Variables and Calculations (JFS Chap 4)

- Handling data in Java programs:
  - Data *types*
  - *Variables*
  - *Assignment statements*
  - Calculating new data
  - Input dialogues and reading in data

Programs and *Data*

- All interesting/useful programs handle *data*
- **Questions:**
  - What *kinds of data* can a Java program handle?
  - How does a program *hold/refer to* data internally?
  - How can a program *manipulate/calculate* with data?
  - How can a program *display* ("output") data resulting from manipulation/calculation?
  - How can a program *obtain* ("input" or "read") data from the user to use in manipulations/calculations?

What kinds of data? (JFS Chap 4, p35)

- Programming language jargon for "kind of data" is **type**
- Textual data ("strings" of characters) (JFS p46):
  - Type name: `String`
  - Values enclosed in double quotes: `"Hello World!"`
  - Strings can be joined together using `+`
    Example: `"Hello " + "World!"`
- Whole numbers:
  - Type name: `int` (for "integer")
  - Possible values (for example): `3` `42` `-27` `0`
  - Range allowed: approximately
    -2 000 000 000 → +2 000 000 000
  - `int`s are stored *precisely* but the range is "limited"(?!)
  - (There is a `BigInteger` library if required)

Types of data (continued)

- Decimal point numbers ("real" or "floating point" numbers)
  - Type name: `float`
  - Possible values (for example): `12.34f` `-32.0f`
  - Range allowed: approximately
    -3.4*10^38 → +3.4*10^38
  - Huge range, but only 7 digits of accuracy are held
- There is also:
  - Type name: `double` ("double precision")
  - Possible values (for example): `12.34` `-32.0`
    (note: no letter!)
  - Range allowed: approximately
    -1.8*10^308 → +1.8*10^308
  - 15 digits of accuracy
- In practice `double` is simpler to use than `float`

Storing/referring to data

- A program holds data in *variables*
- A variable
  - Is a "slot" or "location" in the computer's main memory (RAM) – a small electronic storage box
  - Has a *name*, which we use to refer to the "slot"
  - Can hold data of a specified *type* (which we must choose and *declare* in advance)
  - Contains a *single data value* at any one time
  - Can have its value *changed*
- Example: Counting cars - we need an `int` variable, say called `count`:

Naming variables

- During program design we decide what items of data need to be held – one variable for each
- We must choose *names* ("identifiers") for the variables required
- Java has certain rules about identifiers:
  - Different variables must have different names!
  - The name of a variable must start with a letter, `_` or `$`, and can contain letters, digits, `_`s or `$`s
  - The language is *case-sensitive*
  - Certain words are "reserved", and may not be used as variable names (e.g. `class`, `public`, `import`, …)

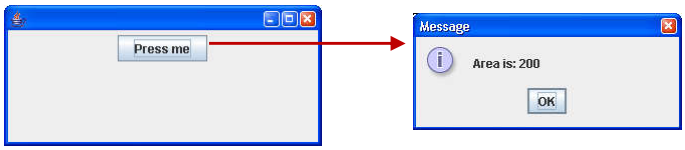
Naming variables (continued)

- However, there are some additional *conventions*:
  - Start a name of a variable with a *lower case letter*
  - Start subsequent word segments with an upper case letter
    e.g. `numberOfCars` `totalWeight`
  - Don't use underscores _ or $
  - Make names meaningful to the reader!

Example: A calculation application

- An application to calculate the area of a rectangle 20 units long and 10 units wide and display the result on the screen in a pop-up dialogue box (when a button is pressed)
- The appearance and behaviour:

Demo
AreaCalculation



- Next we think about the *data* to be handled, and the *variables* needed to store it

- Deciding about the variables we need:
  - `length` and `width` (of type `int`, we decide)
  - `area` (also of type `int`)
- The computation will proceed in steps like this:

	length	width	area
Memory locations:			
Store 20 in `length`	20		
Store 10 in `width`	20	10	
Calculate `length*width` and store in `area`	20	10	200

AreaCalculation application

(just the actionPerformed method)

Sequence

```
1. // This method calculates and displays the area
2. // of a rectangle when the button is clicked
3. public void actionPerformed(ActionEvent event) {
4.     int area;        // Declare the variables
5.     int length;
6.     int width;
7.     length = 20;     // Place data in variables
8.     width = 10;
9.     area = length * width; // Perform calculation
10.    JOptionPane.showMessageDialog(null,
                                 "Area is: " + area);
11. }
```

Note: `actionPerformed` is not obliged to do drawing!

