Week 1

Object Orientation

→ A car has a speed that is affected by applying car brake.
[object] [data item] [behaviour]

→ Bank Account [concept]

variables - storage location to store data item

methods - implement a behaviour of the object by changing it's data

* abstraction - process of selecting essential data & behaviour of an obj for a particular application context

state - current value of an obj variables.

→ obj method result in changes of object state.

Classes

→ create instances (object)

→ process of creating class instances = instantiation.

instance variables - identical but independent set for each class instance

* Encapsulation / data hiding

→ bundle obj data (variables) & obj method into a class, making them inaccessible outside of the class

→ visibility - ① variables - data is accessible by code outside of the class to set (change) & get its current value (state)

- ② methods - it can be executed (invoked) by code in other classes.

→ modularisation - bundle code into well designed methods

↳ code a task only once & invoke it several times.

↳ to avoid duplication of code

Why use OO

→ reusability from application to application.

→ by using inheritance.

→ easier maintenance, reduce error.

→ mirror reality

→ confining programmer focus

→ refactoring

NO: _____ DATE: _____

compiler - translate entire code at once, batch process, arise error at the end

interpreter - interpret instruction one at a time, interactive process, immediately arise error

write high-lvl code → compile them to machine code → execute machine code

[debussing take place]

Quiz: bytecode = intermediary code produced after compilation.

Java is a compiled & interpreted language.

A Java virtual translates bytecode to machine code

week 2

Variable Declaration Statement

dataType variableName [= dataValue] [, variableName [= dataValue]] ... ;

Assignment statement

variableName = expression;

evaluate RHS expression & store to LHS variableName

capital letter for constant

* Constant declaration statement

final dataType [CONST_NAME [= dataValue] [, CONST_NAME [= dataValue]] ... ;

DataType

→ ① value type

↳ primitive data type

- byte, short, char, int, long, float, double, boolean

↳ data type conversion

- ⓐ widening conversion
  - no info loss, but loss of accuracy.
  - smaller value range → large value range
  - eg: int 2 → double 2.0
- ⓑ narrowing conversion
  - info loss
  - large → smaller value range
  - eg: double 2.0 → int ??
- assignment conversion, promotion (multiple mixed type operands), casting.

* data stored in value type variable cannot be changed

* naming convention: camelCase

```
float money;
int dollars = 3;
money = dollars;
=> money = 3.0
```

```
double sum = 17.0, result;
int count = 10;
result = sum / count;
=> result = 1.7
```

=> a copy of count is promoted to double

```
double money = 5.67;
int dollar = 3;
dollar = (int) money;
=> dollar = 5 [truncation]
```

Quiz: Java Variables & Constants cannot have name begin with num.

Java constant that has been declared & initialised can be used to initialise a Java variable

" variable " " " " but not yet initialised cannot be used to initialise a Java Constant

```
int x, y, z;
x = 3;
y = 5;
z = 7;
z = x = y;
System.out.println(x + " " + y + " " + z);
```

=> 5 5 5

```
int no = 10;
boolean flag = true;
double rate = 1.5;
String unit = "1051";
System.out.println(no + unit + rate);
System.out.println(no + flag + unit + rate);
```

→ 10105111.5 [all will be convert to String]

→ error [boolean & string cannot Concatenation]

Casting betw integer and double

```
int inum = 10;
double dnum = inum;
System.out.println(dnum); // 10.0
```

[no need convert by typecasting bcoz int → double is done automatically]

```
double dnum2 = 20.99;
int inum2 = (int) dnum2;
System.out.println(num2); // 20
```

[convert by typecasting, double → int is narrowing conversion, cannot be done automatically]

