5. Defining Classes and Methods

Harald Gall, Prof. Dr.
Michael Würsch, Dr.

Institut für Informatik
Universität Zürich

http://seal.ifi.uzh.ch/info1

s.e.a.l.
software evolution & architecture lab

Learning Objectives

- Get familiar with the object-oriented terminology
- Learn how to define classes, attributes, and methods
- Learn how to obtain classes, attributes and methods from a natural language description
- Learn to use the class String and the Java API in general

Assembly Language – The old fashioned Way

```
.model tiny
.code
org 100h

main  proc

      mov    ah,9                        ; Display String Service
      mov    dx,offset hello_message     ; Offset of message (Segment DS is the right
                                         ; segment in .COM files)
      int    21h                         ; call DOS int 21h service to display message at
                                         ; ptr ds:dx

      retn                               ; returns to address 0000 off the stack
                                         ; which points to bytes which make int 20h (exit
                                         ; program)

hello_message db 'Hello, world!$'

main  endp
end   main
```

Assembly Language – The old fashioned Way

```
        .section        .rodata
string:
        .ascii "Hello, world!\n"
length:
        .quad . -string         #Dot = 'here'

        .section        .text
        .globl _start           #Make entry point visible to linker
_start:
        movq $4, %rax           #4=write
        movq $1, %rbx           #1=stdout
        movq $string, %rcx
        movq length, %rdx
        int $0x80               #Call Operating System
        movq %rax, %rbx         #Make program return syscall exit status
        movq $1, %rax           #1=exit
        int $0x80               #Call System Again
```

Object-Oriented Terminology

- Objects/Instances
  - are an abstraction of real-world things
  - have a state, behavior, and identity
  - are instances of a single class
- Classes
  - are blueprints for a family of objects
  - define attributes and methods
- Attributes/Instance Variables
  - their values at runtime represent the state or data of an object
- Methods
  - define the actions/behavior of an object of a class
  - access/change the state of an object
  - call other methods

Example: Automobile

- A class **`Automobile`** as a blueprint

Class Name: `Automobile`

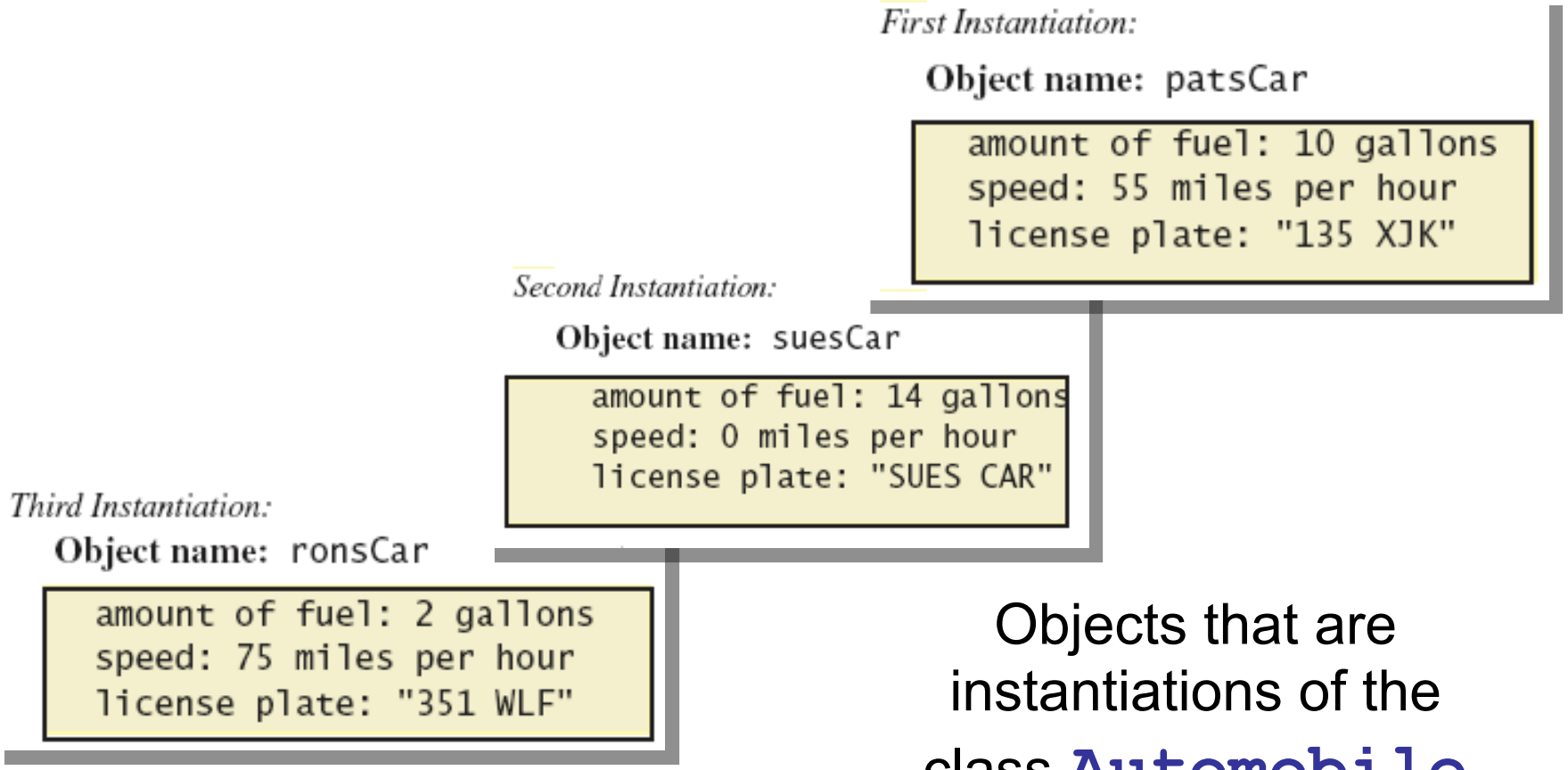
Data:

