Java Basics

- First Example
- Variables and Types
- Controls Statements

First Example

First Program

- Create a file named `Hello.java`
  - Java class files have `.java` extension
  - Note to naming convention
- Copy this lines to the file
  - Note: File name and class name should be the same.

```
/**
 * This is the first program.
 */
public class Hello {
    public static void main(String args[]) {
        // Prints a welcome message
        System.out.println("Hello");
    }
}
```

Compile and Run

- Using command line:
  - Compile
    - `javac Hello.java`
  - Run
    - `java Hello`
  - Compile and run together
    - `java Hello.java`
- Using IDE
  - From VS Code, just press Ctrl+F5

Anatomy

- Class declaration: `Hello`
- Main method: `main`
- Statements terminated by `;`
- Reserved words
- Comments
- Blocks

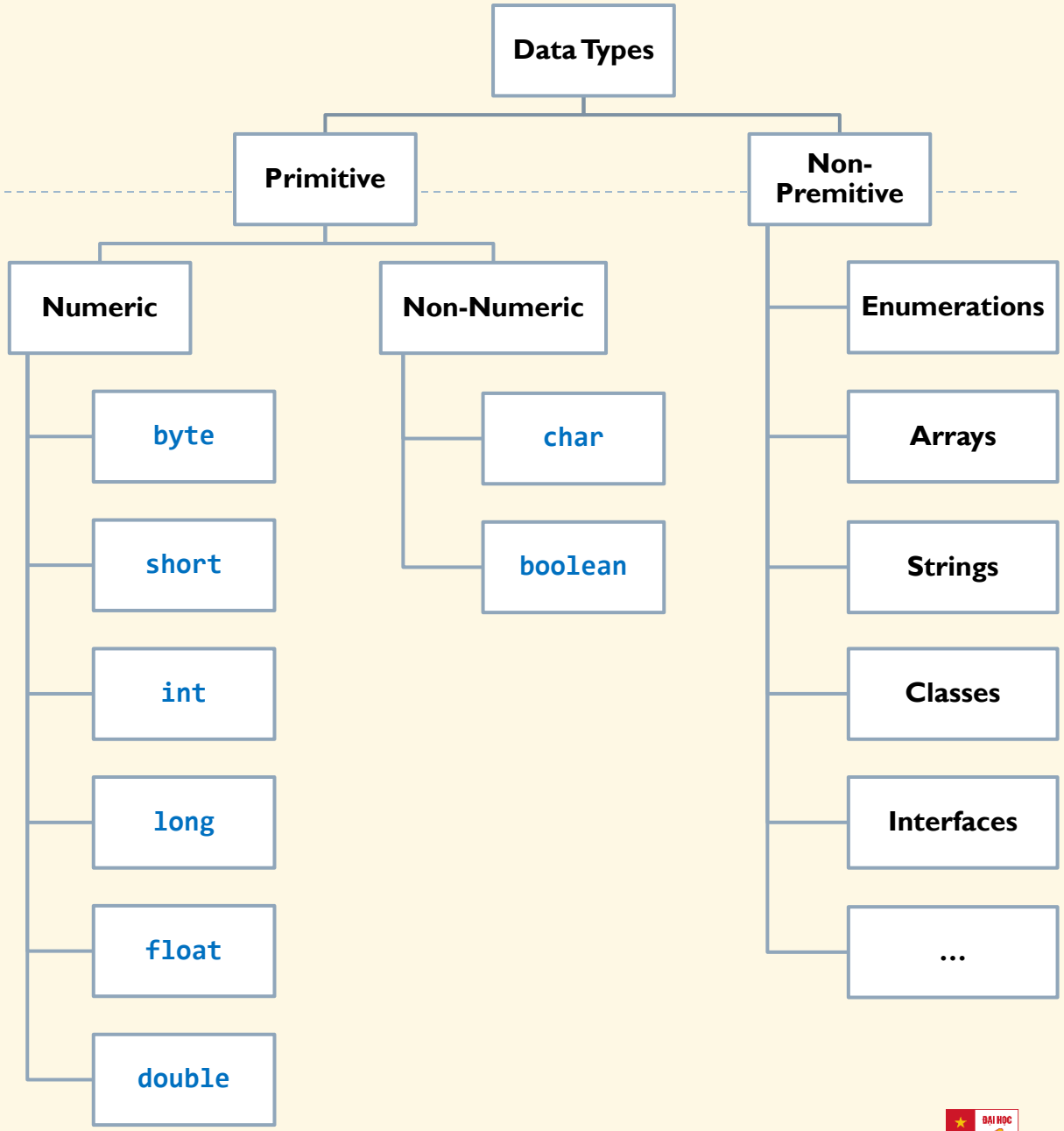
Naming Conventions

- Choose meaningful and descriptive names
- Class names: capitalize the first letter of each word in the name
  - Ex: `UserInfomationList`
- Method, variable names: capitalize the first letter of each word in the name, except the first one
  - Ex: `numberOfRecords`, `doSomethingFun()`

Variables and Types

Overview of Typing System

- Primitive types:
  - Types that are predefined and ready to use
  - Have no methods to call
  - Always have a value
  - Have fixed sizes
- Non-primitive types:
  - Mostly (except `String`) not defined by Java but by the programmer
  - Have methods to perform certain operations
  - Can be `null`



Primitive Types

Type Name	Size	Description
`byte`	1 byte	Stores whole numbers from -128 to 127
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes	Stores fractional numbers Sufficient for storing 6 to 7 decimal digits
`double`	8 bytes	Stores fractional numbers Sufficient for storing 15 decimal digits
