COMP9103: Software Development in Java

W2: Java Basics and Primitive Data Types

Presented by
Dr Ali Anaissi
School of Computer Science

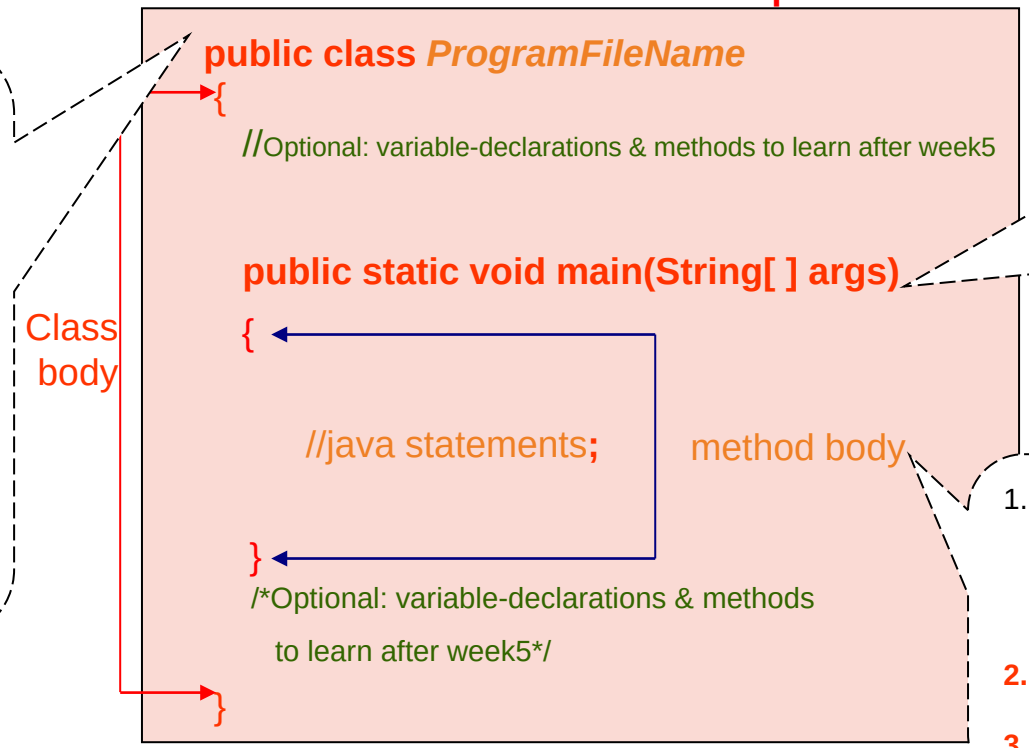
THE UNIVERSITY OF SYDNEY

Java Basics

Java Program Structure

// comments are for human and are ignored by compiler
/* the name of a program source file **MUST** be the **SAME** as the name of the **public class** */

1. A program is made up of one or more **classes;**
2. A source file can contain **at most ONE public class**
3. **The name of the public class MUST match the name of the file containing the public class**. e.g. the public class ***HelloWorld*** must be saved in the file ***HelloWorld.java***



1. A class contains one or more **methods**
2. A Java application always contains a method called **main() which is starting point of the running program**

1. Contains a collection of instructions to define how to handle a given task, e.g. System.out.println("Hello World!");
2. **Each statement ends with a semicolon (;)**
3. **Java is case sensitive**

Identifiers

- **Identifiers**
  - Words to be used in a program, e.g., name of a class, a method, or a variable
  - **Rules for identifiers in Java**
    - ✓ **Can be made up of letters, digits, the underscore (_) character, and the dollar sign ($)**
    - × **Cannot start with a digit**
    - × **Space is not permitted inside an identifier**
    - × Cannot use reserved words such as ***public, class, static, void, main***
    - **Case sensitive**: for instance, *Total*, *total* and *TOTAL* are different identifiers
    - Conventionally, **different case styles for different types of identifiers**, for instance,

> - **Title case for class names:** ***HelloWorld***
> - **Upper case for constants:** ***MAXIMUM***
> - **Variable names and method names should start with lowercase letter:** ***sum, main***