`amount of fuel_____`
`speed _____`
`license plate _____`

Methods (actions):

`accelerate:`
How: Press on gas pedal.
`decelerate:`
How: Press on brake pedal.

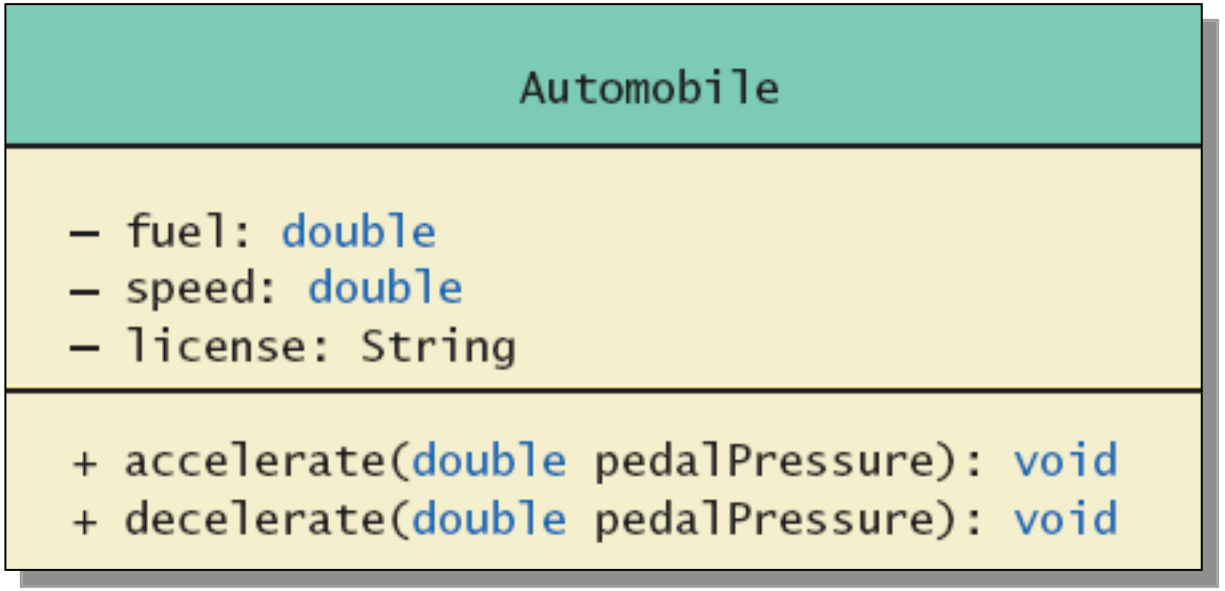
Class and Method Definitions



Objects that are instantiations of the class **`Automobile`**

Class and Method Definitions

- A class outline as a UML class diagram



Example: Automobile Code

Attributes

- Attributes or instance variables are variables defined in a class (outside of a method)
- Each object of the class has a separate copy
- They live in memory for the life of the object
- They can be accessed from anywhere in the class

