.07;
        interest = principal * rate;    // Compute the interest.

        principal = principal + interest;
            // Compute value of investment after one year, with interest.
            // (Note: The new value replaces the old value of principal.)

        /* Output the results. */

        System.out.print("The interest earned is $");
        System.out.println(interest);
        System.out.print("The value of the investment after one year is $");
        System.out.println(principal);

    } // end of main()

} // end of class Interest
```

# Literals

- Numbers
  - int: 1200, -30
  - long: 1244L, -30L
- Floating point
  - double: 55.4
    - $12e5 = 12 * 10^5$
  - float 12.3F, 24,953f

# Literals 2

- String literals
  - “This is a String”
- Char literals
  - 'a', '\n', '\t', '\\'
  - '\u00E9' = é
- Boolean
  - true
  - false



# Literals 3

- Hexadecimal – 0 ... 9 A ... F
  - 0x45 or 0xFF7A
  - $4 * 16 + 5$
- Octal – 0 ... 7
  - $045 = 37$
  - $4 * 8 + 5$

# Two Purposes of Classes

- Static collections of functions
  - Example: Math
    - Collection of static members
    - Math.PI
    - Math.random() // random double between 0 and 1
- Template for Objects
  - Example: String
    - String.equals()

# Subroutine call statement

- `<method-name>(<parameters>)`
- Examples
  - `System.out.println("This is a String")`
  - `Math.rand() // no parameters`

# String

- `String firstName = "Ulrik";`
- `firstName.equals("Nyman");`
- `false`

# String 2

- `s1.equals(s2)`
- `s1.equalsIgnoreCase(s2)`
- `s1.length()`
- `s1.charAt(N)`
- `s1.substring(N,M)`
- `s1.toUpperCase()`

# Concatenation

- String + anything = String
- Examples:
  - `int numberOfDays = 7;`
  - `"The week has " + numberOfDays + " days"`
  - 
  - `numberOfDays * 2 + " this is 14"`

# Math

- `Math.rand()`
- `Math.PI` // constant
- `Math.sqrt(x)` Square root
- `Math.sin(y)`
- `Math.floor(double d)` // returns integer

# Enums

- Special type of classes
  - `enum <enum-type-name> { <list-of-enum-values> }`
  - `enum Season { SPRING, SUMMER, FALL, WINTER }`
  - `Season.WINTER`
  - `Season vacation;`
  - `vacation = Season.SUMMER;`
  - `Season.FALL.ordinal()` is 2,
  - `System.out.println(vacation);`



# Expressions

- plus +: result =  $4.0 + 3$
- multiplication \*:  $x = 3 * 4$
- division /:  $z = 5/6$  Gives an integer
  - $5.0/6$  Gives a double
- modulus %:  $34577 \% 100 = 77$
- minus -:  $t = 5 - 2$
- unary minus -:  $-4$

# Increment and decrement

- `counter = counter + 1;`
- `goalsScored = goalsScored + 1;`
- `counter = 4`
- `x = counter++;` // x = 4 old value
- `x = ++counter;` // x = 5 new value
- `goalsScored--;`

# Relational Operators

- $A == B$       Is A "equal to" B?
- $A != B$       Is A "not equal to" B?
- $A < B$         Is A "less than" B?
- $A > B$         Is A "greater than" B?
- $A <= B$       Is A "less than or equal to" B?
- $A >= B$       Is A "greater than or equal to" B?
- `boolean sameSign;`
- `sameSign = ((x > 0) == (y > 0));`

# Boolean Operators

- Comparison
  - ==
  - !=
- And &&
  - true && false
- Or ||
  - true || false
- Not !
  - true == !false

# Conditional Operator

- We save this for later

# Assignment Operators

- $x -= y;$       // same as:  $x = x - y;$
- $x *= y;$       // same as:  $x = x * y;$
- $x /= y;$       // same as:  $x = x / y;$
- $x \% = y;$       // same as:  $x = x \% y;$ 
  - (for integers  $x$  and  $y$ )
- $q \&\& = p;$       // same as:  $q = q \&\& p;$ 
  - (for booleans  $q$  and  $p$ )

# Type Casts

- `int A;`
- `double X;`
- `short B;`
- `A = 17;`
- `X = A; // OK; A is converted to a double`
- `B = A; // illegal; no automatic conversion`
- `// from int to short`

# Type Casts 2

- `int A;`
- `short B;`
- `A = 17;`
- `B = (short)A; // OK; A is explicitly type cast`
- `// to a value of type short`



# Precedence Rules

- Unary operators: ++, --, !, unary - and +, type-cast
- Multiplication and division: \*, /, %
- Addition and subtraction: +, -
- Relational operators: <, >, <=, >=
- Equality and inequality: ==, !=
- Boolean and: &&
- Boolean or: ||
- Conditional operator: ?:
- Assignment operators: =, +=, -=, \*=, /=, %=

# Type Conversion of Strings

- `Integer.parseInt("123")`
- `Double.parseDouble("3.14")`
- `Double.parseDouble("12.3e-7")`
- Same as literals
- Enum
  - `Season.valueOf("SUMMER")`

# TextIO

- Class file from textbook with modifications
- Hides details of getting Input
- `System.out.println("String")`
- `TextIO.put("String")`

# TextIO – printf – putf

- `System.out.printf("The product of %d and %d is %d", x, y, x*y);`
- Variable number of arguments
- `%d` – integer (decimal) number
  - `%12d`, minimum 12 characters
- `%s` – String (converted into string)
  - `%10s`, minimum 10 characters

# TextIO – printf – putf 2

- %f – floating point
  - %12.3f – 12 characters, 3 digits after decimal point
- %e – exponential
  - %15.8e – 8 digits after the decimal point
- %g – floating point or exponential
  - %12.4g – a total of 4 digits in the answer
- “ 5.345”
- “ 34.453”
- “ 123.875”

# TextIO 2

- `j = TextIO.getInt();` // Reads a value of type int.
- `y = TextIO.getDouble();` // Reads a value of type double.
- `a = TextIO.getBoolean();` // Reads a value of type boolean.
- `c = TextIO.getChar();` // Reads a value of type char.
- `w = TextIO.getWord();` // Reads one "word" as a value of type String.
- `s = TextIO.getln();` // Reads an entire input line as a String.

# TextIO – File I/O

- `TextIO.writeFile("result.txt")`
- `TextIO.writeUserSelectedFile()`
- `TextIO.writeStandardOutput()`
- `TextIO.readFile("data.txt")`
- `TextIO.readUserSelectedFile()`
- `TextIO.readStandardInput()`

# Example

```
⊖/**
 * This class implements a simple program that will compute
 * the amount of interest that is earned on an investment over
 * a period of one year.  The initial amount of the investment
 * and the interest rate are input by the user.  The value of
 * the investment at the end of the year is output.  The
 * rate must be input as a decimal, not a percentage (for
 * example, 0.05 rather than 5).
 */

public class Interest2 {

⊖    public static void main(String[] args) {

        double principal; // The value of the investment.
        double rate;      // The annual interest rate.
        double interest;  // The interest earned during the year.

        TextIO.put("Enter the initial investment: ");
        principal = TextIO.getlnDouble();

        TextIO.put("Enter the annual interest rate (decimal, not percentage!): ");
        rate = TextIO.getlnDouble();

        interest = principal * rate; // Compute this year's interest.
        principal = principal + interest; // Add it to principal.

        TextIO.put("The value of the investment after one year is $");
        TextIO.putln(new Double(principal));

    } // end of main()

} // end of class Interest2
```



# Recap

- Program Structure
- Variables
- Types
- The String Class
- Expressions
- TextIO