Java language

Object Oriented Programming - Java
Chapter 2

FPTU Da Nang – IT Department

Objectives

- Completing this chapter, students will be able to:
  - Understand and use proficiently the basic components of the Java programming language
  - Variables and data types
  - Primitive and reference types
  - Control command structures
  - Function definition and invocation
  - Writing Java Applications
  - Professional naming technic

Content

- Character sets, keywords, captions
- Variables & Constants
- Datatypes
  - Primitive
  - Reference
- Style conversion & style compression
- Command structures
  - Assignment
  - Branching
  - Loop
- main() function
- *Practice* : Array of integers: sum, max, min, sort

Character set - keywords - comment

- Character set
  - Characters, numbers, some symbols..
  - Case sensitive
- About 50 keywords
  - Distinguish keywords & standard names
- Naming convention – highly recommend!!!
  - **Class name starts with a capital letter**
  - **Variable and function names start with a lowercase letter**
  - **Name should be meaningfull, concate words with 1 capital letter**
    Example: int numberOfStudent, class PhanSo
- Comment
  - On 1 line after the symbol // comment
  - Multiple lines: enclose /* comment */

Variables & Constants

- A variable is **a memory location** associated with a name and a data type.
- Variable declaration syntax
  - dataType varName1, varName2=value, VarNameN;
  - Ex: int a,b=6,c; char grade='E';
  - End with a semicolon ;
- Location of the variable declaration determines the scope of the variable
  - Local variables - declared in a function or block {}, function argument
  - Class variables (class scope)
  - Do not allowed variables with same name in the same scope
- Constants are variables declared after ***final*** keyword
  - Must initialize value up on declared & not allowed to change
  - ***Ex:*** final int A=10;

Variables & Constants

- Notes when declaring variables.
  - Data range of the variable value
  - The size of memory allocated.
  - Limit the data errors.
  - Data abstraction – data encoding
- Example: declaring a variable representing gender
  - boolean gender = true;
  - String gender="Male";
  - char sex='F';
  - byte gioiTinh = 1;
  - int gender=0;
  - double gioiTinh= 0.0;

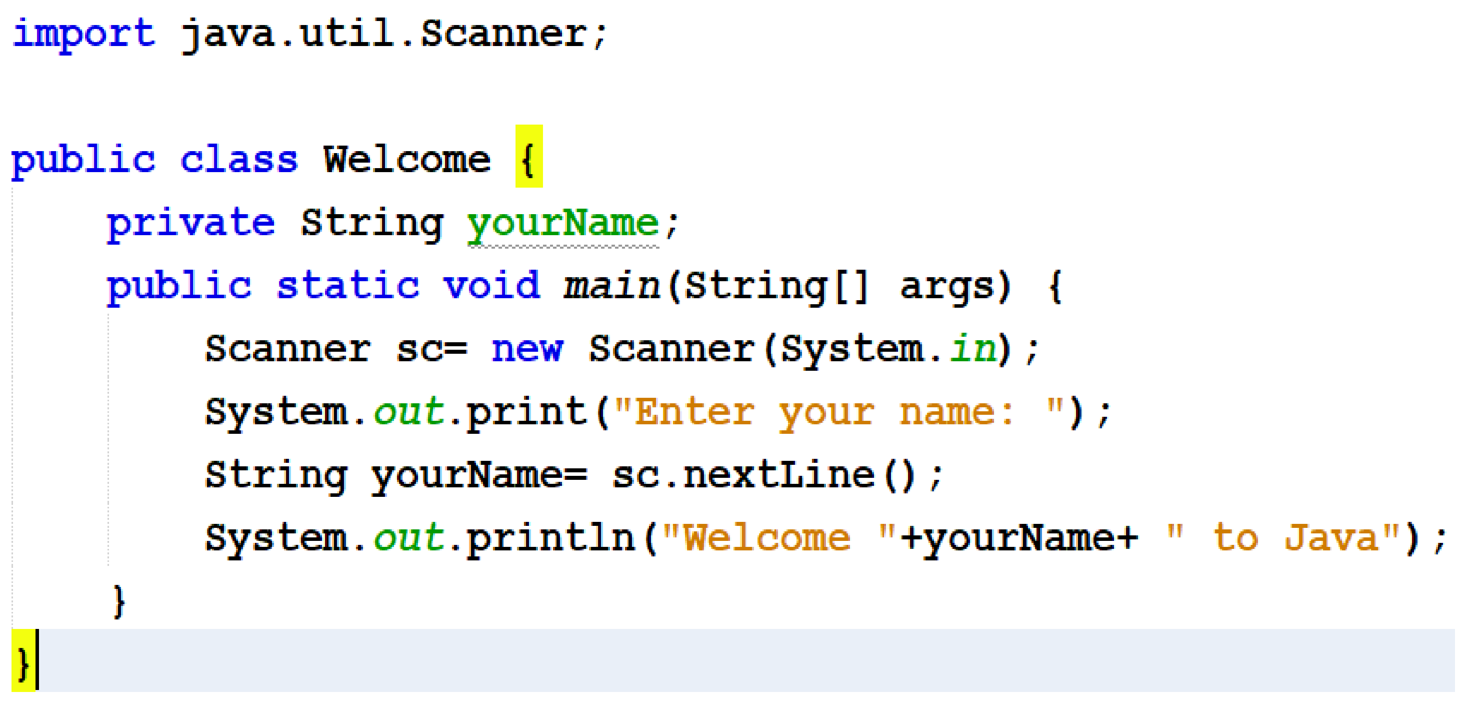
Which declaration is not valid? Which declaration should be used?