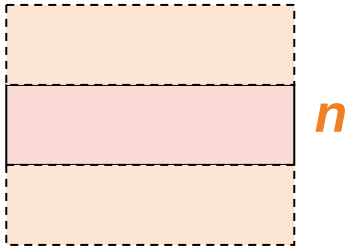
Identifiers/names should be descriptive and readable

Variables

- **A variable represents a named space in memory that can hold data.**
  - You can think of a variable as a container or box where you can store data
  - Since the data in the box (variable) may vary, a variable may contain different data values at different times during the execution of the program, but it always refers to the same location
- Variable declaration syntax

  ***dataType* variablename;**

  e.g. : ***int n;*** // declares a variable ***n*** of integer type.
- When the computer executes a variable declaration, it
  - sets aside memory for the variable
  - associates the variable's name with that memory.



Only after a variable has been declared first, then it can be used in a program

Java Primitive Types

There are **8 primitive data types** in Java

- 1 character type:
  - ✓ *char*
- 4 integer types:
  - ✓ *byte*
  - ✓ *short*
  - ✓ *int*
  - ✓ *long*
- 2 floating point types:
  - ✓ *float*
  - ✓ *double*
- 1 Boolean type:
  - ✓ *boolean*

Literals

- A **literal** is an explicit data value in the source code.
- Examples:
  - Values for integers/floating-points
  - Values of textual strings

```
public class Cube {//calculate the volume of a cube
   public static void main(String[] args) {
      double length = 3.0;
      double width  = 4.0;
      double height = 5.0;
      System.out.println(length*width*height);
   }
}
```

Numeric literals

Variable Assignment

- **Assignment (=)**: to store a value into a variable

  count = 500; //save value of 500 to the box with name "count"

- Once you saved a data value to a variable, you can read and change it later.

- When a new value is assigned to the same variable, the previous value will be overwritten by the new value

  count = 700; //change the value of variable "count"

```
public class VariableEx {
    public static void main(String[] args)
    {
        int count;
        double xsz, ysz;
        double x, y;
        count=500;
        System.out.print(count);
    }
}
```

```
public class VariableEx {
    public static void main(String[] args)
    {
        int count;
        double xsz, ysz;
        double x, y;
        count=500; //save 500 into count
        System.out.println(count);
        //output is 500
        count=700; //save 700 into count
        System.out.print(count); //output 700
    }
}
```

Assignment, Increment, & Decrement

- The right- and left-hand sides of an assignment statement can contain the same variable

Firstly, 1 is added to the original value in *count*

Then the addition result is saved in *count*, and the original value in *count* is replaced by the new value

```
Count = 700;
Count = count+1;
/* means 700+1(=701) →count*/
```

- Increment operator (++)
  - Adds 1 to its operand
  - count++; //same as count=count+1;
- Decrement operator (--)
  - Subtracts 1 from its operand
  - count--; //same as count=count-1;

Variable Initialization

- **Initialization** means Declaration & Assignment in one go
  - *dataType* variableName = expression/value;

```
public class VariableEx {
    public static void main(String[] args) {
       int sum = 700;
         /*equivalent to:
           int sum;
           sum=700;
         */
       System.out.print(sum);
       sum=105;
    }
}
```

Constants

- Once defined, the value of a constant cannot be changed durreserveding its entire existence
- The word "final" indicates a constant
- Constant Declaration
  - ***final type* NAME_IN_UPPERCASE = value;**

E.g.: final int MINIMUM = 69;

✘ MINIMUM=71; //the value of a constant cannot be changed

- Named constants make programs easier to read and maintain

```
double interest = balance * 0.07;            //not very clear
final double INTEREST_RATE = 0.07;
interest = balance * INTEREST_RATE;   //much clearer;
balance = balance * (1+ INTEREST_RATE);
```

think about when INTEREST_RATE raises to 0.08, how to modify the statements???