`boolean`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single 16-bit character/letter values

- Attn:
  - No unsigned numeric types, all are signed
  - `char` size is 2 bytes → be careful when working with Unicode characters

Literals

- Literals are values that appears in the source code
  - ```
    byte b = -54;
    int i = 10;
    short s = 20;
    long l1 = -543_534_534_534_534l;
    long l2 = 0x5F_82_11_94_AE_B4l;
    long l3 = 0b11010010_01101001_10010100_10010010l;
    float f = 15.5f;
    double d1 = -53.22;
    double d2 = 3.14_159d;
    boolean bb = false;
    char c = '*';
    ```
  - Pay attention to the suffixes and the underscores in the numeric literals
    ```

Variables

- Variables can hold values
  - A variable occupies some memory for this purpose
  - The value of a variable can be changed at any time, via an *assignment*
- Variables need to be declared before using, with a *type*
  - This information guides the JVM how to interpret the stored value
  - The type of a variable can not be changed during its lifetime
- Example:

```
double r;           // declare an uninitialized single variable
long c = '#';       // declare a single variable with an initial value
int x, y = 3, z;    // declare multiple variables with a same type
r = 5.25;                   // assign a new value to the variable
double a = 3.14 * r * r;    // use a variable in an expression
```

Output

- Print a value
  - `System.out.print(value);`
  - `value` can be of any type of primitive types, or string
- Print a value then go to a new line
  - `System.out.println(value);`
- C-like printing
  - `System.out.printf(format, value1, value2,…);`
  - Use `%f`, `%d`, `%s`, `%c`, `%n`,… as placeholders (similar to C's `printf()`)
- Example