Variables & Constants

- Data member variables – class variables
  - Receive the default value if not initialized
  - Effective inside the class
  - Declare inside class, outside method/functions
- Local variables
  - Must be initialized before accessing
  - May have the same name of class variables – will hidie class variables
  - Local variables inside blocks & functions **cannot** have the **same name**
- The data type of the variable specifies how to initialize & assign the variable
  - Primitive variables only need to be assigned
  - Reference type variables must be initialized and assigned
- **Member variable of a class** has the same name *with local variable* is distinguished using the syntax **this.varName** and ***varName***

Variables & Constants..

- How many variables of various types are there in the code below?
- How many classes are there? how many functions mentioned?.



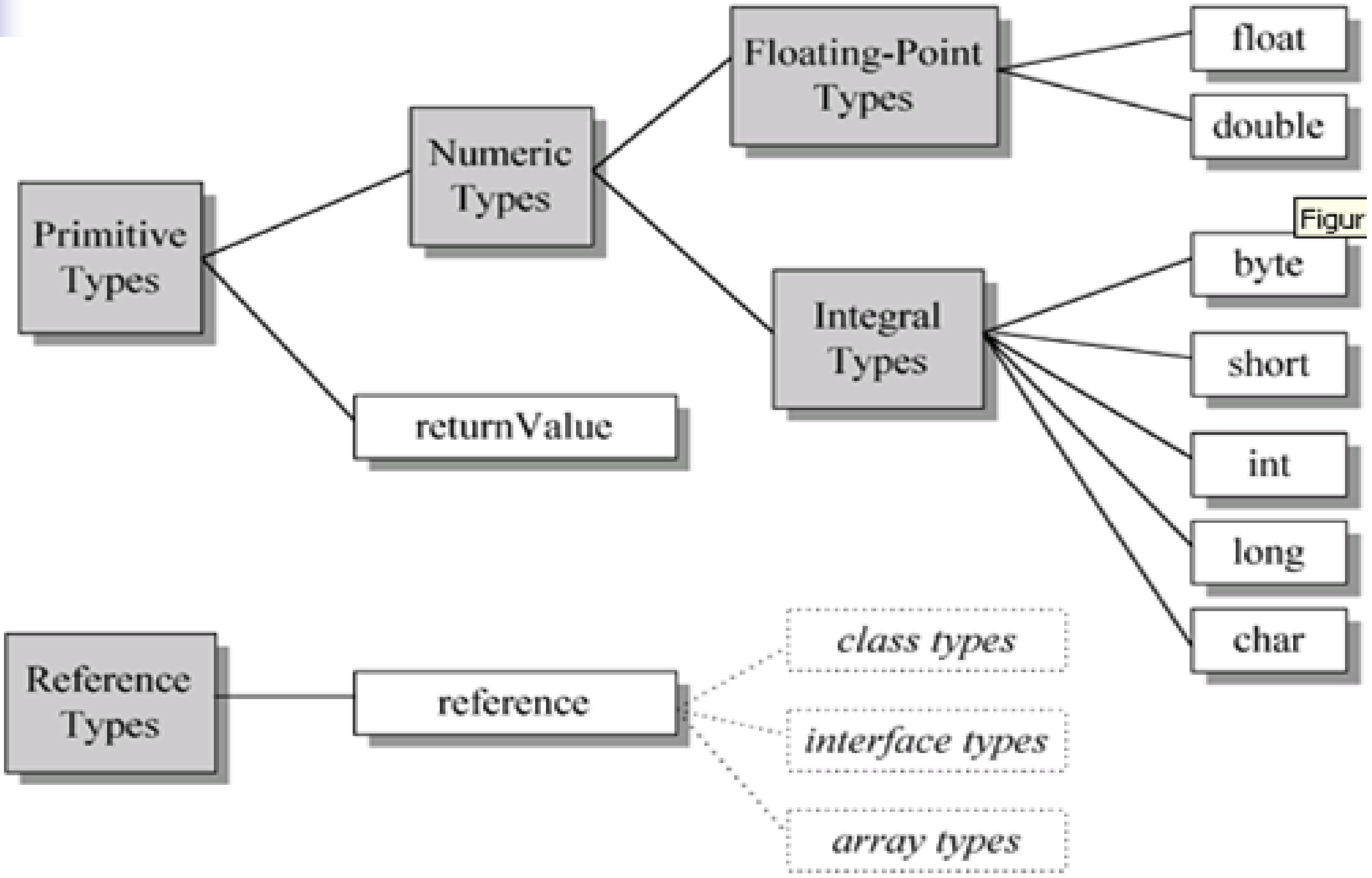
```
import java.util.Scanner;

public class Welcome {
    private String yourName;
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        System.out.print("Enter your name:  ");
        String yourName= sc.nextLine();
        System.out.println("Welcome "+yourName+ " to Java");
    }
}
```