* → int i1 i2;
  i1 = 1; i2 = 2;
  → int i1 = 1 i2 = 2;
  → int i1 = 1; int i2 = 2;

equality == relational < <= > >=

inequality !=

week 3: Expressions

- perform arithmetic calculations
- perform String manipulation
- assign values to variables
- compare data value [return boolean true / false]

pre increment / decrement

eg: int i1, i2;
i1 = 1; i2 = 2;
System.out.println(i1 + ++i2);
=> 4

=> i1 remain to be 1,
i2 become 3

post incre / decrement

int i3, i4;
i3 = 3, i4 = 4;
System.out.println(i3 + i4--);
=> 7

=> i3 remain to be 3,
i4 decrement to 3

NO: _____ DATE: _____

Take in user input

```
import java.util.Scanner;
public class Week4Class {
    public static void main(String[] args) {
        String s1 = "dog", s2; ① declare variable
        Scanner scanner = new Scanner(System.in); ② make Scanner
        System.out.println("Enter a word: "); ③ prompt user input
        s2 = scanner.nextLine(); ④ for user to enter input
        System.out.println("s1 == s2: " + (s1 == s2));
    }
}
```

depends on dataType
→ int = scanner.nextInt()
→ String = scanner.nextLine()

Operators

numeric [Equality / Inequality]
→ num1 == num2

String [relational
→ str1.equals(str2)

Logical Operators

		eg: b1	!b1
Not	!	true	false
And	&&	false	true
Or	||		

\|\| < && < !

Short Circuit

→ && (AND)
↳ if 1st statement is false, the 2nd statement won't be executed
↳ output straight away be false

→ || (OR)
↳ if 1st statement == true.
straight away true

Logicals < Equality / Inequality < relationals < arithmetic

```
boolean bool = false;
System.out.println(bool); // false
System.out.println(!bool); // true
if (!bool) { // as long as this line evaluate to be true, the statement block will be executed
    System.out.println("abc"); // abc
}
```

NO: _____ DATE: _____

Primitive Data Type

→ fundamental data type that's predefined as part of the language

① integer

→ byte, short, int, long

* ' ' for char
" " for string

② real num

→ float, double

③ other

→ char, boolean

Declaration: `int num1;`

Initialisation: `num1 = 12;`

`int num1 = 12;` Declaration + Initialisation

`num1 = 13;` Assignment

Relational Operators

```
int age = 20;
age > 20   // false
age >= 20  // true
age < 20   // false
age <= 20  // true
age == 20  // true
age != 20  // false
boolean isStudent = false;
isStudent  // false
!isStudent // true
```

Arithmetic Operators

+, -, *, /, %

```
eg: int b = 8
    int d = b % 5 // 1
```

```
eg: int x = 2;
    int y = 4;
    int z = 3;
    int w = x++ - --y + 3;
            2 - 3 + 3
          = 2
```

→ x then increment in memory to 3
→ y become 3

```
eg: int x = 10;
int w = x++ + x++ + x++ - (--x - --x - --x)
      = 10 + 11 + 12 - (12 - 11 - 10)
      = 33 - 12 + 11 + 10
      = 42
```

```
eg: int a = 12; // 12
    char b = "x" // 88
    a + b = 100
```

```
eg: int x = 1;
    int w = ++x + ++x + ++x
            2 + 3 + 4
          = 9
```

→ x now because 5

NO: _____ DATE: _____

Quiz:

```
int z = 25;
int y = (z % 5) % 2;
boolean test = false;
test == ((z % 2) % 5 == y)
```

25 % 5 = 0 ; 0 % 2 = 0 ; y = 0

25 % 2 = 1 ; 1 % 5 = 1 ; 1 == 0 => false

false == false => true

=> true #1

```
boolean test = false;
(!(!(!test))) == test
```

true ← false ← true ; false

true == false => false

```
int a = 5;
int b = 3;
int c = 2;
int d = 4;
int e = 1;
a * b + c % d++ - e
```

(5) (3) (2) (4) (1)

then become 5

① 2 % 4 => 2

② 5 * 3 = 15

③ 15 + 2 - 1 = 16 #1

Explicit type conversion

eg: b = (int) 10.25

=> 10

eg: b = (char) 65;

=> 'A'

eg: a = 10;

b = 20;

c = (int)((double)((double)a * (int)b++ / 2 + 1) / 3)

10.0 ; 20, then become 21

10.0 * 20 / 2

= 200.0 / 2

= 100.0

(double)(100.0 + 1) = 101.0

(int)(101.0 ÷ 3) = (int)(33.6667)

= 33 #1

eg: 10 - 3 * 2 / 1 = 4