```
String name = "John", weather = "sunny";
double temp = 35.94;
System.out.print("Hello " + name + "!");
System.out.printf(" The temperature is %.1f degrees Celcius.", temp);
System.out.println(" It's a " + weather + " day.");
```

Reading User Input

- Read a whole line:
  - `String line = System.console().readLine();`
  - ```
    java.util.Scanner in = new java.util.Scanner(System.in);
    String line = in.nextLine();
    ```
- Read typed values:
  - ```
    java.util.Scanner in = new java.util.Scanner(System.in);
    int a = in.nextInt();
    double b = in.nextDouble();
    String c = in.next();
    String d = in.next("[a-zA-Z1-9]*");
    ```

Numeric Operators

- Arithmetic:
  - Binary: `a + b`, `a - b`, `a * b`, `a / b`, `a % b`
    - Both operands are integer → result is integer
    - At least one operand is floating point → result is floating point
  - Unary: `+a`, `-a`, `++a`, `a++`, `--a`, `a--`
- Logical: `a && b`, `a || b`, `!a`
- Bitwise: `a & b`, `a | b`, `a ^ b`, `~a`
- Comparison: `a == b`, `a != b`, `a > b`, `a < b`, `a >= b`, `a <= b`
- Ternary conditional: `<condition> ? <expression 1> : <expression 2>`
  - Example: `a > b ? a : b`

Assignment Operators

- Normal assignment: `a = <expression>`
- Compound assignments:
  - Example: `a += <expression>`
  - `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `^=`, `|=`, `<<=`, `>>=`, `>>>=`
- All assignments return a value, which enables the following statements:
  - `x = y = 10;`
  - `if ((c = readInput()) == 'Y' || c == 'y') { … }`

Type Casting

- Implicit casting
  - `double d = 3;   // type widening`
- Explicit casting
  - `int i = (int)3.0;   // type narrowing`
  - `int i = (int)3.9;   // fraction part is truncated`
- What is the result?
  - `int x = 3 / 10;`
  - `int x = (int)(3 / 10.0);`
  - `int x = (int)(3.0 / 10.0);`
  - `double x = 3 / 10;`

Constants

- A constant is a variable (yes, it's correct) which is read-only, using `final` keyword
- Constants are conventionally named with all uppercase letters

- Example:
  - ```
    final String SERVER_ADDRESS = "https://www.hust.edu.vn/";
    final int SERVER_PORT = 80;
    SERVER_PORT = 8080;      // Compile error
    ```
    ```

Enumerations

- An enumeration represents a group of constants

- Example:

```
enum Level {
     LOW, MEDIUM, HIGH
}

Level l = Level.LOW;
l = Level.HIGH;
```

Arrays

Declaration

- Arrays are used to store multiple values of a same type in a single variable, instead of declaring separate variables for each value
- Examples
  - ```
    int a[];                    // Declare an array variable ...
    a = new int[20];            // ... then allocate its elements
    ```
  - ```
    int a[] = new int[20];      // Combine both in one statement
    a = new int[50];            // Assign it with a new array
    ```
  - ```
    int b[] = new int[]{5, 7, 1, 2, 20}; // Create an array of 5 elements with values
    ```
  - ```
    int b[] = {5, 7, 1, 2, 20};          // Do the same thing
    System.out.println("Array length: " + d.length);
    ```
- Attention to array size:
  - Array length (number of elements) is fixed for a given array
  - If one needs the array to grow or shrink, assign the variable with another array with the desired length

Working with Array Elements

- Individual elements can be used as independent variables
- Each element is identified by its zero-based index
- Example:
  - ```
    long a[] = {10, 20, 30, 40, 50};
    a[3] = 100;
    a[2] = a[3] + a[0];
    a[5] = 200;      // Runtime error: Index out of bound
    ```
    ```

Multi-dimensional Arrays

- A multi-dimensional array is an array of arrays
- Example of two-dimensional arrays

```
int a[][] = new int[2][3];
int b[][] = new int[][]{{1, 2, 3, 4, 5}, {}, {6, 7}};
int c[][] = {{1, 2, 3, 4, 5}, {}, {6, 7}};

b[0][3] = 10;