Data types

- 4 characteristics of data types
- Two types of data types:
  - Primitive types: memory location that stores data
  - Reference types: memory loaction that stores the address of a data location
- Primitive type
  - Logic type : boolean
  - Number type
    - **Integer** : **byte, short, int, long**
    - Real: float, double.
  - Character type: **char**

Data types


Primitive Types
Numeric Types
Floating-Point Types
float
double
Figur
Integral Types
byte
short
int
long
char
returnValue
Reference Types
reference
class types
interface types
array types

Logic Data Type

- boolean
- true, false
- 1 byte/bit
- For example
  - boolean isAbsent = true;
  - boolean sex = false;
- Logical operations run from left to right until the value of the logical expression is determined.
  - Math operations &&, || , !, compare returns logical value
  - if (x!=0 && y++/x>=3)... Calculation stops if x=0

Boolean Operators

Operator	Description	`true` Example	`false` Example
<	Less than	3 < 8	8 < 3
>	Greater than	4 > 2	2 > 4
==	Equal to	7 == 7	3 == 9
<=	Less than or equal to	5 <= 5	8 <= 6
>=	Greater than or equal to	7 >= 3	1 >= 2
!=	Not equal to	5 != 6	3 != 3

Integer

Type	Minimum Value	Maximum Value	Size in Bytes
byte	-128	127	1
short	-32,768	32,767	2
int	-2,147,483,648	2,147,483,647	4
long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	8

Double

- **Range**:
  - **float –32 bit - can be precise to 6 or 7 digits**
  - **double – 64 bit -accurate to 14 or 15 digits**
- The following value constants represent special values (defined in the Float and Double classes)
  - Float.NaN
  - Float.NEGATIVE_INFINITY
  - Float.POSITIVE_INFINITY
  - Double.NaN
  - Double.NEGATIVE_INFINITY
  - Double.POSITIVE_INFINITY

Numeric operations

- **All numeric data use the following operators:**
  - Arithmetic operations +, - , * , /
    - byte a=3, b=2,c;
    - c=a*b; /* error multiplication a*b automatically converts to int, out of range of c */
    - 15/2 equals 7 - 15.0/2 equals 7.5
  - Remainder division **%**
    - 5%3 =2
    - 3.14%2.2=0.94
  - Increment **++ and Decrement --** , before & after
    - int a=5, b=7,c; c=a++ +b; // c=12 a=6 c= ++a +b; // c=13 a=6
  - Combination of assignment and arithmetic +=, -=, *=, /=, %=
    - x+=3; equivalent to x=x+3

1/3 + 1/3 +1/3 = ?

1.0/3+1/3.0+1.0/3.0==1 (?)