Primitive Data Types

Character type

- Character type
  - A ***char*** variable stores **a single character** from Unicode encoding scheme (*)
  - **Character literals** are delimited by **single quotes (' ')**:

    'X' 'b' '&' 'a'
  - **Define char variables and constants**:

```
char topGrade = 'a';
final char TEMINATOR = ';';
```

* Unicode Encoding Scheme
  - The Unicode encoding scheme uses **16 bits per character**, allowing for 65,536 unique characters
  - It is an international character set containing symbols and characters from many languages

Example

```
//char type variables and constants
public class CharTypeEx1 {
    public static void main(String[] args) {

        final char DISTINCTION = 'd' ;
        final char CREDIT = 'c';
        char grade = CREDIT;
        System.out.print(grade);
        grade = DISTICTION;
        System.out.print(grade);
        CREDIT='p';       //error: constants cannot be changed
    }
}
```

Integer Types

- Used to represent whole numbers without fractional/decimal part
- The difference between the various numeric primitive types is their sizes, which will therefore affect the value ranges they can express

Type	Description	Size	Value range
byte	A single byte	1 byte (8 bits)	**-128 ~ 127**
short	Short integer type	2 bytes (16 bits)	**-32768 ~ 32767**
int	Integer type	4 bytes (32 bits)	Big range of values
long	Long integer type	8 bytes (64 bits)	Very big range of values

- Examples:

```
int answer = 42;
final int  SMLNUM=255;
byte wrongnum = 128; //
```

compiling error : Type mismatch: cannot convert from int to byte

Floating-point Types

- Used to represent real numbers

Type	Size
float	4 bytes (32 bits)
double	8 bytes (64 bits)

- **NOTE:** Java assumes that all floating-point literals are of ***double*** type.
- If we need to treat a floating-point literal as a ***float***, we need to append an ***F*** or ***f*** to the end of the literal
  - E.g., *float* ratio = 0.236**F**; → A ***float*** literal using 32 bits
    double delta = 453.7; → A ***double*** literal using 64 bits

Boolean Type

- A ***boolean*** variable uses 1 bit to represent a true or false value
- The ***boolean*** type has just two literal values: *true* and *false*
- The reserved words of *true* and *false* can be assigned to a boolean variable

  e.g., *boolean* done=*false*;

Boolean literal

Expressions

- Java provides a rich set of expressions:
  - Arithmetic
  - Relational
  - Logical
  - String related
  - Bit level

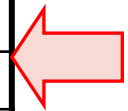
Arithmetic Expressions

- Java provides the usual set of arithmetic operators:

+	addition	Add numbers together (3+4)=7
-	subtraction	Subtract one number from another (5-2)=3
*	multiplication	Multiply two numbers (2*3)=6
/	division	Divide one number by another (18/2)=9
%	modulus (remainder)	The remainder of one number divided by another (19%2)=1

- Remainder (modulus) operator (%) ***returns the remainder*** after the exact division. E.g.,

expression	result
17 % 4	1
-20 % 3	-2
10 % -5	0
3 % 8	3



17=4*4 + ***1***

-20=-6*3 + (***-2***)

10=2*(-5)+ ***0***

3=0*8 + ***3***

Arithmetic Operators and Casting

- Division (/) operator
  - **If both operands are integers, the result is an integer,** and the remainder (or fractional part) is discarded – this is known as **integer division.**
  - E.g.: 7/4 yields 1 (7=***1****4+3 and 3 is discarded)
- In order to avoid discard the fraction part, you need to use the cast operator:
  - E.g.: (float)7/4 yields 1.75

Data Conversion

- **Data conversion: to convert data from one type to another**
  - Three ways to do data conversion
    1. Automatic assignment conversion
    2. Automatic promotion
    3. Explicit casting

- **Automatic assignment conversion**
  - is used to convert a small data type to a larger data type
  - E.g.:

```
float balance=13.75f;
double dollars = balance;
//float to double
```

Data Conversion

- **Automatic Promotion**
  - Happens automatically in expression to convert a smaller data type to a larger data type
  - For example:

```
float sum=101.8f;
int   count=2;
double result;
result=sum/count;    //count is promoted to a float value;
                     //sum/count (float type) is then converted to double type via assignment
```

- **Explicit Casting**
  - Converts a value/variable to another type by an explicit declaration
  - Put the new type in parentheses in front of the expression/value to be converted:

    (type) expression
  - For example:

```
double balance=13.75;
int dollars = (int) balance * 4;
```

Different!

```
double balance=13.75;
int dollars = (int) (balance * 4);
```

Relational Expressions

- Java provides the following relational operators:
  - The result of relational operation is a Boolean value (either ***true*** or ***false***)

