Java language syntax

Agenda

- Comments
- Variable declaration
- Variable initialization
- Identifiers
- Keywords
- Operators
- Statements
- Blocks
- Primitive data types
- Variable scope
- Basic conditions

Code parts

- Main task of any program is:
  - Data processing
  - Control flow management
- Main syntax elements of program are:
  - Data
  - Statements, which process data
  - Comments
  - Whitespaces

Comments

JAVA supports three types of comments

// One-line commets (usually put at the end of line)

/* Multi-line comments

– or block of comments */

/*- Unformatted block comment
*/

http://en.wikipedia.org/wiki/Java_syntax#Comments

Variables

- **Variables** at any time have some specific value. Variables are defined and referred using their type and **identifiers**, **e.g.**
  int a;
- Variable which value is used as a **constant** is defined using **final** keyword, e.g.:
  final int b;
- **Literals** or **constants** are values, which are defined *inside code* "literally", e.g.:

```
101
-22
3.14
'f'
true
"Some string"
```

Declaration vs Initialization

- Variables can be declared without explicitly assigned values
- IDE will warn about that
  - and if non-initialized variable is used, will get error or exception

```
public class Assign {
   public static void main (String args []) {
     int x, y; // declare two int variables without initial value
     float z = 3.414f; // declare float and initialize value

   }
}
```

Keywords

- JAVA file structure:
  - **package, import, class, enum, interface**
- For inheritance:
  - **extends, implements**
- For conditions and branching
  - **if, else, for, while, do, break, continue, switch, case, default, return**
- For exceptions:
  - **try, catch, finally, throw, throws**
- To check for conditions:
  - **assert**
- For primitive types:
  - **boolean, char, byte, short, int, long, float, double**

Keywords

- To construct and use objects:
  - **new, this, super**
- To check type of variable (actually operator):
  - **instanceof**
- Access modifiers:
  - **public, protected, private**
- Other modifiers:
  - **abstract, final, static, synchronized, transient, volatile, native, strictfp**
- Type of method without returned value:
  - **void**
- Reserved but actually not used keywords:
  - **const, goto**
- Full list of keywords:
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

Operators

- Operators are methods (functions) with specific syntax. Main groups of operators are
  - Arithmetic
  - Bit shift (out of scope in this course)
  - Comparison
  - Logical
  - Assignment
  - Conditional
  - https://www.tutorialspoint.com/java/java_basic_operators.htm

Statements

- Statement is the smallest standalone element that defines executable action
- Meaning of statement depends on keywords operators and variables
  In Java statements are delimited with semi-column **;**

```
totals = a + b + c
                + d + e + f;
```

Data types

Java has primitive and complex data types

- Primitive data
  - Literals or constants
  - Variables
- Objects
  - Built-in objects
  - Custom objects

Identifiers

- ***Identifier*** is name of some element (variable, class, method) in Java code
- Variable name should start with:
  letter, underscore (_) or dollar ($)
- By convention variable starts with lowercase letter
- Variable name is Case sensitive and can be unlimited length
- Example:

  Identifier

  UserName

  user_name

  _sys_var1

  $change

- http://en.wikipedia.org/wiki/Java_syntax#Identifier

Primitive data types

- Java has following primitive data types:
  - Integer numbers: **byte, short, int** and **long**
  - Floating-point numbers: **float** and **double**
  - Single (Unicode, two byte) character: **char**
  - Boolean type with true/false value: **boolean**

Floating point numbers
float and double

- By default assigned value with simple number value is **double**
- Subtype of assigned value can be set explicitly using postfixes:
  - **F** or **f** — float
  - **D** or **d** — double
- Exponential form can be used, using **E** or **e**
- Examples
  - **3.14** — double
  - **6.02E23** — double in exponential form
  - **2.718F** — float
  - **123.4E+306D** — double in exponential form
- http://en.wikipedia.org/wiki/Primitive_data_type
- https://dzone.com/articles/never-use-float-and-double-for-monetary-calculatio

char

- **char** contains one 16-bit Unicode character (~32 of characters).
- Value to char is assigned using single quotes/apostrophe around it(**' '**).
- Examples
  - **'a'** — is letter **a**
  - **'\t'** — is tabulation character
  - **'\u????'** — unicode character
    where **????** are four hexadecimal digits, e.g.:
  - **'\u0100'** is '**ā**'
  - or just enter **'↠' or '☺' or '↞'** (if your display is not black and green anymore)