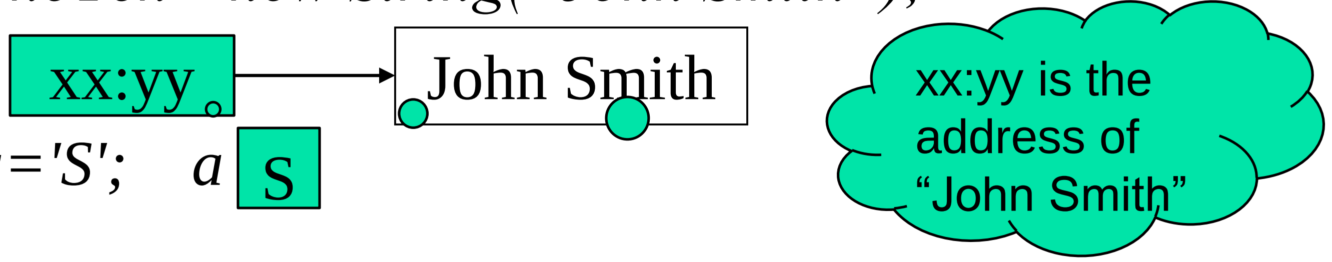
- **The following operations apply to integer types only**
  - Bit shift left, right <<, >>, and >>>
  - Bitwise operations & -and, | -or, ^ -xor, ~ -not

Character data type

- unsigned integer type
  - char
  - using 16-bit Unicode encoding character set
  - Character value between parentheses: 'a'
- Some special characters
  - \t tab key
  - \r to the beginning of the line
  - \n down the line
  - The \" is placed before special characters when you want to print the characters themselves \" , \'

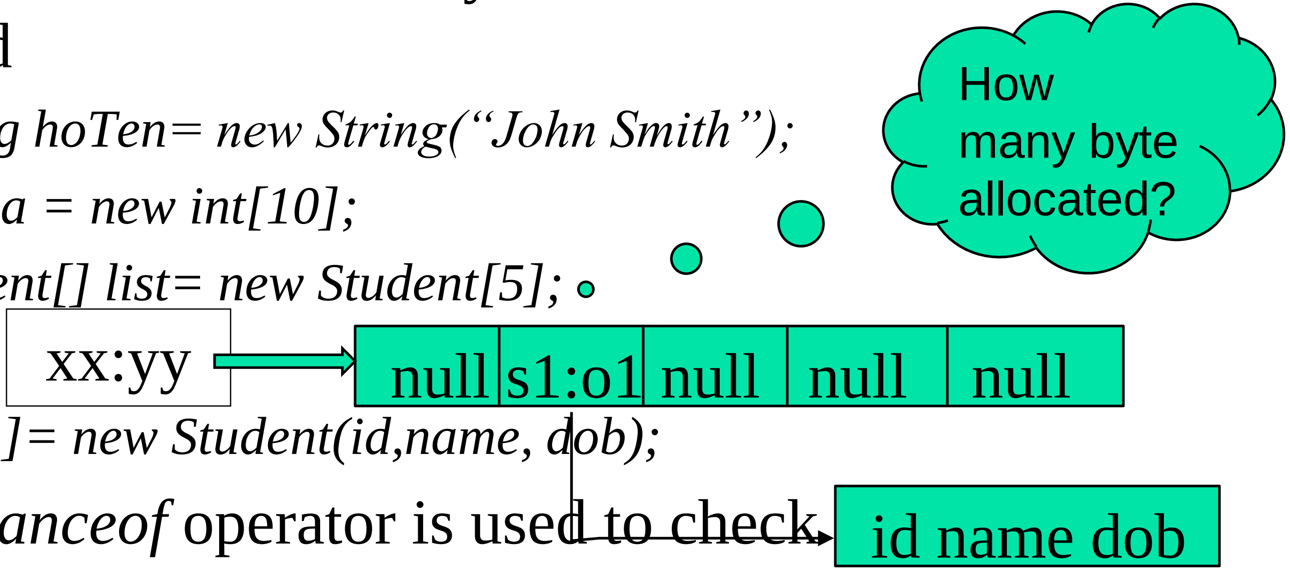
Reference type

- Pointer data types
- Includes classes, interfaces, and arrays.
- The value of a reference type variable is the address refer to an object.
  - An object is an instance variable of a class or array.
  - A reference variable is actually a pointer (address of memory location) pointing to an object,
  - *String hoTen= new String("John Smith");*
  - *hoTen*
  - *char a='S'; a*



Reference type

- The ***null*** constant refers to an empty or uninitialized reference variable.
- Reference variables usually must be initialized with the new . keyword
  - *String hoTen= new String("John Smith");*
  - *int[] a = new int[10];*
  - *Student[] list= new Student[5];*
  - *list*
  - *list[1]= new Student(id,name, dob);*



