JAVA

ENG.ASMA ABEDLKAREEM

BY:SARAH ALKASSASBEH

POWERUNIT

* Machine language "Computer's native language" differs among different types of computers, it depends on how the processor "CPU" read the language. Intel processor reads for example (0101) differently of how amd processor reads it.

* We can run high-level languages codes on different types of machines "processors" (platform independent)

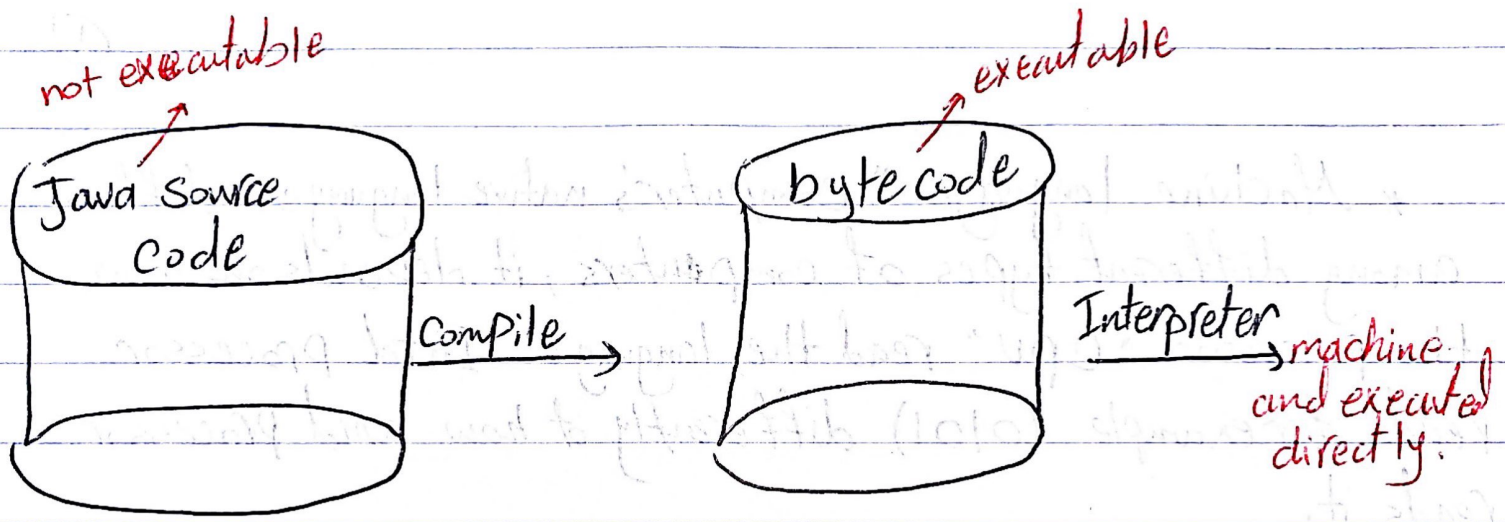
* source code → high-level language code

* languages that use compiler to translate the source code
→ Compiled languages. ex: C, C++

* languages that use an interpreter to translate the source code
→ interpreted languages. ex: Python.

* In Compiled languages the execution is not portable
→ Compilation can't be done on an intel processor then execute on an amd processor.

* Java is in between the compiled languages and the interpreted languages.



byte Code :- language provided by Java team, "low-level but platform indepen."

- API → library (Contains predefined classes and interfaces for developing Java Programs).

- JDK → programs (kit) for developing and testing Java Programs. (Java Development Kit)

- IDE → programs designed to help programmers to write the code in an easy way by using graphical user interface.
  ex. netbeans (icons)
  - JDK is a part of IDE

- Java editions depend on platform

- methods ≡ functions in C++

- Class : place where we write our methods, Also, we can't write any method outside the class body.

- Console :- Area where the user enter the information and extracts it as well.

- to define a class in Java Programs we use (Public)
  - Public / class are reserved words in Java.
  - reserved words are always small letter.

- Main method defintion (Public static void main)

Public static void main (String[] args)

// → Comment "single line comment"

/* → multiple lines comment.