b[1] = new int[]{8, 9, 10};
b[1] = {8, 9, 10};  // Compile error
```

Remarks on the Syntax

- Two different syntaxes for array declaration
  - `int a1[];`
    `int[] a2;`
  - `int b1[][];`
    `int[][] b2;`
- How about these declarations?
  - `int c1[], c2;`
  - `int[] d1, d2;`
  - `int[] e1, e2[];`

Strings

Strings

- A string is a sequence of characters – just like an array of characters
- Declaration
  - `String s1 = "This is a string";`
  - `String s2 = new String("D:\\folder\\filename.ext");`
  - ```
    char ca[] = {'H', 'e', 'l', 'l', 'o'};
    String s3 = new String (ca);
    ```
- A string once created is immutable (its value cannot be modified) → updates made to a string will lead to a new string object
  - ```
    char c1 = s1.charAt(3);   // Get a char by its zero-based index
    char c2 = s1.charAt(20);  // Runtime error: Index out of bound
    s1.charAt(6) = '$';       // Compile error
    s1 = "This i$ a string";  // Ok!
    ```

Comparison

- By value:
  - ```
    String s1 = "cartoon", s2 = "Cartus";
    boolean r1 = s1 == s2;                  // false
    boolean r2 = s1 != s2;                  // true
    boolean r3 = s1.equals(s2);             // false
    int r4 = s1.compareTo(s2);              // > 0
    int r5 = s1.compareToIgnoreCase(s2);    // < 0
    ```
- By reference:
  - ```
    String a = "cartoon",
        b1 = "cartoon",
        b2 = "car" + "toon";
    String b3 = "car";
    b3 += "toon";

    boolean r1 = a == b1;   // true
    boolean r2 = a == b2;   // ?
    boolean r3 = a == b3;   // ?
    ```

Other Operations

Method	Return type	Description
`length()`	`int`	Returns string length
`substring(int start, [int end])`	`String`	Returns substring for given begin index and end index
`contains(String s)`	`boolean`	Checks whether the string contains the given value
`equals(String s)`	`boolean`	Checks the equality of string with the given value
`isEmpty()`	`boolean`	Checks if the string is empty
`concat(String s)`	`String`	Concatenates the specified string
`replace(char old, char new)`	`String`	Replaces all occurrences of the given char value
`split(String regex)`	`String[]`	Returns a split string matching the given regular expression
`indexOf(int c, [int from])` `indexOf(String s, [int from])`	`int`	Returns the index position for the given character/substring
`toLowerCase()` `toUpperCase()`	`String`	Returns the given string converted in lowercase/uppercase
`trim()`	`String`	Removes beginning and ending spaces of the given string
`format(String fmt, Object... args)`	`String`	(static) Returns a formatted string
`join(String dem, String... elms)`	`String`	(static) Returns a joined string

More Fashions to Create Strings

- Multi-line string literals
  - ```
    String name = "John";
    int yob = 2005;
    String s = name + " was born in " + yob + ". He is " +
      (Calendar.getInstance().get(Calendar.YEAR) - yob) +
      " years old.";
    ```
- Using `format()`
  - ```
    String s = String.format("%s was born in %d. He is %d years old.",
    name, yob, (Calendar.getInstance().get(Calendar.YEAR) - yob));
    ```
- Using `join()`
  - ```
    String[] cities = { "Hanoi", "Hue", "Danang" };
    String s = String.join(", ", cities);
    ```
    ```

Mutable Strings

- Use StringBuilder class for mutable strings
- Example:

```
StringBuilder sb = new StringBuilder("car");
sb.append("toon");
sb.setCharAt(4, '0');

System.out.println(sb);    // cart0on

String s = sb.toString();  // convert to String
System.out.println(s);     // cart0on
```

Control Statements

Introduction

- Statements are executed from top to bottom by default
- Control statements can be used to change the order of execution
- Two groups of control statements
  - Decision making
    - `if` statement
    - `switch` statement
  - Looping
    - `do-while` loop
    - `while` loop
    - `for` loop
    - `for-each` loop

`if` Statement

- Decision making statement
- Syntax:
  - `if (<condition>) <statement>`
    `[else <statement>]`
- Examples:
  - ```
    if (x != 0.)
        System.out.println("Inverse = " + (1 / x));
    else System.out.println("Irreversible");
    ```
  - ```
    if (score > current_record)
        NewRecord(score);
    ```