Math Notation	Name	Java Notation	Java Examples
=	Equal to	==	`balance == 0` `answer == 'y'`
≠	Not equal to	!=	`income != tax` `answer != 'y'`
>	Greater than	>	`expenses > income`
≥	Greater than or equal to	>=	`points >= 60`
<	Less than	<	`pressure < max`
≤	Less than or equal to	<=	`expenses <= income`

Example

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

        int year = 2013;
        boolean newcentury;
        newcentury=((year%100)==0);
        System.out.print(newcentury);

    }
}
```

Logical Operators

- Java provides the following logical operators:

Name	Java Notation	Java Examples
Logical *and*	&&	`(sum > min) && (sum < max)`
Logical *or*	||	`(answer == 'y') || (answer == 'Y')`
Logical *not*	!	`!(number < 0)`

- The truth table as follows

Value of *A*	Value of *B*	Value of *A* && *B*	Value of *A* || *B*	Value of ! (*A*)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Useful Classes

String Class

- **String**: a sequence of characters enclosed in double quotation marks
  - Examples: **"Hello World!"; "This is a Java String"; "X"**
- Concatenation operation (+)
  - The string concatenation operator (**+**) is used to append one string to the end of another string
    - Example: **"Enjoy " + "your " + "university life!";**
      → "Enjoy your university life!"
  - If one of operands is a string, + operator performs string concatenation
    - e.g.

Expression	value
"Hi, " + "Bob"	"Hi, Bob"
"1" + " 2 " + "1"	"1 2 1"
"1234" + " + " + "99"	"1234 + 99"
"1234" + "99"	"123499"

  - If operands are numbers, + operator performs addition

Example

```
public class MyString{
    public static void main(String[] args) {
      String s1 =  “Hello” + “ World” ;
      System.out.println(s1);
      int i = 35, j = 44;
      System.out.println("The value of i is " + i +
                         " and the value of j is " + j);
    }
}
```

```
Hello World!
The value of i is 35 and the value of j is 44
```

Arrays

Arrays

- Arrays can be used to store a number of elements of the same type:

```
int[] a; // an uninitialized array of integers
float[] b; // an uninitialized array of floats
String[] c; // an uninitialized array of Strings
```

- *Important*: The declaration does not specify a size. However, it can be inferred when initialized:

```
int[] a = {13, 56, 2034, 4, 55};          // size: 5
float[] b = {1.23F, 2.1F};                // size: 2
String[] c = {"Java", "is", "great"};     // size: 3
```

Arrays

- Other possibility to allocate space for arrays consists in the use of the operator **new**:

```
int i = 3,  = 5; double[] d;  // uninitialized array of doubles
d = new double[i+j];
// array of 8 doubles
```

- Components of the arrays are initialized with default values:
  - 0 for numeric type elements,
  - '\0' for characters
  - **null** for references.

Arrays

- Components can be accessed with an integer index with values from 0 to length minus 1.

```
a[2] = 1000; // modify the third element of a
```

- Every array has a member called **length** that can be used to get the length of the array:

```
int len = a.length; // get the size of the array
```

Command-line Arguments

- We have seen that the method **main** has to be defined as follows:

```
public static void main(String[] args)
```

- Through the array argument, the program can get access to the command line arguments

Example: Command-line Arguments

```
//*************************************************************************
//  using a command-line argument
//*************************************************************************

public class UseArgument {
   public static void main(String[ ] args) {
    System.out.print("Hi, ");
    System.out.print(args[0]);
    System.out.println(". How are you?");
   }
}// end of class
```

Declare the command-line arguments

Use the command-line arguments

Compile: *javac UseArgument.java*
Execute: *java UseArgument **Alice***
Display: Hi, **Alice**. How are you?

Execute: *java UseArgument **Bob***
Display: Hi, **Bob**. How are you?

Converting strings to primitive values for command-line arguments

- Command-line arguments are strings
- Java provides the library methods to convert strings that we typed as command-line arguments into numeric values of primitive types:
  - **Integer.parseInt()** and **Double.parseDouble()** are used to convert a string on the command line to *int* literal and *double* literal

	Library method	function
int	Integer.parseInt(String s)	Convert *s* to an **int** value
double	Double.parseDouble(String s)	Convert *s* to an **double** value
long	Long.parseLong(String s)	Convert *s* to an **long** value

Example: Command-line Arguments