- The *instanceof* operator is used to check whether a reference variable belongs to a *class*?
  - *a instanceof Integer // returns true if a is an instance variable of Integer*

Array type

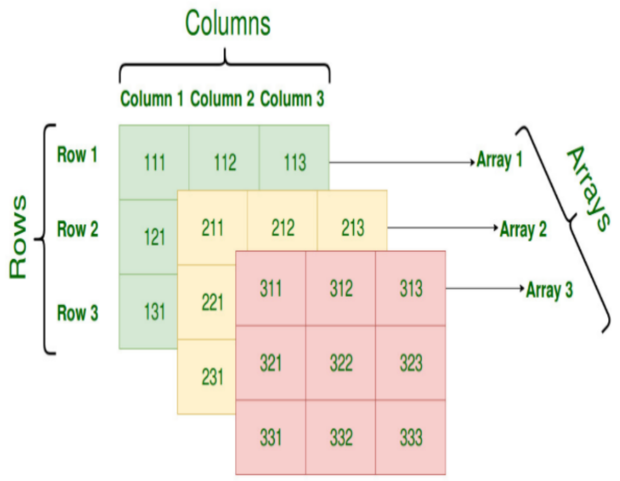
- Declare:
  - Syntax
    - Tenkieu[] tenbien; or Tenkieu tenbien[];
    - Tenkieu[] tenbien= new tenkieu[sophantu];
    - Tenkieu[] tenbien={comma-separated array elements};
  - For example
    - int a[]; //declare variable a as integer array, uninitialized
    - String b[]={“Java”,”Sun Mircosystem”,”Oak”,”1995”} Integer c[]=new Integer[10]; Integer d=new Integer(10);
  - The number of array elements is constant after initialization.
  - The new keyword requires memory allocation for the array

Array type

- *Array element reference*
  - using element index, starting from 0 to n-1
  - References beyond the array index are not allowed
  - using [] notation: a[0]=12;
- Array is a special class – an array is an object
  - The length property specifies the number of elements of String array
    a[]= {“a”,”b”,”c”,”def”} -> a.length equal to 4
  - Copy array: System.arraycopy(srcArr, fromIndex, destArr, toIndex, elementCount);
  - Sorting array: Arrays.sort(arrVar); ascending sort array with comparable elements type.
  - Multidimensional array is actually an array of arrays, arrays elements do not necessarily have the same number of elements..

Array type

- *Assignment of two array variables of the same element type*
  - Change the reference of the array variable assigned int[] a={ 2,4,6}; int b[]={3,5}; a=b; // a[0]=? a[1]=? a[2]=?
- Multidimensional array
  - int[][] a=new int[3][5];
  - int b[][]={{1,2} ,{3,4,5},{6,7,8,9
  - Element reference a[row][col]
  - Multidimensional array.. a[array][row][col] a[0][3][0]=131