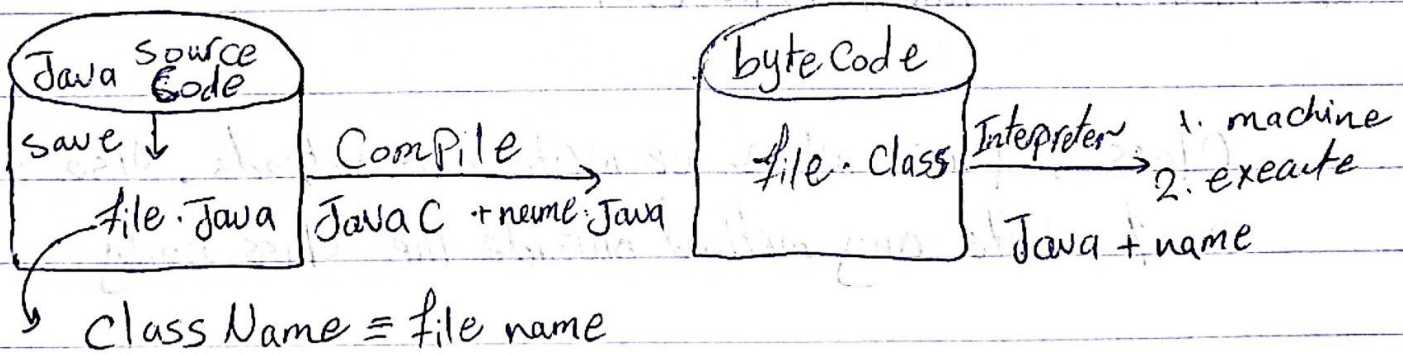
/* - beginning

*/ in the end.

- Printing the code → System.out.println (يترك سطر جديد بعد الطباعة)

System.out.print

" " " قبل الطباعة.



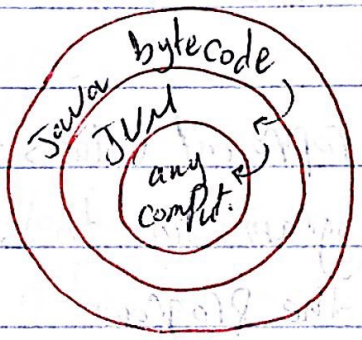
in Java Interpreter ≡ Java virtual machine (JVM)

- if you have (JVM) in your device then you can execute any byte code.

Runtime errors → type of errors can't be discovered by the compiler. (another name → exception)

incorrect result → logical error or output error

* as long as the computer has JVM it can run any Java byte code



* class loade → Part of JDK it loads the bytecode of the class to memory

* bytecode verifier → used by JVM to check the validity of the bytecode and to ensure that the bytecode does not violate Java's security restrictions to execute it.

* Identifiers → name of things in the program
letters, digits, underscores, dollar sign
- Cannot start with digits
- Cannot be a reserved word
- Cannot be true, fals, null
- Should be short and descriptive.

* Variables → represent values that may change in the program, and to store values to be used later in the program
- we declare the variable in the program (type + name) so the compiler allocate appropriate memory space for the variable based on its type.

ex:- Char stores in 16 bit so the compiler allocate space in memory (16 bit) for the variable.

- we can define more than on variable in the same line if they are the same type.

* Assignment statement :- assign a value for a variable after declaring it.
- we use equal sign (=)

- we use underscores when the constant name contains more than one word (ex. MAX_NO_STUD)
  - all characters in the words must be capitals.

- Conventions are not rules.

- Total number of data type in Java = 8, 6 of them are numeric.

- 1.0 / 2.0 = 0.5 (floating point numbers)

  but 1 / 2 = 0, takes only the integer numbers (integer part)

- floating point numbers by default considered double values.

- if you don't give the variable an initial value, the JVM will not put any value in the variable (its value will be rubbish)

```
int x;
System.out.println(x=1);
```

```
int x;
x=1;
System.out.println(x);
```

```
int x = 5;
double y = x; // No syntax error because this is widening
System.out.println(y); // 5.0
```

* Casting (conversion): automatic / implicit → widening.
(up-casting)

* Narrowing

```
double x = 3.0;
int y = x; // syntax error.
int y = (int) x;
```

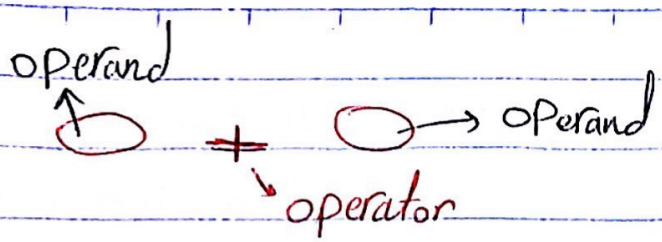
Casting (conversion): not automatic / explicit
(down-casting)

0X (means hexa)

hexa ^0XAB0041
24 bits

string → is not a primitive data type, it is predefined class in the java library.