Methods

- Signature:
  `<return type> <identifier>(<param list>) { }`
- Two kinds of Java methods
  - Return a single item, i.e. **`return type`**
  - No return type: a **`void`** method
- The method **`main`** is a **`void`** method
  - Invoked by the system
  - Not by the program

Primitive Types as Formal Method Parameters

- Parameters are a means of passing information from a caller of a method to the method itself
- A method can have zero or more parameters of different types
- Parameters are variables, they are local to a method
- Callers must provide the correct number of values/ types, automatic type conversion is carried out were appropriate

Object Analysis

From Problem Descriptions in Natural Language to Object-Oriented Designs

Look out for different parts of speech to obtain a first set of candidates for classes, attributes and methods:

- Nouns
  Candidates for classes and attributes
- Verbs
  Relations or behaviors (methods)
- Adjectives
  Define or restrict ranges of values

Exercise: University Information System

Domain Description:

“Students have a first and a last name. Each student can be uniquely identified by his or her student number. The year of their first semester enrollment is recorded. This information is then used to report every year how long students remain at the UZH in average.”

The Class String

- It is part of the Java class library, but it is not a primitive type.
- A value of type `String` is a sequence of characters treated as a single item.
- Strings are immutable

Declaring and Printing Strings

- declaring

```
String greeting;
greeting = "Hello!";
```

- or

```
String greeting = "Hello!";
```

- or

```
String greeting = new String("Hello!");
```

- printing

```
System.out.println(greeting);
```

Concatenation of Strings

- Two strings are *concatenated* using the + operator.

```
String greeting = "Hello";
String sentence;
sentence = greeting + " officer";
System.out.println(sentence);
```

- Any number of strings can be concatenated using the + operator.

Concatenating Strings and Integers

```
String solution;
solution = "The temperature is " +
72;
System.out.println (solution);
```

```
> The temperature is 72
```

String Methods

Method	Description	Example
`length()`	*Returns the length of the* `String` *object.*	`String greeting = "Hello!";` `greeting.length()` *returns* 6.
`equals(`*Other_String*`)`	*Returns* `true` *if the calling object string and the* *Other_String* *are equal. Otherwise, returns* `false`.	`String greeting =` `keyboard.next();` `if (greeting.equals("Hi"))` `System.out.println(` `"Informal Greeting.");`
`equalsIgnoreCase(` *Other_String*`)`	*Returns* `true` *if the calling object string and the* *Other_String* *are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns* `false`.	*If a program contains* `String s1 = "mary!";` *then after this assignment,* `s1.equalsIgnoreCase("Mary!")` *returns* `true`.
`toLowerCase()`	*Returns a string with the same characters as the calling object string, but with all characters converted to lowercase.*	`String greeting = "Hi Mary!";` `greeting.toLowerCase()` *returns* `"hi mary!"`
`toUpperCase()`	*Returns a string with the same characters as the calling object string, but with all characters converted to uppercase.*	`String greeting = "Hi Mary!";` `greeting.toUpperCase()` *returns* `"HI MARY!"`
`trim()`	*Returns a string with the same characters as the calling object string, but with leading and trailing whitespace removed.*	`String pause = " Hmm ";` `pause.trim()` *returns* `"Hmm"`
`charAt(`*Position*`)`	*Returns the character in the calling object string at* *Position*. *Positions are counted 0, 1, 2, etc.*	`String greeting = "Hello!";` `greeting.charAt(0)` *returns* `'H'`. `greeting.charAt(1)` *returns* `'e'`.
`substring(`*Start*`)`	*Returns the substring of the calling object string from position* *Start* *through to the end of the calling object. Positions are counted 0, 1, 2, etc.*	`String sample = "AbcdefG";` `sample.substring(2)` *returns* `"cdefG"`.