Nested `if` Statement

- Nested `if` statement is usually used to check multiple conditions
- Example:

```
if (mark >= 8.)
    System.out.println("Good grade");
else if (mark >= 7.)
    System.out.println("Fairly good");
else if (mark >= 5.)
    System.out.println("Passed");
else
    System.out.println("Not passed");
```

`switch` Statement

- Multi-choice decision making statement
- Syntax:
  - `switch (<expression>) {`

    `case <value1>: <statements>`

    `case <value2>: <statements>`

    `...`

    `[default: <statements>]`

    `}`
- Execute different statements depending on which case the expression value corresponds to
- `default` case is executed when the expression value corresponds to none of the above cases
- The execution falls to the next case after finishing one → use `break` to terminate if necessary
- Can only be used with integer expressions (`char`, `int`, `enum`,…), and the listed values must be constant

`switch` Statement *(cont.)*

```
switch (x) {
case 0:
    System.out.println("x is 0");
    break;
case 1:
    System.out.println("x is 1");
    break;
default:
    System.out.println("x is something else");
}
```

```
switch (day) {
case SATURDAY:
case SUNDAY:
    System.out.println("Weekend");
    break;
default:
    System.out.println("Working day");
}
```

do and while Loops

- Used to repeatedly execute tasks
- Syntax:
  - `while(<condition>) <statement>`

    Check the condition *before* the execution of the statement in each iteration
  - `do <statement> while(<condition>);`

    Check the condition *after* the execution of the statement in each iteration
- Examples:
  - ```
    x = 0;
    do {
        System.out.println(x + " ");
        x++;
    } while (x < 10);
    ```
  - ```
    x = 0;
    while (x < 10) {
        System.out.println(x + " ");
        x++;
    }
    ```

`for` Loop

- Compact loop statement with initialization, condition and increment/decrement in a single line
- Syntax:
  - `for (<initialization>; <condition>; <increment/decrement>)`

    `    <statement>`
  - Each expression can be ignored when unused
  - Condition is checked before each iteration
- Examples:
  - ```
    for (int x = 0; x < 10; x++)
        System.out.println(x + " ");
    ```
  - ```
    for (int i = 0; i < N; i++)
        for (int j = 0; j < M; j++)
            a[i][j] = i + j;
    ```

`break` and `continue` statements

- `break`: used to terminate a whole loop regardless the condition
  - ```
    for (int x = 1; x <= 10; x++) {
        if (x == 8) break;
        System.out.println(x + " ");
    }
    ```

- `continue`: used to terminate one iteration, and execute the next one
  - ```
    for (int x = 1; x <= 10; x++) {
        if (x == 8) continue;
        System.out.println(x + " ");
    }
    ```

for-each Loop

- Looping through a one-dimensional array
  - Normal for loop
    - `for (int i = 0; i < a.length; i++) { ... }`
    - `for (int i = a.length - 1; i >= 0; i++) { ... }`
  - for-each loop
    - `for (long ai : a) { ... }`

- Looping through a two-dimensional array
  - ```
    for (int i = 0; i < b.length; i++) {
        for (int j = 0; j < b[i].length; j++) { … }
    }
    ```
  - ```
    for (int bi[] : b) {
        for (int bii : bi) { … }
    }
    ```

Exercises

Write programs to:

1. Enter coefficients a, b, c of an 2^{nd} order equation and solve it
2. Calculate sum of inverses of even numbers from 2 to 100: 1/2+1/4+1/6+…+1/100
3. Enter a chain of numbers and calculate the average value
4. Print a menu and read user's choice by two ways: (1) using decision making statements, (2) using array
5. Calculate the traversed distance of a falling object at moments: t = 1, 2, 3,…, 20s
6. Same as Prob. 5, but enter the initial height and calculate only until the object reaches the ground
7. Read an integer n, then print all natural numbers of n digits (for example, 100000 to 999999 if n is 6)