Logical type boolean

- **boolean** is primitive type with two possible values:
  - true
  - false
- Example:

```
boolean lastElement = true;
```

Range and size of primitive types

Type Name	Kind of Value	Memory Used	Range of Values
byte	Integer	1 byte	–128 to 127
short	Integer	2 bytes	–32,768 to 32,767
int	Integer	4 bytes	–2,147,483,648 to 2,147,483,647
long	Integer	8 bytes	–9,223,372,036,8547,75,808 to 9,223,372,036,854,775,807
float	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
double	Floating-point	8 bytes	$\pm 1.7976931348623157 \times 10^{+308}$ to $\pm 4.9406564584124654 \times 10^{-324}$
char	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
boolean		~ 1 byte	True or false

- The actual size of a boolean variable in memory is not precisely defined by the Java specification.

Type casting

- Sometimes it is necessary to assign value to variable of different data type
- Java support **automatic casting** from "shorter" (by counting stored bytes) primitive types of the same group (integers, floats) to "longer" types
- If target variable is "shorter" than type of assigned value, explicit casting using **(type)** operator is necessary
- Examples:
  - ```
    int intValue = 1954;
    long longValue = intValue; // correct, automatic casting
    long bigValue = 99L;
    int squashed = bigValue;  // incorrect, need explicit casting
    int squashed = (int) bigValue; // correct
    int squashed = 99L; // incorrect, need casting
    int squashed = (int) 99L; // correct, but redundant
    int squashed = 99;   // correct
    ```
    ```

Type casting caveats

- **boolean** cannot be casted to any other primitive type
- **char** can be casted to any other primitive type
- In general, value can be casted, if number of bytes for value storage on left side (for variable) is equal or larger than for value at the right side
- Examples:

```
char charval = 'x';

int intval = charval; // correct, intval will hold value
                      // of Unicode character

double z = 12.414F;   // correct double is larger than  float

int z1 = (int)z;      // will work, but digits after
                      // decimal point will be lost

long l = 99999999999999999l;
double d = (double) l; // will work, but digits before
                       // decimal point will be lost
```

Control flow statements

Control flow statements control branches and cycles of execution flow.

- Branching statements
- Cycling statements
- Conditional statements

Branching statements if and else

General syntax:

```
if (boolean expression) {
  statement or block;
}
if (boolean expression) {
  statement or block;
}
else if (boolean expression) {
  statement or block;
}
else {
  statement or block;
}
```

Branching statement switch

General syntax:

```
switch (expr1) {
      case constant2:
            statements;
            break;
      case constant3:
            statements;
            break;
      default:
            statements;
            break;
}
```

break is necessary to exit from successful match and not check following cases!

Strings in switch

- Starting with **Java 7** Instead of:

```
if (input.equals("yes")) {
 return true;
} else if (input.equals("no")) {
 return false;
} else {
AskAgain();
}
```

- you can write:

```
switch(input) {
 case "yes": return true;
 case "no": return false;
 default: askAgain();
}
```

C-like for

General syntax:

```
for (init_expr; boolean testexpr; alter_expr)
{
    statement or block;
}
```

Example:

```
for (int i = 0; i < 10; i++) {
 System.out.println("Are you finished yet?");
}
System.out.println("Finally!");
```

for (each item) **in collection**

- Starting with **Java 5** you can use **for** (each item) loop, which iterates over each element in a collection:

```
public class Program {
    public static void main(String[] args) {

        String[] values = new String[3];
        values[0] = "Dot";
        values[1] = "Net";
        values[2] = "Perls";

        for (String value : values) {
            System.out.println(value);
        }
    }
}
```

Cycle statement while

General syntax:

```
while (boolean) {
      statement or block;
}
```

Example:

```
int i = 0;
while (i < 10) {
     System.out.println("Are you finished yet?");
     i++;
}
System.out.println("Done");
```

Cycle statement do ... while

General syntax:

```
do {
    statement or block;
} while (boolean test);
```

Example:

```
int i = 0;
do {
     System.out.println("Are you finished yet?");
     i++;
} while (i < 10);
System.out.println("Done");
```

Control statements
break and continue

General syntax:

```
break [label];
continue [label];
label: statement;
```

Where

`Label:` is optional label in code

`statement` usually is cycle statement

Label can be added at the start of any line of code, but ordering of execution flow has limits.

continue — orders execution flow to the start of cycle or labeled line.

break — order execution flow to next line after cycle or to labeled line.

Statement break

General usage of break:

```
do {
        statement;
        if (condition is true) {
          break;
        }
        statement;
} while (boolean expression);
```

Statement continue

General usage of continue:

```
do {
     statement;
     if (boolean expression) {
          continue;
     }
     statement;
} while (boolean expression);
```

continue do not break execution of cycle, but it interrupts execution of lines after **continue** statement inside cycle

Reference variables

- Object variables are also called reference type variables, because they refer to place where objects are located
- When variable of custom Object is defined, reference variable is created which points to null.
- When object is created reference refers (points) to place in memory, where object is created
- **Example:**
  ```
  String user = new String("user");
  ```
  `user` is reference to object which contains string "user"

Reference example

Reference variables do not store object value, but address (reference) to the object

```
Point p = new Point(1.0,2.0);
Point q = p; // p and q are two references to the same object
Point r = p.clone(); // r is another copy of the previous object

p.x = 13.0;

System.out.println(q.x);
System.out.println(r.x);
```

Arrays

- Array is data structure, consisting of elements with the same type.
- Arrays have name and their elements have an index.
- Arrays can be one dimensional or multi dimensional.
- Array is declared using type of its elements, name and dimensions in form:

  type var-name[ ];
- Array is initialized using form

  var-name = new type[n];
- Or in one line with declaration and initialization in the same line:

  type var-name = new type[n];

  Where
  - **type** — built-in or custom Java type
  - **var-name** — name of the array;
  - **n** – number of elements in array
- Example code:

```
int month_days[ ] = new int[12];
```

- If array is just declared but not initialized, it refers to ***null***.

Multi-dimensional arrays

- Multi-dimensional arrays are arrays of arrays
- Elements of arrays should be initialized/assigned starting from left side of dimensions, example:

```
int twoDim [][] = new int [4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5]; // correct

int twoDim [][] = new int [][4]; // wrong
// 1st dimension is not initialized yet
```

Multi-dimensional arrays

- Multi-dimensional arrays can be with irregular size:

```
int twoDim [][] = new int [4][];
twoDim[0] = new int[2];
twoDim[1] = new int[4];
twoDim[2] = new int[6];
twoDim[3] = new int[8];
```

- Regular multi-dimensional arrays is just special case
- Regular multi-dimensional arrays can be initialized setting their dimension sizes on declaration:

```
int twoDim[][] = new int[4][5];
```

Class `java.lang.String`

- **String** is immutable (non-changeable) Java built-in Object
- **String** (in difference with **char**) *is not primitive type* and is also not array of characters
- String contains Unicode characters
- String has overridden methods for convenience:
  - No need for explicit construction
  - Value of string can be assigned using literal enclosed in double quotes ("")

```
String user = new String("user");
is the same as:
String user = "user";
```

- **Examples**:

```
String s1 = "The quick brown fox jumps over the lazy dog.";
String s2 = "This is\n"
+ "multi line\n"
+ "value of the\n"
+ "String.";
```

Operator "+" for String means concatenation (appending) of values

Other String operation methods

- **concat, replace, substring, toLowerCase, toUpperCase, trim;**
- **endsWith, startsWith, indexOf, lastIndexOf;**
- **equals, equalsIgnoreCase, compareTo;**
- **charAt;**
- **length**