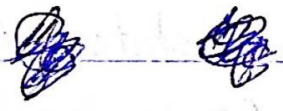
capital ^String
//



↓

Concatenation (if at least one operand is string)

&& (logical anding)



* in printf statement (%s) → string variable
(%d) → integer variable
(%f) → floating variable

(%4d) → 4 integers (4 digits)
spaces

* local variables → variables in the main method

* in methods
  we can 1) reusable code by calling (invoke) without copy-paste
  2) organize and simplify code

* Reusable code → same steps
  instead of repeating, we can group the same steps into a method
  * Method body → contains the statements to execute. (common code)
    (between curly braces)
  * Method header → (public static ....)

ex:- public static int sum (parameter list)
method signature ← (sum → name (noun); int → type; parameter list → parameters → formal variables)

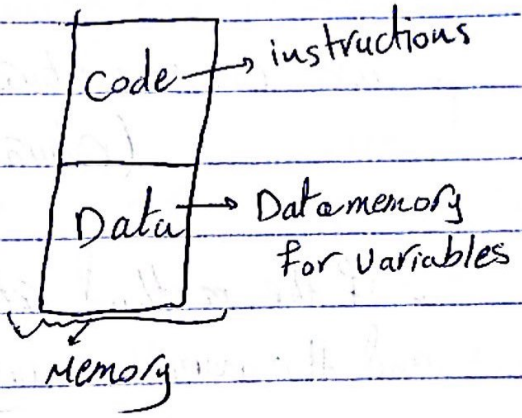
ex:- int variable ( )
return type ←

return → result of the code.

* then in the main method we involke (call) the method's name and give (put) values in the parameters. (actual / argument parameters)

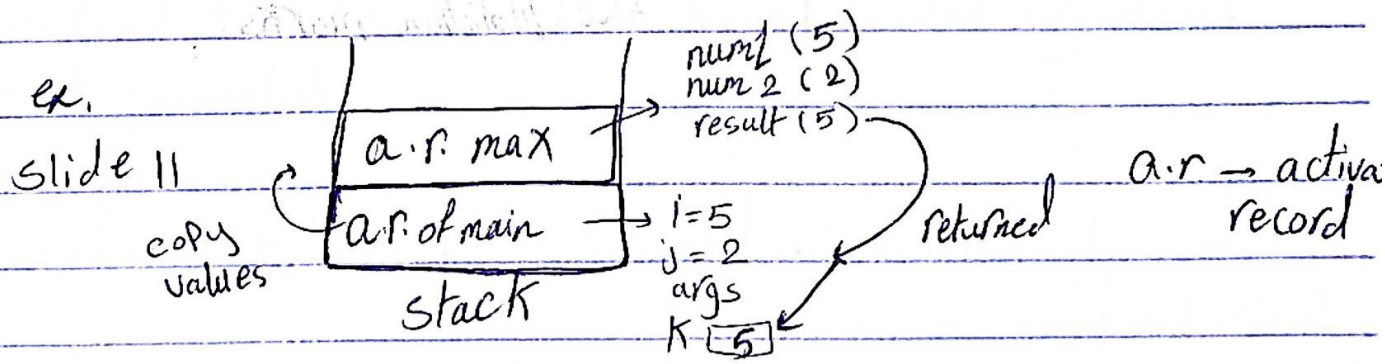
- we write the method inside the class and out of the main method

- Data memory
  - heap
  - stack
    - where all variables in the methods will be stored.



- activation record located in stack and it contains all the local variables in the methods.

ex. slide 11



- Once the value returned to the main, the activation record will be destroyed (removed)

for max

- Once the program ends, the main method activation record will be destroyed too.

* what you can pass to the methods maybe variable, value, Constant.... etc.

* we can use return statement in the void method
(return;)

* if the method return boolean the the type is boolean and the name usually starts with "is"

ex:- public static boolean isPrime

* to define an empty string:
string s = ""; → without space between the quotation marks.

* Modularicing → create methods (modules)

* Methods in same class must have unique signature.