Method	Description	Example
`substring(`*Start*`,` *End*`)`	*Returns the substring of the calling object string from position* *Start* *through, but not including, position* *End* *of the calling object. Positions are counted 0, 1, 2, etc.*	`String sample = "AbcdefG";` `sample.substring(2, 5)` *returns* `"cde"`.
`indexOf(`*A_String*`)`	*Returns the position of the first occurrence of the string* *A_String* *in the calling object string. Positions are counted 0, 1, 2, etc. Returns* –1 *if* *A_String* *is not found.*	`String greeting = "Hi Mary!";` `greeting.indexOf("Mary")` *returns* 3. `greeting.indexOf("Sally")` *returns* –1
`indexOf(`*A_String*`,` *Start*`)`	*Returns the position of the first occurrence of the string* *A_String* *in the calling object string that occurs at or after position* *Start*. *Positions are counted 0, 1, 2, etc. Returns* –1 *if* *A_String* *is not found.*	`String name =` `"Mary, Mary quite contrary";` `name.indexOf("Mary", 1)` *returns* 6. *The same value is returned if* 1 *is replaced by any number up to and including* 6. `name.indexOf("Mary", 0)` *returns* 0. `name.indexOf("Mary", 8)` *returns* –1
`lastIndexOf(` *A_String*`)`	*Returns the position of the last occurrence of the string* *A_String* *in the calling object string. Positions are counted 0, 1, 2, etc. Returns* –1 *if* *A_String* *is not found.*	`String name =` `"Mary, Mary, Mary quite so";` `name.lastIndexOf("Mary")` *returns* 12.
`compareTo(`*A_String*`)` *Compares the calling object string with* *A_String* *to see which comes first in the lexicographic ordering. Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase or all lowercase. If the calling string is first,* `compareTo` *returns a negative value. If the two strings are equal, it returns zero. If the argument is first, it returns a positive number.*		`String entry = "adventure";` `entry.compareTo("zoo")` *returns a negative number.* `entry.compareTo("adventure")` *returns zero.* `entry.compareTo("above")` *returns a positive number.*

Display 2.7
Methods in the Class String

The Method length()

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```
int count = solution.length();
System.out.println(solution.length());
spaces = solution.length() + 3;
```

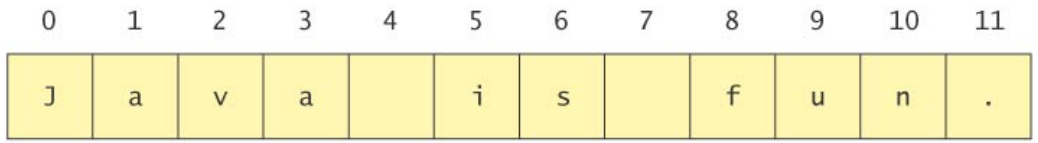
Positions in a String

- positions start with 0, not 1.
  - The `'J'` in `"Java is fun."` is in position 0

Positions in a `String`, cont.

- A position is referred to an an *index.*
  - The `'f'` in `"Java is fun."` is at index 9.

The twelve characters in the string `"Java is fun."` have indices 0 through 11. The index of each character is shown above it.



Note that the blanks and the period count as characters in the string.

Display 2.8

String Indices

(Not) Changing `String` Objects

- No methods allow you to change the value of a `String` object.
- But you can change the value of a String variable.

```
String pause = "   Hmm   ";
pause = pause.trim();
pause = pause + "mmm!";
pause = "Ahhh";
```

value of pause

Hmm

Hmm

Hmmmmm

Ahhh

Escape Characters

- How would you print

```
"Java" refers to a language?
```

- The compiler needs to be told that the quotation marks (") do not signal the start or end of a string, but instead are to be printed.

```
System.out.println(
"\"Java\" refers to a language.");
```

Escape Characters

`\"`	Double quote.
`\'`	Single quote.
`\\`	Backslash.
`\n`	New line. Go to the beginning of the next line.
`\r`	Carriage return. Go to the beginning of the current line.
`\t`	Tab. Add whitespace up to the next tab stop.

Display 2.10

Escape Characters

- Each escape sequence is a single character even though it is written with two symbols.

Examples

```
System.out.println("abc\\def");
abc\def
System.out.println("new\nline");
new
line
char singleQuote = '\'';
System.out.println(singleQuote);
'
```