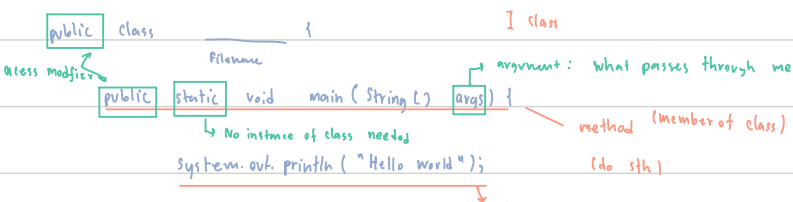
Hello world



// make comment

Variable declaration / initilization

int x ; declaration

x = 5 ; initilization

final int Y = 5 constant

↳ constants use capital letters

Import (Scanner)

(at top) import java.util.Scanner

(in methods) Scanner scanner = new Scanner (System.in) ;

create new object : Type Identifier = new type ( ) ;

constructor

Datatypes

object Integer x = 5 ;

object

primitive types (don't need new) int x = 5 ;

Priminative data types

boolean // true false

byte // "C" 5 store as binary (ascii)

char // "z" — " — (unicode)

short // small number 16 bit

int // number 32 bit

long // 50L number 64 bit

↳ let program knows it is long value

float // decimal 20.5f (32-bit)

↳ let program know it is float

make it smaller

double // — " — 20.5 (64-bit)

Input

```
Scanner scanner = new Scanner(System.in);

String x = scanner.nextLine();

int y = scanner.nextInt();

BigDecimal z = scanner.nextBigDecimal();
```

* need to import

↳ more precise

```
int x = scanner.nextInt(8);
```

radix

base

```
int x = (int) scanner.nextDouble();
```

↳ reassign type

⋮

ex.

```
int x = 5          // x = 6
int a = x++        // a = 5
```

left to right

```
int x = 5          // x = 6
int a = ++x        // a = 6
```

Numeric method

static

```
int x = 10
int y = 20

Integer.max(x,y)
```

↓ method

```
Integer.compare(x,y)
```

x < y // -1

x > y // 1

Numeric expression

```
double x = 5/2          // 2.0
```

↓

```
double x = 5.0/2        // 2.5

double x = (double) 5/2 // 2.5
```

```
Integer a = Integer.valueOf(____);   // object
            Double
int b = Integer.parseInt(____);      // int
        Double
```

(string)

Operators

precedence (order evaluated)

(last) ↑ (first)

+	add	/	division
-	minus	%	modulus
*	multiplication		

x++ add x by 1

x+= ____ num

x-- minus x by 1

x-= ____ num

x ____ = ____ num

↓ any operator

String Class

String x = "hello" // object (instance of class)

\n new line \t tabs

x + "____" concatonate string

System.out.println (String.format ("____ %s", name)

for complex string

name .length( ) # return length of name
var

String Methods

x # string

x. charAt (__)
↳ location

x. charAt (x.length( ) - 1) # last char

x. contains ("____") # True False
String

x. indexOf ("__", ____) # int
↓
where to start looking for

x. lastIndexOf ("__")
search from last → start

x. toUpperCase (____) # uppercase

x. strip( )
leading ( ) # in the begining

x. repeat (__)
↳ times # int

____ . equals (____) # T/F
String1 String2

Class

public class User {

// members — method and properties

public String name = "____"

}

in main of the other file

User user = new User( )

user.name = "____" # setting property

in User

public String Naming {

return ____ + ____ ;

}

in Main

user. Naming ( ) # getting method

String comparison

String1 == String2

* Work with primitives not object

↓

String1 . equals (String2)

If statements

if (______) {
expression

____

return; # end main method
}

if ( ) {

______

} else if ( ) {

} ⋮

else {

}

Comparison & logical operators

== equality

!= not equality

< > <= >=

&& and

|| or

! not

Switch statements

switch (______) {
var

* must have break for each case

case " ___ ":

______________ ;
break;

case " ___ ":

______________ ;
break;

default:

break;

→ Multiple case

case " ___ ":
case " ___ ":
case " ___ ":

______________ ← do this

break;

Ternary Operator

type var = ______ ? true : false ;

condition, value, value

Single line if statement

if (______) true ;

condition, true case

While loops

initialization
comparison
update

int i = 0;
while (i < ___) {
con

______

i++;

}

Do while loops

do
{

______

} while (___) ;
con

** variable in do-while loops can't be declare inside

↓

declare var → do-while

For loops

```
for ( int i = 0; i < ___ , i++ ) {
        ____________ ;
}
```
(comparators: <, int)

if and loops can be nested

to create more complex condition

Break

```
if ( ___ ) {      (con)
    break;        # stop the continuous loops
}
```

Continue

exit that interation / skip only that case
↳ (only that case)

Arrays

int [ ] grades = new int [10] ; / int grades [ ]
(type, name, size; c method)

grade.length ; # size

grade [a] # get the value position a / assign value at a^th

Array (continue)

int[ ] grades = { 1, 2, 3, 4, 5, 6, 7, 8 } ; # assign value to each array

grades[1] = 900 ;

Array to string

* Array is not printable *

Array.toString(___) ; # convert arrays to string (variable)

Array.deepToString(___) ; # Nested arrays (var)

Loops and arrays

```
for (                    ) ;
        grades [i] = 5 ;
```
(var, value)

Sorting Arrays

Arrays.sort (___) (array var)

Arrays.parrallelSort (___) # large arrays (array var)

Arrays Methods

Arrays.equals(____, ____) # Compare arrays equality
(array1, array2)

Arrays.fill(____, ____) # fill array of ____ value
(array, value)

