Chapter 1

- bytecode → machine code
- compiling = batch process
- OOP paradigm

ERRORS: ① syntax, ② runtime, ③ logic

* writing → compiling → debugging → executing

* syntax error: (Java tells exactly the error)
  - ↳ cannot run cus, X compiled
  - ↳ if problem = incorrectly nested braces, actual error can be beginning of nested block
  - ↳ eg. undeclared variables used, missing ;, wrong indentation etc.

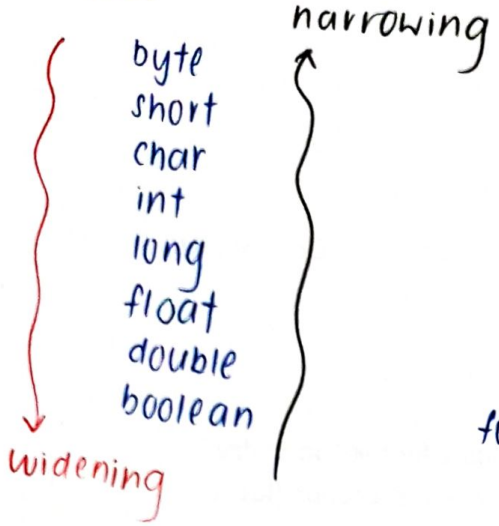
* Logic error: "bug" (Java don't provide any information on the error)
  - ↳ program runs and compiles successfully but does not give expected output
  - ↳ how to track:
    - ↳ debugger
    - ↳ use print statements

* Runtime error: (Java discover but need locate error yourselves)
  - ↳ can also dun to insufficient memory
  - ↳ result in program crashing
    - ↳ continuously use more RAM as program runs
  - ↳ Java shows where the error is expected but is unable to correct
    - ↳ eg. null pointer exception
  - ↳ will get error message

Chapter 2

- datatype variableName = datavalue;
- initialised with sensible value automatically:
  - boolean = false
  - integers = 0
- variable naming:
  - no space
  - start with lower case (unless class name)
  - don't start with digits
  - meaningful names
- print statement: System.out.println( )
- constant declaration statements: final datatype CON_VARIABLE = datavalue
* syntactically correct: double no1=10, no2=3.6, no3;
- float store data approximately
* All statements terminated with semi-colons (;)

chapter 3



operators : ① arithmetic
② comparison
③ logical
④ assignment
⑤ bitwise

found in pdf "WEEK-3 NOTES"

functions : → arithmetic calculation
→ string manipulation
→ assign value to variables
→ compare data values

scope of variables :

- declared in class (outside any methods)
  ↳ scope : entire class (include method)
- declared in method
  ↳ scope : only in method the variable is declared
- declared in block
  ↳ scope : only in block the variable is declared

control flow statements :

- fix sequence of statements that should or shouldn't be executed repeatedly or not depending on conditions associated
- 3 types: ① decision making statements (if, switch)
  ② loop statement (do-while, while, for, for-each)
  ③ jump statement (break, continue)

execution depends on conditions → ② repeatedly

③ → transfer control of program to specific statements

- precedence order : 2 operators sharing operands, operator with higher precedence will be evaluated first
  ↳ BODMAS

Type conversion :

① implicit type conversion
↳ operation change resulting data type based on operand's data type
↳ eg. int a = 10;
b = a + 10.0;

② explicit type conversion
↳ forcibly convert one data type to another
↳ eg. b = (int) 10.25;

Chapter 4

* execute = invok = call = run

- syntax for outlining a method:

start with lower case

if add, need datatype

AccessModifier ReturnType methodname (optional parameter) { }

- eg.
```
public int modulus (){
    return (10%3);
}
```

```
public int product (int num_1, int num_2){
    return ( num_1 * num_2);
}
```

① ② ③

```
public void sum ( int numOne, int numTWO) {
    int sum = numOne + numTWO;
}
```

① method header
② method signature
③ method body

** method overloading (form of polymorphism)
↳ same name, different parameters
↳ in same class

method overriding
↳ same signature, same name & parameters

** Access modifiers (lower case)

↳ defines visibility of a section of code
- instance variables
- constructors
- methods

↳ 3 types : ① public (seen by all)
② private (seen by system only)
③ protected (seen by inherited class)

** Return types + method types

↳ specifies data type returned to calling object, method / user
↳ void : does not return anything
↳ return statement at last line of code
↳ stop execution at return statement & return value
* after return statement, code don't get executed

** Local variables + parameters

↳ local variables = variables defined within the body of method
↳ accessible within the method only
↳ formal parameters = variable defined with specific data type that acts as a placeholder in method header
↳ accepts any values given when method is called but have to match data type
↳ actual arguments = values passed when method is executed
↳ substitute formal parameters

*** public static void

↳ public = accessible by everyone
↳ static = method associated with class → not specific instance of class (no class object)
↳ void = return nothing