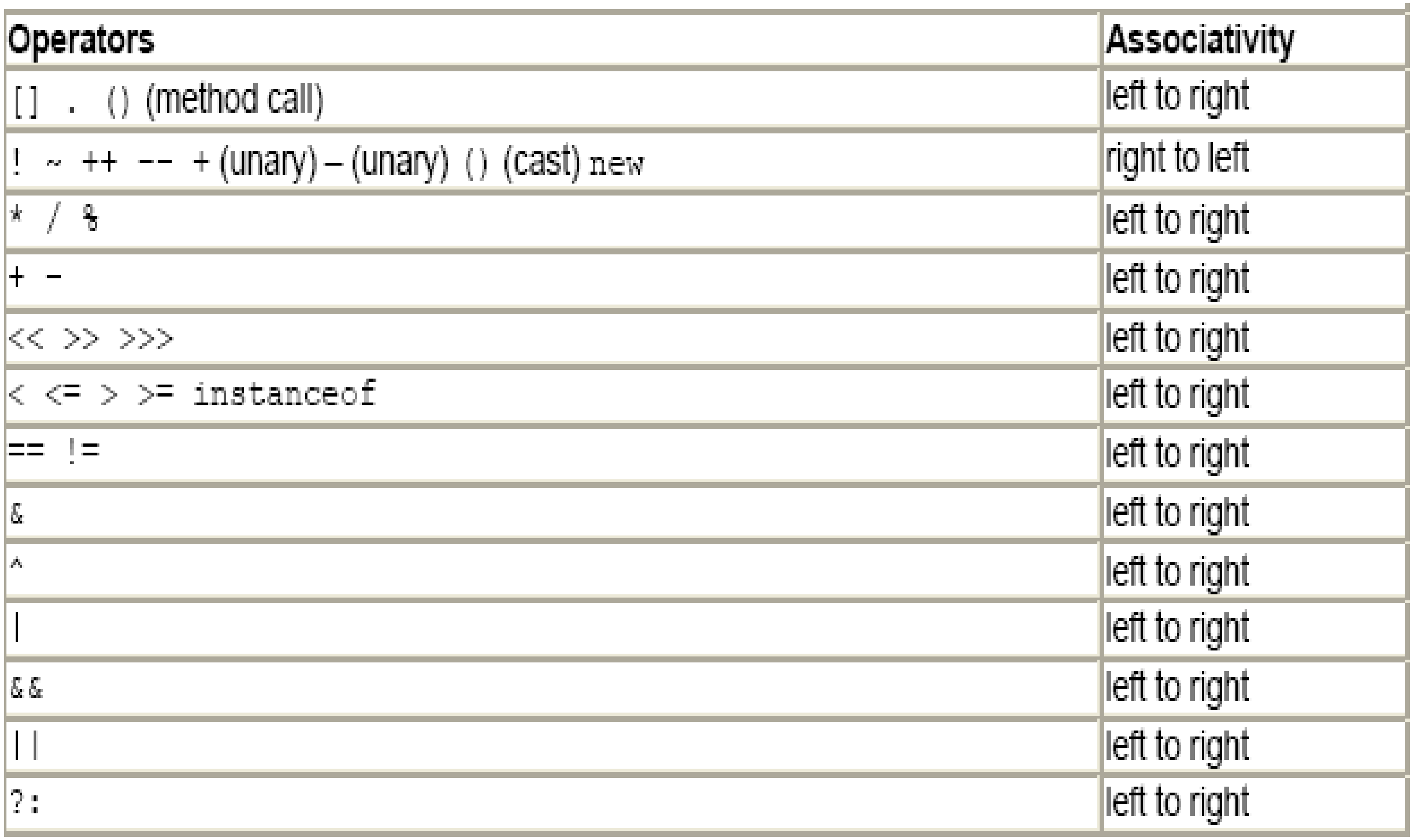
String class

- Represent string constants
  - Literal string using quotation mark: "abc"
  - + operator to concatenate 2 strings
  - All data types are automatically converted to string when joining + with 1 string
  - A string constant treated as a variable of the String class
    “abc”.length() returns 3
- Commonly used functions
  - length()
  - equals()
  - substring()
  - charAt()
  - indexOf()..
  - match(regular expr)
- String Pool: mechanism for storing string constants
- StringBuffer for string processing, StringTokenizer: string separator

Conditional operator

- Syntax:
  - (Expr1)? Expr2:Expr3
  - Ex: int a= (x>y)?10:5
- The expression expr1 must have a logic value
- Expr2 and Expr3 can take numeric, logical, or reference values, but must be of the same type
- The operand Expr2 & Expr3 can be a function call that returns the same type – (must not void!)

Order of precedence of operators



Operators	Associativity
[] . () (method call)	left to right
! ~ ++ -- + (unary) – (unary) () (cast) new	right to left
* / %	left to right
+ -	left to right
<< >> >>>	left to right
< <= > >= instanceof	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	left to right

Type casting

- Type casting is the process of transforming the data type of a value/variable/expression, following the following rules:
  - Do not convert from reference type to primitive type & vice versa.
  - Any value can be converted to String.
  - Array type can be converted to Object or String type.
  - The numeric type is automatically converted to the larger range numeric type when the two operands are of different types.
- Syntax:
  - Put (castingType) before the value/expression to be converted
  - Example:

    ```
    int a = (int)3.14; // cast real value to integer
    float b= 4; //automatically convert
    long c=b; // invalid, must be casted
    Vector d = (Vector)x; // x is cast to Vector
    ```

Type casting

$$\begin{tikzcd}
 & & \text{char} \arrow[d] & \\
\text{byte} \arrow[r] & \text{short} \arrow[r] & \text{int} \arrow[r] \arrow[d, gray] \arrow[dr] & \text{long} \arrow[dl, gray] \arrow[d, gray] \\
 & & \text{float} \arrow[r] & \text{double}
\end{tikzcd}$$

Assign Operator

- The = sign is used for assignment
  - Calculate the value of the expression on the right side, then assign it to the variable on the left side.
  - Must be of the same type or automatically convert a valid type
  - Returns the assigned value
- For example
  - double x,y,z;
    x=y=z=3.14; // assign x,y,z with 3.14 simultaneously
  - String cat='c'; // invalid
    String namSinh=""+1985; // valid
  - System.out.println(x=5); // print what?