* Signature (name + parameter list). unique name or unique parameter list.

* Methods overloading → methods with the same name, but different parameter list, in one class.

* invoking is based on the method signature, but when we have 2 or more methods with the same name, then the compiler will find the most specific method for the invocation based on the arguments and parameters.

* matching between parameters and arguments → type, order, number of arguments and parameters.

* to choose which method is the most specific match → choose the one that need the least number of upcasting the Parameters required.

* Scope of a variable is a part of a program where the variable can be used.

↓

starts from declaration

↓

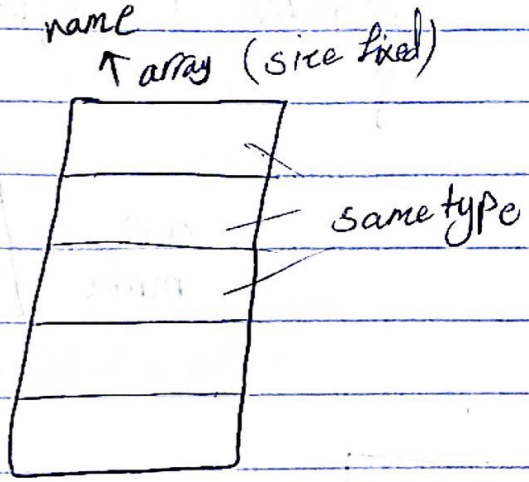
to the end of the method or the curly braces "{}"

* local variable must be declared and assigned a value before it can be used (read).

* Parameters → scope starts from the beguing of the method to its ending.

* we cannot declare a local variable twice in the same block or in nested blocks (overlap).

* An array is a data structure which stores a fixed size sequential collection of elements of the same type.



* you can't scale out or scale in in the array (fixed size)

array declaration — array creation

* refrence variable [address] (step 1)
↓
ref. data.

ex:- double [] myList; → declaration.

refrence variable — name of the Ref. variable

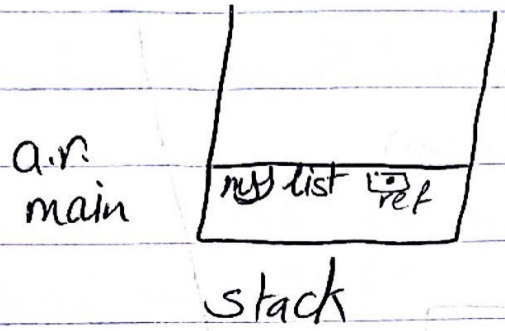
* Ref. Var = new+element type [array size];
→ # of elements
↳ creation

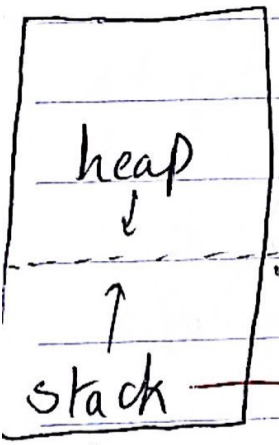
ex:- double [] myList; → declaration
myList = new double [10]; → creation

* we can declare and create the array in the same line.

```
Public static void main (String [] args) {
    double [] myList = new double [10];
}
```

↳ 10 double values



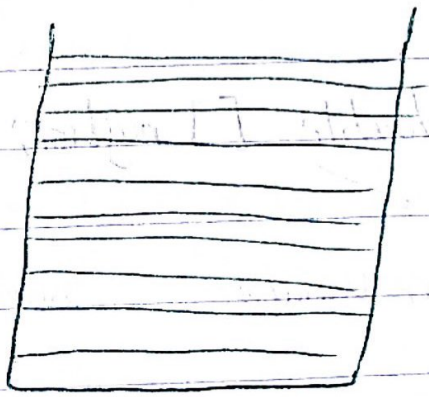


→ anything created by the word (new) for ex:- data structure

virtual line

→ a.r of methods local variable

Data memory



heap (10 elements)

we can reach them by the ref. variable on the main

array size

* if the size for example is 10, then we start in index 0 and end with index (10-1) = 9

- to know the size or length of an Array (obtaine the size) by :-

arrayRef.Var.length
↳ name

---

ex.) int [] a = {1, 2, 3, 4};
System.out.println(a);
↳ it will print the address of a

- Array of char.

char [] city = {'A', 'm', 'm', 'a', 'n'};
System.out.println(city);
↳ it will print (A, m, m, a, n)

- foreach loop enables you to traverse the array sequentially without using an index variable

↳ Array's elements type

for (double e : Ref.Var)
{ system.out.println(e+" ");

only if you want to traverse the array elements sequentially.

exception ≡ runtime error

* if we skip the first element on the array and start with index 1 ⟶ off-by-one error.