Chapter 5

data storage memory
- stack
  - 64-bits
  - primitive data values
  - ✓ rapid, efficiently
- heap
  - primitive + non-primitive
  - object data size = large
  - slower, ↓ efficient

** Arrays :
- ↳ store multiple variables of same type using a single variable
- ↳ syntax : ① elemType [] arrayName ;
  arrayName = new elemType [length] ;
  ② elemType [] arrayName = new elemType [length] ;
- ↳ cannot resized once sized

** null = no address
- ↳ if run method with reference variable with value null, null pointer exception error

** Array Lists : (a resizable array) → only need max size
- ↳ must import array list library : import java.util.ArrayList ;
- ↳ method + class + import :

```
import java.util.ArrayList;
public class Week5Class {
    public static void main(String[] args) {
        ArrayList<Integer> myList = new ArrayList<>();
        myList.add(3);  ⇒ add elements
        myList.add(0,4); ⇒ add 4 at index 0
        myList.set(0,7); ⇒ replace existing elements
        System.out.println(myList); → print ArrayList
    }
}
```

Chapter 6

Select control structure

↳ ✓ conditions (usually boolean)

↳ allow statement blocks to be executed or not depending on conditions in code

↳ ① if-trap
② if-else
③ if-else if-else

* repetition control structures
↳ repeatedly execute statement blocks

** If-trap :

↳ statement block included or excluded from statements depending on condition

↳ syntax : `if (condition) { }` — { } = included statements blocks; (condition) = parentheses

↳ condition → TRUE = execute statement blocks
→ FALSE = don't execute statement blocks

** If-else :

↳ one OR two alternate statements blocks can be executed, depending on condition

↳ syntax :

```
if (condition) {
  (statements)
}                  ← if condition = TRUE
else {
}
  (statements)     ← if condition = FALSE
```

** If-else if-else : (like : if-elif-else in python)

↳ multiple conditions = multi-way selection

↳ syntax :

```
if (condition) {
}
  (statements)
else if ( condition 2) {
  (statements)
}
else { (statements) }
```

evaluated from top to bottom
(once a condition is TRUE, proceed to next statement)

** Switch statement (switch-case)

↳ a number of possible execution paths

↳ syntax = non-standard

↳ eg. days Of The Week :

* may be wrong (I wrote without reference TAT)

```
import java.util.Scanner;

public class Week6Applied {
    public static void main (String [] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.println (" Enter an integer ");
        int dayInt = in.nextInt();
        System.out.println(W6.daysOfTheWeek( dayInt));
    }

    public String daysOfTheWeek (int dayInt) {
        String dayName;
        switch (dayInt) {
            case 1:
                dayName = " monday ";
                break;
            case 2:
                dayName = " Tuesday ";
                break;
            case 3:
                dayName = " Wednesday ";
                break;
            case 4:
                dayName = " Thursday ";
                break;
            case 5:
                dayName = " Friday ";
                break;
            case 6:
                dayName = " saturday ";
                break;
            case 7:
                dayName = " sunday ";
                break;
            default:
                dayName = " Invalid day ";
                break;
        }
        system.out.println ( dayName );
    }
}
```

Chapter 7

- repetition control structure
  - ↳ statement blocks repeatedly executed as long as condition is met
  - ↳ ① while loop
    - ② for loop
    - ③ do-while loop

** while loop

↳ syntax : while (condition) { }

↳ eg.
```
count = 0
while (count < 10) {
    sum = a + b;
    count ++;
}
```
← increment by 1

↳ command runs until condition is reached
↳ number of repetition unknown

** for loop

↳ syntax : for (initialisation ; loop condition ; decrement / increment) { }

↳ eg.
```
for (int i = 1 , i <= 10; i++) {
    sum = a + b;
}
```

↳ used to obtained a certain result
↳ number of repetition is known

** Type of loops : → FOR loops
① counter controlled loops (repetition known) ← counter initialised as 0 or 1 b4 loop
② sentinel controlled loops (repetition unknown)
③ value controlled loops (repetition unknown)

WHILE loops

specific value occuring
eg. while (value != smtg) { }
↳ test occurence of a special value

calculate value } eg. user input value
eg. while (value is not a terminating value) { }
↳ test occurance of a calculated value

** Do-while loop

↳ post-test loop

↳ statement block executed at least once

↳ test perform after first repetition

↳ syntax:

```
int count = 1; }"counter"
do {
    system.out.println ("count is: " + count);
    count ++;
}
while ( count < 11);
```

↳ expression

* Enhanced For Loop:

↳ traversing array / collections (eg. Arraylist)

↳ eg. sum of all numbers in an array of integers

↳ syntax:

```
int [] num = {3, 4, 5, 6, 7};
int sum = 0;

for ( int nums: num) {
    sum += nums;
}

system.out.println ( "sum =" + sum);
```