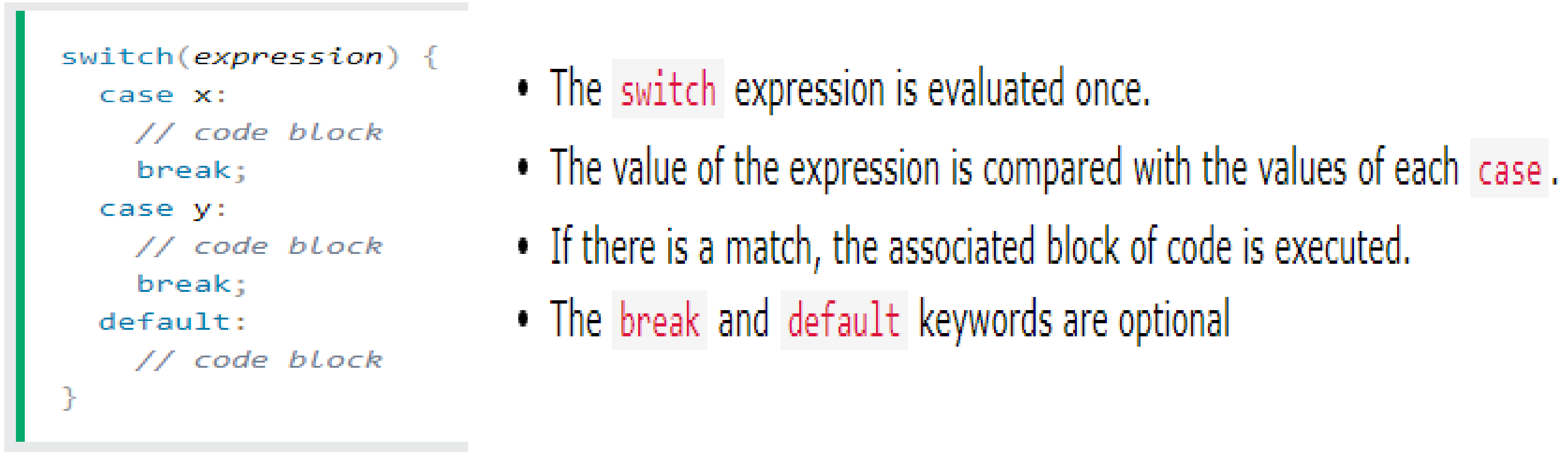
Conditional statement *if()/else*

Syntax

- if(condition) statement1; else statement2;
  - Statement1, statement2 can be a block of code, between the pair of bracket { }
  - There can be multiple if()/else nested
  - There may be no else . clause
- For example
  - byte month, daysOfMonth, year;
    if (month==2) if (year%4 == 0) daysOfMonth =29; else daysOfMonth =28;
    else if(month ==4 || month ==6 || month ==9 || month =11) daysOfMonth =30; else if(month <=12) daysOfMonth =31;

switch statement

- **switch**: to select one of many code blocks to be executed.



```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

- The `switch` expression is evaluated once.
- The value of the expression is compared with the values of each `case`.
- If there is a match, the associated block of code is executed.
- The `break` and `default` keywords are optional

- switch (month){

```
 case 1: case 3: case 5: case 7: case 8: case 10: case 12: days=31; break;
 case 4: case 6: case 9: case 11: days=30; break;
 case 2: days=(year%4==0)? 29:28; break;
 default: days= -1;
}
```

While loop

- while(condition){statements;}
  - Check condition prior to execute
  - Unknown number of iterations, ends when condition is false
  - Terminate loop any time with **break**
  - Example
    - while (x!=y){ if (x>y) x-=y; else y-=x; }
    - while (true) { } //an endless loop doing nothing
    - while(a<b && b<c) {
      a=a+b;
      b+=c;
      if (a==b+c) break; }

do.. while loop

- do {statements} while (condition);
  - Execute first, check condition thereafter.
  - Unknown number of iterations, ends when condition is false
  - Execution at least once
  - Terminate loop with break
- Example
  - do { if (x>y) x-=y; else y-=x; } while (x!=y);
  - do { x*=5; if (x>100) break; } while (true);

For loop

- for(bk1; bk2; bk3) { statements}
  - The loop allows a combination of tasks to initialize variables, check loop conditions, and count the number of iterations.
  - bk1 is usually a counter variable declaration & initialization
  - bk2 is for the testing loop condition
  - bk3 control counter increment
- Sequence:
  - step 1: bk1 is executed - usually declaration, variable initialization.
  - step 2: check bk2, if true then execute statements, otherwise end looping.
  - step 3: execute bk3, go back to step 2
- Example:
  - for(s=0,int i=1; i<=10; i++) s=s+i;
  - for(s=0, int i=1; i<=10;) s+=i++;

why variable s is not declared?

For each loop

- for(variable: arrayVar) { statements}
  - The loop allows exclusively to loop through elements in an **array**
- Example:

  -
    ```
    int[] a={ 3,5,7,9,11};
    int s=0; for(int x:a ) { s+=x; }
    ```

  -
    ```
    Student[] list= new Student[10];
    for(int i=0; i<list.length; i++) {
      list[i] = new Student();
      list[i].getData(); }
    for(Student s:list) System.out.println(s);
    ```

Function

- Performs a specify task and returns a value
- Function definition syntax

```
returnType methodName(arguments)
        { // body – must have a
          return value; // if return type is not void }
```

Input

out put

- Example: calculate the greatest common divisor of two integers
  - int uscln(int a, int b) {
    a=Math.abs(a); b=Math.abs(b);
    while (a!=b) { if (a>b) a-=b; else b-=a; }
    return a; }

Function call –argument passing

- When calling the function, pass arguments with the number, order and type corresponding to the declared arguments.
- **Pass-by-value: Changes to the passed argument inside the function do not change the value of the original passed argument.**
  - Example: The function calculates the greatest common divisor of two integers
    ```
    int uscln(int a, int b) {
    a=Math.abs(a); b=Math.abs(b);
    while (a!=b) if (a>b) a-=b; else b-=a; return a; }
    ```

  -
    ```
    int x=10, y=15;
    int us=uscln(x,y); //x passes to a, y passes to b, returns 5
    ```
    After the function call x is still 10 and y is 15
- There is no standalone function, the function declaration must be inside the class

main() function

- All executable Java applications must have a main() function, where the program begins execution.
- public static void main(String[] args)
  - public: keyword specifying access scope
  - *static: keyword indicates that all instance variables of this application share the same* main() *function* .
  - void: returns no value
  - String[] args :
    - argument of main
    - usually passed from the command window when running ct
    - args[0] is the first argument..

Chapter II exercises

- Write a program that prints your name, work office, and the greeting "Welcome to Java"
  - System.out.println(String x): prints the string x
  - Name the file with the same class name, notice Java – case-sensitive.
- Write a program IntArray
    - initializes an array of n integers, n is passed vie main() arguments
    - random initialization of array elements <=30,
    - sort array ascending,
    - print out the list of elements,
    - sum of element values,
    - count how many elements have even value
  - Integer.parseInt(s) converts string s to integer value
  - Math.random(): function returns a random double value betwwen 0 and 1
  - Arrays.sort(a): sort ascending array a
  - Put this line at the beginning of the program to declare the use of Arrays import java.util.Arrays;

Constructive Questions

- How should naming parttern to be used for easy of code reading?
- Count the number of variables used in a code block.
- Classifying variables in Java
- Why are there only 4 primitive data types: integer, real, character, logical?
- Why do you need data types? Is there any language that doesn't need to declare data types when declaring variables?
- How many data types are there in Java?
- Declare at least 4 way of variable declaration to represent data of your choice. Indicate which declaration should be used and explain.
- Compare **keyword** and **identifier**
- To convert GPA score to letter grades, which command structure should be used?
- Why is there no pointer type in Java like in C?

Constructive Questions

- Which iteration structure should be used to sum the following sequence S=1+1/2+1/3+…+1/n, given 1/n >e
- Give at least 3 example code block where the loop runs endlessly
- Declare a function that calculates the area of a triangle –
  List all available declarations and indicate the best one in your opinion
- Write a function to sum the elements of an integer array
- How to check a String meets the date format?
- Write a function to swap two integers
- Compare array in C and in Java
- Make a request to perform a certain task and convert it to function coding.
- Write a function that takes an argument of reference type. Make possible changes of this argument inside the function and display the changes after calling the function. What kind of change remains in the argument after calling the function?