. Accessing an array out of bounds ⟶ ArrayIndexOutofBound Exception
"runtim error"

list1 ⟶ Contents of list 1
list2 ⟶ Contents of list2

list2 = list1;

list1 ⟶ Contents of list 1 (we have one array referensed by 2 variables)
list2 ↗

this is not Copying

list2 → Garbage

* To copy the contents of one array into another, you have to copy the array's individual elements into the other array using a loop.

* to print the array elements in revese order
for(int i = list1.length-1; i >= 0; i--)

* whenever you pass an array of elements you actually pass the reference, and whenever you return an array you actually return the reference

---

* Variable length Argument list

* by arrays we can define one of the methods' parameters or arguments as a (variable length), which means that we can pass any value to it.

* The variable length parameter, always must be the last one in the parameter list.

ex: m(int x, double y, int ... z)

here you can pass integer values to it. Also, you can pass nothing.

invoke

m (5, 5.5, 1, 2)

m (5, 5.5, 1, 2, 3, 4, 5)

* Type ... Name

- Command-Line Arguments ≡ Main method arguments
  - Parameter (array of strings)
  - (String [] args)
- Main method is invoked by the JVM

ex:- java Test/Main "First num" alpha 53
- "First num" → args0
- alpha → args1
- 53 → args2

- in netbeans to pass these arguments :- Press ↑ Run → set project Configuration → Customize → write the arguments
- these arguments will be stored as strings.

- in two-dimensional Arrays we will need row idex, Column index

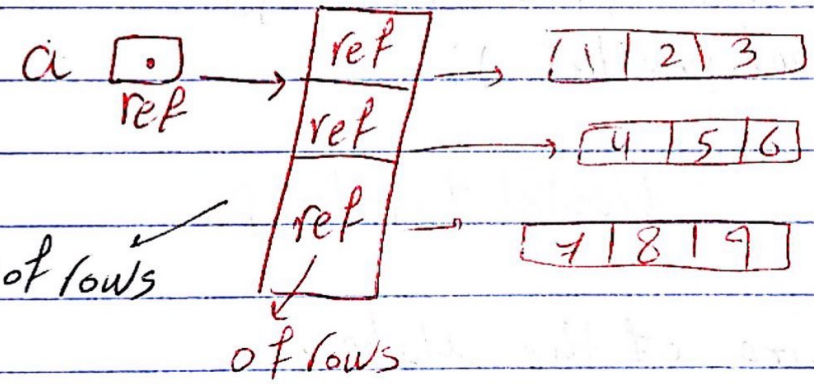
type [ ][ ] array;
- # of rows
- # of columns

- we use two-dimensional array to represent table or Matrix.

- Two-dimensional array is an array of arrays
  it is an array in which each element
  is a one-dimensional array.

ex:- int [][] x = new int [3][4]
↳ Ref.Var for one dimensional array.

ex:- int[][] a = {{1,2,3},{4,5,6},{7,8,9}}



- System.out.println(a.length);

3

↓

array size (number of Ref)

- for(int i=0; i< a.length; i++)
  System.out.print(a[i] + " ") = 3 addresses

- for (int i=0 ; i<a.length ; i++)
  System.out.print (a[i].length+" ");

3 3 3

- to print the matrix

```
for(int i=0 ; i< a.length ; i++) {
    for (int j=0 ; j < a[i].length; j++)
        System.out.println(a[i][j]+" ");
    System.out.println( );
}
```

Nested for loops

- to know the size of the Matrix

```
int size = 0;
for (int i=0 ; i <a.length ; i++)
    size += a[i].length;
```

objects and classes (Part 1)

- data fields :- places where we store the data of an object
- behaviors :- actions represented as methods.

- objects are instances of classes

- instantiation → creating an object (instance)

- objects of the same type → same class.