List<String> ____ = Array.asList(____) # Array → List
(var) (array)

Initilize List (print list)

____.toArray(); # list → array
↓
print in array ways

For each loop

for(int ____ : ____){
(var) (List)
↕ same var
____;
(var)
}

2D arrays

int[][] ____ = new int[__][__]
(name) (row size) (col size)

= {
{1,2,3,4
{4,5,6,7
} # easier to see

grades[0][1] # grab / assign value

Nested List

List<List<Integer>> ____ = new el
(name)

ArrayList<List<Integer>>();

ArrayList

ArrayList<Integer> ____ = new ArrayList<Integer>();
(type) (name) (class) (type)
Method

____.add(__);
(name) (value)

____.get(__);
(name) (index)

____.size(); # size of List

____.set(__, __);
(name) (index) (value)

____.contain(__); # T/F
(value)

____.isEmpty();

____.remove(__);
(value)

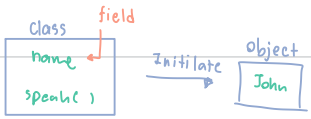
____.clear(); # Clear ArrayList

Sorting List

Collections.sort(__);
(list)

Collections.reverse(__);
(list)

OOP



field variable created outside method

(be able to acess locally)

ex.

```
Person me = new Person( );
me.name = "John";
```

```
public string speak( ) {
    return "_____"
}
```

Parameters

method

ex. public void (____ name) {
type

_____

}

Return Statement

ex. public String _____(_) {
name

same type

return "Hello"

}

Public / Private

Private made variable only accessible in class / through object

ex. private string name;

Encapsulation (Getter, Setter)

Getter Method * public only

ex. public String _____ {
type getter name

return "_____" ← get public object (declared publicly)

}

Refference Construtor

ex. User user = new User( )
class

Setter Method * Work with private

ex. public void _____ (type String variable fn) {
setter name parameter

firstname = fn; ← * assign object to parameter

}

Method

public void ____( ) {
name

use the variable declared from constructor

_____;

}

Custom Type List

ex. List < User > users = new ArrayList < User > ;

- List < User > users: class to store object in
- new ArrayList < User >: store object

Custom type Arguments

in same class

ex. public void _____ ( ) {
(parameters; name)

_____ ;

}

in main

class name | var = new | class name ( ) ;

var . name ( ) ;

Call here

Overloading

(same name method (function) recieving different parameter type)

ex. public String say ( ) {

_____ ;

}

public String say (boolean x) {

_____ ;

}

Search List for custom object

in other class

public static void search (name) (List < __ > ____ (class, name), String ____, String ____) {
(what to search)

for (int i=0 ; i < user.size() ; i++) {

if (user. _____ (getter method) . _____ (object) ( ) == _____ (what to search)) {

}

}

Static Methods

(call on class directly)

in other class

public static void _____ (name) (parameter) {

_____ ;

}

in main

_____ . _____ ( ) ;

class | static method name

Method Overriding (Intro)

(replace default outcome)

ex. @ override ← declare @override

public ____ ____ ( ) {

_____ ;

}

* some cases the parameters are Object type

ex Object ____ ;
name

Return Custom Object

ex `public static ____ ____ ( ) {` (____ = class, ____ = name)

```
if ( ) {
    return ____;
}
return null;
```

null = no value

Passing by Value / Reference

`public static void ____ ( ____ ____ ) {` (____ = name, ____ = class, ____ = var)

```
    return ____;
}
```

Inheritance

`public class ____ extend ____ {` (____ = class 1, ____ = class 2)

reference from

Abstract

`public abstract class ____ {` → make class not inheritable

```
}
```

Abstract method

prevent this method being reference by other classes

`public abstract void ____ ;`

```
}
```

Polymorphism

(different obj do different things)

in different sub inherited class

```
@Override
public void ____ ( ) {
    ____;
}
```

→ override for only this class

Constructor method

`User u = new User ( "____" )` ← custom constructor

(User = class, User = name)

`User ( )` → default constructor

Referencing default constructor in class

when class is mention → do this

```
public ____ {
    ____;
}
```

(____ = class)

Referencing Custom constructor

```
public ____ ( ____ ) {
    ____;
}
```

(____ = class, ____ = parameter)

Involve parent class with superkeyword in the reference class

```
@override
public void ____ ( ) {
    super.____ ( )
}
```

(____ = name)

Readonly Field

(private after constructor method)

* read only

make object private

in class

create user constructor

public user (______) { (parameter)

______ = ______ ;

______ = ______ ;

}

in other class (inherited)

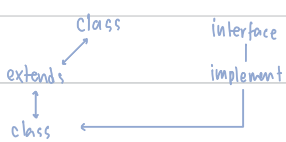
public ______ (______) { (name; parameters)

super (______) ;

}

Interface

(Make requirement for object)



Create Interface

public interface ______ {

name = filename

@ overid

public void ______ ( ) {

}

}

in class

public class ______ implements ______ {

name / name

}

Final Methods

in extends class

public final void ______ ( ) {

→ in class: can't use this method / refference

}

Final classes

public final class {

make class can't be inherrited

}

Enum

in class

public enum ______ { ten , ______ , ______ } ;

stat name; data; no: " ______ "

public ______ ______ ;

stat name / stat name

in main

constructor method : S

S obj . stat name = S obj . Stat name . ten ;

data elements

when in use (S obj . stat name)

Enum in switch

ex. switch (s . stat)
obj enum name

{

case _____ :
element 1

break ;

case _____ :
element 2

break ;

⋮

}

What to do next:

Web development

Interface dev. Android apps

Generic program

database in file out file

plugins