Analysing the code: Key points

- Lines 1, 2, 4, 7 and 9 contain "**comments**":
  - *Everything* after `//` on a line is *ignored* by the compiler
  - So: for the human reader
  - Also everything following `/*` up to the next `*/`, whichever line it's on
- Lines 7-9 are the core of the program:
  - A sequence of *assignment statements*, e.g.

    ```
    height = 20;
    ```
  - More about assignment statements on the next slide
  - Note that each is followed by a semicolon `;`
  - They implement computation steps 2-4 (see slide 9)

Assignment statements

- In general an assignment statement has the form:

  ```
  variable = expression;
  ```
  - The value of `expression` is calculated,
  - *then* this new value *replaces* the current contents of the named variable
- How to say it: the `=` should be read as "becomes", or "becomes equal to", or "takes the value", **BUT NEVER AS "equals"!**
- Note: Thinking "becomes": `n = n + 1;` makes sense:
  - `n` gets a *new* value, which is the *current value* plus one
  - Java has a shorthand for this: `n++;`
    Similarly `n--;`

Declarations of variables

- Lines 4-6 are *"declarations of variables"*

```
int area;
int length;
int width;
```

  - They formally *introduce* the variables that we require
  - They must appear *before* the variables are used
  - They specify the *name* and the *type* of data that the variable is to hold:
    the compiler can then check that we use them consistently
- Variant forms of declaration:
  Grouping: `int length, width, area;`
  Initializing: `int length = 20;`
  Combined: `int length = 20, width = 10, area;`

Output using JOptionPane.showMessageDialog

- In line 10 we have a call of `JOptionPane.showMessageDialog`:

```
JOptionPane.showMessageDialog(null,
                              "Area is: " + area);
```

  - This displays our result
  - Note: `"Area is " + area`
    This builds up a message by *concatenating* the literal string `"Area is "` (note the space after `is`) with the current value of the variable `area`
  - The integer value in `area` is automatically converted to readable text (a string)
  - This can be taken further, eg:
    `"Area is " + area + " square metres"`
    and so on

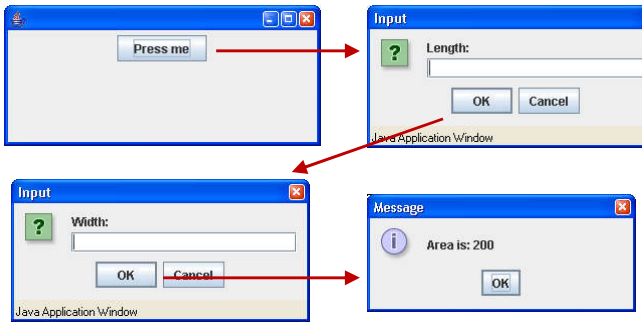
About arithmetic expressions and values

- "Expressions" are formulas built from:
  - Literal values: `27` `1.2f`
  - Variables: `length` `area`
  - Arithmetic operators: `+` `-` `*` `/` `%`
  - Round brackets: `( )`
  - Example: `(20 + a/3) % 5`
- Division with `int` values gives `int` result
- The `%` computes *remainders* (e.g. `27 % 4` has value `3`)
- See JFS for more details
  - Expressions & calculations follow standard maths conventions
  - Read JFS, Chapter 4, pages 42-46

Example: Another calculation application

- An application to calculate and display the area of a rectangle, but prompting the user to enter the length and width:

Demo
AreaDialogs



- For this we need *input dialogues* and *text-to-number conversion*

AreaDialogs application
(just the actionPerformed method)

```
public void actionPerformed(ActionEvent event) {
    int area;
    int length;
    int width;
    String lengthString;
    String widthString;
    lengthString =
            JOptionPane.showInputDialog("Length:");
    length = Integer.parseInt(lengthString);
    widthString =
            JOptionPane.showInputDialog("Width:");
    width = Integer.parseInt(widthString);
    area = length * width;
    JOptionPane.showMessageDialog(null,
                                  "Area is: " + area);
}
```


Key points

- We can declare variables to hold *text*:

  ```
  String lengthString;
  ```

- We use an *input dialogue box* to obtain some typed text from the user:

  ```
  JOptionPane.showInputDialog("Length:")
  ```

- That method call "returns a value" – the text typed, and we store it in a variable using an assignment statement:

  ```
  lengthString = JOptionPane. . . . ("Length:");
  ```

- That *really is just text*, so we need to *convert it* to a Java `int` number using another library method call:

  ```
  length = Integer.parseInt(lengthString);
  ```

- Now there is a true `int` in length to be used in calculations
- Phew!

End of Section