- Defining classes :-

ex:- Class Circle { ← type

Variables → data fields " any object of type Circle must have this Variable

Methods → actions { without public, static + this method has access to the data field (variables)

}

}

make objects (instance)

- objects are Reference Variables.

object's Members

- data fields
- methods

to access the object's members we will use the dot operator (.).

refvariable . data field / method.

ex. → object ref. variable

myCircle.radius

or

mycircle.radius = 5;

- To invoke a method

  refvariable.method(argument);

- Remember :- program starts execution in the main method.
- The name of the file = Public class name.

data fileds :-

ex:- → = 0 by default in the heap

double radius;

double radius = 5;

↓

any object of type (Classname) will have 5 for its radius by default.

* only changing the ~~default~~ default value.

- The Constructor must have the same name as the Class.
- a Class can be defined without any Constructors. then the Compiler will define default construc

name ( ) { } ←

empty

- we can define more than one Construct. with the same name in the same class, but with different Parameters list

- to copy the value to other object.

ex:- Circle1.radius = Circle2.radius

- we define and create the class out of the main class. and we define and create the object in the main method.
- we invoke the methods in the main method as :-

objectname.method();

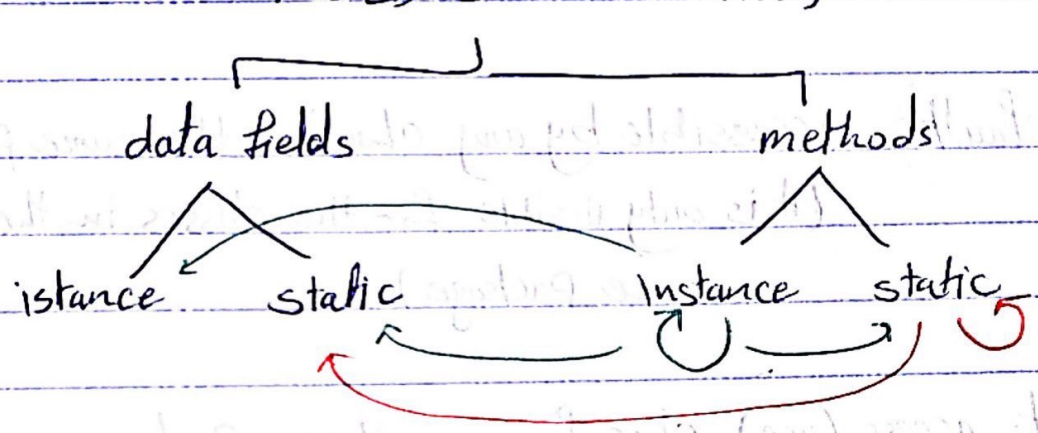
- static datafields → every object will have the same value (datafield, variable value)
- static datafield will have location in the memory even if we don't declare or create objects.

- when we define the method (static), we can invoke it without objects.

- we declare Constants as static.

- we can also access the static variable via objects.

* underlined in UML diagram → static.



objects have access to instances and static (methods and data fields)

Rule :- * you can access anything using instance method
but you can invoke static method and access static data fields only using static methods

* main method is static.

Visibility Modifiers → affects datafields from outside the class

default Public private

Visibility :- ability to access datafields or methods from outside the class.

- default → without any visibility modifier.

ex:- int radius.

- (default):- accessible by any class in the same package.
  (it is only visible for the classes in the same package)

- to access (use) class from another package we use "import"

  import packgename.Classname;

- if we define datafields or methods or constructors (Public), they will be visible from other packages and accessible

- if we define datafields or methods or constructors (Private) they will be accessible by the class itself.

* to access class from other packages, it must be:-
1) Public class
2) using import statement to import the class
import packageName.ClassName;