```
public class Swap {
/* use the command-line inputs*/
   public static void main(String[] args) {
      int a,b,t;
      a = Integer.parseInt(args[0]);
      b = Integer.parseInt(args[1]);
      System.out.println(a+" "+ b);
      t = a;
      a = b;
      b = t;
      System.out.println(a+" "+ b);
   }
}
```

Compile: *javac Swap.java*
Execute: *java Swap **4 9***
Display: ***9 4***

Useful Classes: Math

- Math class (*public class Math*) contains methods for more complex calculations
- For instance,
  - To compute x^n, we write: ***Math***.*pow(x,n);*
  - To take the square root of a number, we use: ***Math***.*sqrt(x);*
  - Mathematical Methods in JAVA

$$\sin^2 x + \cos^2 x$$

Math.pow(**Math.**sin(x),2)+**Math.**pow(**Math.**cos(x),2);

Math.sqrt(x)	Square root
Math.pow(x,y)	Power x^y
Math.exp(x)	e^x
Math.log(x)	Natural log
Math.min(x,y) Math.max(x,y)	Minimum value Maximum value
Math.round(x)	Closest integer to x
Math.sin(x) Math.cos(x) Math.tan(x)	Sin(x) Cos(x) Tan(x)

Example: casting to get a random integer

```
/*  Prints a pseudo-random integer in the range of [0, N-1).
 *  Illustrate an explicit type conversion (cast) from double to int.*/

public class RandomInteger {
    public static void main(String[] args)
    {
        final int N = Integer.parseInt(args[0]);

        double r = Math.random(); // a pseudo-random real between 0.0 and 1.0 exclusive

        int n = (int) (r * N); // a pseudo-random integer between 0 and N-1

      System.out.println("Your random integer is: " + n);
    }
}
```

double to int (cast)

int to double (automatic promotion)

Appendix 1: Reserved Words

- Words reserved for special purposes in Java language and can only be used in the predefined way.
- A reserved word cannot be used for naming a variable, a class or a method.

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

Appendix 2: Operator Precedence

Operator	Description	Level	Associativity

`[]`
`.`
`()`
`++`
`--`
	access array element
	access object member
	invoke a method
	post-increment
	post-decrement
		1	left to right

`++`
`--`
`+`
`-`
`!`
`~`
	pre-increment
	pre-decrement
	unary plus
	unary minus
	logical NOT
	bitwise NOT
		2	right to left

`()` `new`	cast object creation	3	right to left

`*`
`/`
`%`
	multiplicative	4	left to right

`+ -` `+`	additive string concatenation	5	left to right
`<< >>` `>>>`	shift	6	left to right

`< <=`
`> >=`
 	
`instanceof`
	relational
	type comparison
		7	left to right

Operator	Description	Level	Associativity
`==` `!=`	equality	8	left to right
`&`	bitwise AND	9	left to right
`^`	bitwise XOR	10	left to right
`|`	bitwise OR	11	left to right
`&&`	conditional AND	12	left to right
`||`	conditional OR	13	left to right
`?:`	conditional	14	right to left

`= += -=`
`*= /= %=`
`&= ^= |=`
`<<= >>= >>>=`
	assignment	15	right to left

Appendix 3: Math Library

`public class Math`

`double abs(double a)`	*absolute value of a*
`double max(double a, double b)`	*maximum of a and b*
`double min(double a, double b)`	*minimum of a and b*

Note 1: `abs()`, `max()`, *and* `min()` *are defined also for* `int`, `long`, *and* `float`.

`double sin(double theta)`	*sine function*
`double cos(double theta)`	*cosine function*
`double tan(double theta)`	*tangent function*

Note 2: Angles are expressed in radians. Use `toDegrees()` *and* `toRadians()` *to convert.*
Note 3: Use `asin()`, `acos()`, *and* `atan()` *for inverse functions.*

`double exp(double a)`	*exponential* (e^a)
`double log(double a)`	*natural log* ($\log_e a$, *or* $\ln a$)
`double pow(double a, double b)`	*raise a to the bth power* (a^b)
`long round(double a)`	*round to the nearest integer*
`double random()`	*random number in* $[0, 1)$
`double sqrt(double a)`	*square root of a*
`double E`	*value of e (constant)*
`double PI`	*value of* π *(constant)*

See booksite for other available functions.

Excerpts from Java's mathematics library

Questions?