Object Oriented Programming

PART IV

Strings, Coding Standards

Beniamin Bogosel

Aurel Vlaicu University, Arad

- ⋆ a string contains zero or more characters
- ⋆ `String` = non-mutable object
  - `new String("Salut");`
  - `"Salut"`
- ⋆ some methods:
  - `int length()`
  - i-th character: `char charAt(int i)`

Uniqueness of `String` literals

- ⋆ a literal of type `String` is always the same object
- ⋆ to test equality between two `String`: `x.equals(z)` NOT `x==z`
- ⋆ Concatenation using `+`
- ⋆ We can concatenate strings with almost everything: automatic conversion to string
- ⋆ Every class has a method `toString()` which converts the object to a string

```
X a = new X();
System.out.println(a.toString());
```

The class `StringBuffer`

⋆ Class of mutable strings: strings that can be changed

⋆ `x.append(...)`, `x.insert(...)`, `x.toString()`, many more available

```
StringBuffer a = new StringBuffer();
a.append("hello world");
System.out.println(a.toString());
```

The class `StringTokenizer`

- ⋆ `java.util.StringTokenizer`
- ⋆ can cut a string into multiple words
- ⋆ word+ separator+word+separator

`"Buna ziua tuturor !"`

Separator=" "

`"Buna"`, `"ziua"`, `"tuturor"`, `"!"`

Exemplu simplu

```
StringTokenizer hachoir = new StringTokenizer("Bonjour le monde !", "n
    ");
StringBuffer resultat = new StringBuffer();
while (hachoir.hasMoreTokens()) {
    String mot = hachoir.nextToken();
    mot = mot.toUpperCase();
    resultat.append(mot + "\n");
}
```

Another example: Sort words in phrase

```
String data = "Belle Marquise, vos beaux yeux me font mourir d'amour !
    ";
String result = "";
StringTokenizer st = new StringTokenizer(data, " ,!", false);
String[] words = new String[st.countTokens()];
int i = 0;
while (st.hasMoreTokens()) {
    words[i] = st.nextToken();
    i += 1;
}
Arrays.sort(words);
while (i > 0) {
    i -= 1;
    result = words[i] + " " + result;
}
System.out.println(result);
```

Libraries of functions

- ⋆ non instantiable classes
  - constructors are declared `private`
  - al classes are declared `final`
- ⋆ everything is declared `static`

```
public final class MyLibrary {
    public static final Xxx aConstant = ...;
    private MyLibrary() {
        // nothing
    }

    public static Yyy aMethod(...) {
        ...
    }
}
```

java.lang.System

- ⋆ input, output, error streams
- ⋆ program terminations
- ⋆ system properties

java.lang.Math

```
final double E
final double PI
double abs(double)
float abs(float)
int abs(int)
long abs(long)
double max(double, double)
float max(float, float)
int max(int, int)
long max(long, long)
double min(double, double)
float min(float, float)
int min(int, int)
long min(long, long)
```

```
double acos(double)
double asin(double)
double atan(double)
double ceil(double)
double cos(double)
double exp(double)
double floor(double)
double log(double)
double pow(double)
double random()
double rint(double)
long round(double)
int round(float)
double sin(double)
double sqrt(double)
double tan(double)
double toDegrees(double)
double toRadians(double)
```

1. Strings

2. Coding standards

Files

- ⋆ Source files
  - identical name to the public element containing it
  - finishes with .java
- ⋆ Structure of source file:
  - package declaration
  - import ...
  - documentation `/** ... */`
  - definition of the element `... class ...`

Structure of a class

- For each section:
  - start with static elements, then with instance elements
  - public elements first, protected afterwards, private at the end
- Section `// ATTRIBUTES`
- Section `// CONSTRUCTORS`
- Section `// REQUESTS`
- Section `// COMMANDS`
- Section `// METHODS`

Indentation and spacing

- indentation = 4 spaces (exactly)
- Max length for a line = 80 characters
- Expression too long: cut it into multiple lines
  - after a comma
  - before an operator
  - not inside brackets
  - indent at the same level
  - 2 indentations (8 spaces) to mark difference with normal indenting

- 1 space between keyword and paranthesis: `while (condition)`
- 0 space between method name and arguments: `o.m(args)`
- 1 space after each comma in a list: `o.m(arg1, arg2)`
- Every binary operator (except .) must be surrounded by spaces: `x + y`
- 1 space after a casting operator: `(int) 5.2`

Comments

- Documentation comments: for more advanced users
- Simple comment `/* ... */`
- End of line comment `// ...`

Declaration

- one declaration per code line
- Declare tables like: `int[] t`
- Declaration of type, constructor, method
  - opening bracket at the end of first line
  - closing bracket on a new line

Variables

- Local variables
  - declared at the beginning of the block
  - initialize as soon as possible
- instance variables: in constructors

Instructions

- one line of code = one instruction
- composite instructions `while`, `for`, `if-else`
  - open accolade at the end of first line
  - inner instructions = indented
  - accolades even if only one instruction is present
  - no inner instruction: void block, commented
  - closing accolade on new line with previous indentation

Examples

```
if (condition) {
    instructions
} else if (condition) {
    instructions
} else {
    instructions
}
```

```
while (condition) {
    instructions
}
do {
    instructions
} while (condition);
for (init ; cond ; maj) {
    instructions
}
```

Naming

- packets: short name, minuscules
- Classes, interfaces: One majuscule+minuscules
- Constructors: identical to class name
- Methods: first word: minuscules, later words 1 majuscule+minuscules