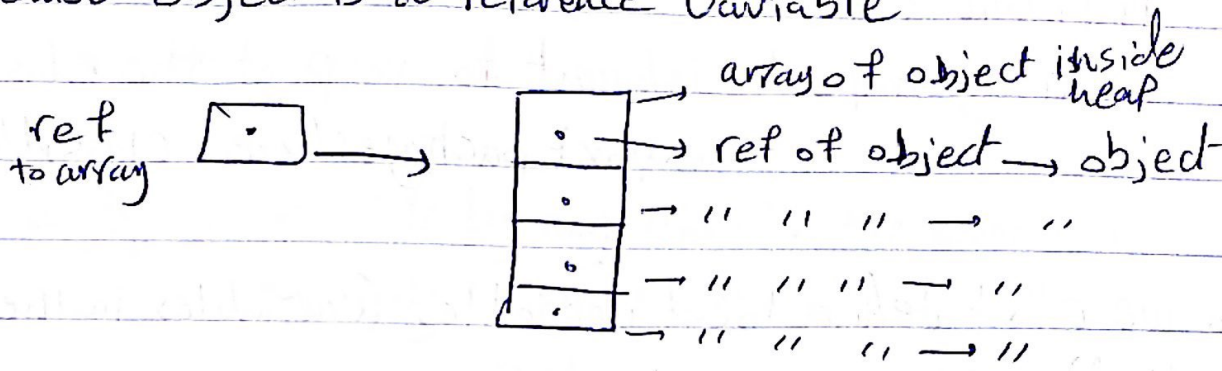
* we can't define local variables (variables in the main method) public or private.

* getter method → reads the value of the data field
Public returnType getPropertyName()
type of the property name
the name of the value which it will return
ولا شيء

* setter method → write the value of the data field.
Public void setPropertyName (datatype propertyValue

* getter method for boolean data type, fields :-
Public returnType isPropertyName().

- Array of objects → 2 level of refrencing because object is a reference variable



Array declaration → Datatype [] Array = new datatype [size];

# of refrences

by default they will be null.

before initializing.

- Reading input from the Console:-

using scanner class which is located in java.util Package

import java.util.Scanner;

- methods inside scanner class are instance methods.

object of type scanner

Scanner objectName = new Scanner (system.in);

- Immutable class ≡ objects are immutable
  - class: ما يمكن التعديل عليه
  - objects: ما يمكن التعديل على الـ data fields

* immutabe class :-
  1) data fields must be private (all of them)
  2) There can't be mutator "setter" methods for data fields.
  3) No accessor "getter" methods can return a reference data field.
     "it is ok if it returns a copy of the refrence data field"

- this Reference / this keyword, has the reference of an object (~~return~~ print the datafield of the object which we invoked the method on)

- if we define a local variable and a data field with same names then the data field will be hidden, so we use "this" to reference the hidden instance data field

- Math Class ⟶ java.lang
  ↓
  you don't have to import it because it is imported by default, which means that you can use the Math class directly.
- Methods in the Maths class are static.
- we have 2 Constants (static) in the Maths clas (PI, E)
  ↓
  Math.PI
- Math.random() ⟶ every time we run the program it generates a random number(double) between 0 and 1
- to change the range of return value to (0,10)
  Math.radom()*10 ⟶ Multiply
- " " " " " " " " " " (50, 100)
  (,50+ Math.random()*50
- to return an integer instead of double value ⟶ down casting
  (int)(Math.random())

البداية ↑ a + ( ) * b ↓ طول الفترة

* Array of objects →

type of elements in the array [] name = new [size];

↙ Class of the objects

```
for (int i=0; i< array length; i++)
    System.out.println(array[i]);
```

↓

Null before initializing / creating

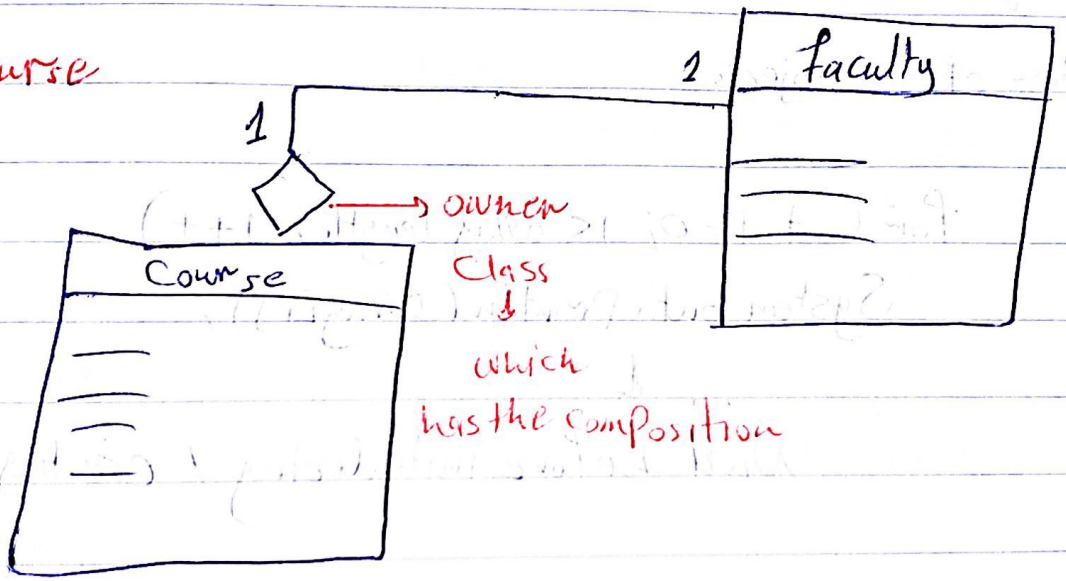
System.out.println(array[0]); → before initializing/creating the object

Runtime error (NullPointerException)

So, we must create the objects by (new+Constructor) after creating the array.

```
for(int i=0; i< array of objects.length; i++)
    array of objects[i] = new Class name();
```

Class Composition :- one of the data fields in the class is an object of another class.



each course has one faculty.

↓

one of the data fields in course class is of type faculty → object

inside the Course class :-

declaration ← Faculty faculty;

Class name ↑ → object name

Arrays class

java.util.Arrays

- Arrays class → Contains static Methods

1) double [] array = { _____ }

java.util.Arrays.sort(array); → ترتيب تصاعدي

. ParallelSort → faster than sort

Char [] chars = { _____ }

Arrays.sorts(chars); → حسب ال ASCII

Array.sorts(chars, 1, 3); → ترتيب الجزء من 1 إلى 3

2) ~~binarySearch~~ binary Search (return ~~address~~ index of the key)

→ ترتيب تصاعدي

int [] list = { 2, 4, 7, 10, 11 ..... }

System.out.println("1. Index is " + Arrays.binarySearch(list, 11));

outPut → Index is 4

3) strictly equal → متساويين بالترتيب والقيم

Arrays.equal → return true if the are strictle equal

else → returns false.

4) fill — Array.fill → to fill the whole array or a partial of the array with a specific value.

5) toString → we must use it inside a print statement to print the array

6) CopyOf → it takes the array to copy and the index where to ~~start~~ ~~copying~~ copy for

```
double [] array2 = Arrays.copyOf(List1, List2.length)
                                 ↓
                    to copy the whole array
// // // = Arrays.copyOf(List1, 3)
                    from index 0
                    to index 2
```

Method Abstraction and stepwise Refinement

Abstraction ≡ hiding (encapsulating) the details and focusing on the interface.

- Method abstraction :- separating the use of method from its implementation

- System . out . print
  - System → Class
  - out → object (static)
  - print → Method (in java library)

- we use the methods in java library without having to know its implementation

Class abstraction ⟶ separates class implementation from how the class of used.

- Class Contract :- methods and fields ~~function~~ to be used from outside the class (with discription of how they are expected to behave.

Class Relationships

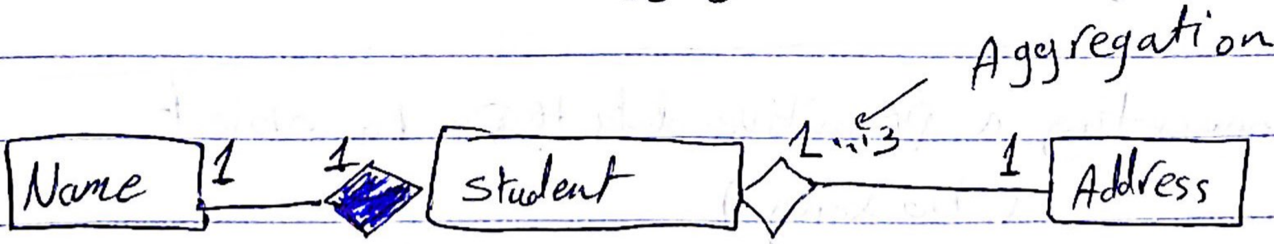
1) Association
2) Aggregation and Composition
3) inheritance

- Association → describes an activity between two classes

- Aggregation → special form of Association
  ↓
  ownership between two objects (class owning objects from another class) or from the same class

- Aggregation = has - a

- owneR object → Aggregating object
- owned object → Aggregated object.



- each student has 1 address
- every address can be owned by 1 - 3 students
- each 1-to-3 students can have 1 address

diamond shape → كدة فاضي

- if the owned object can be owned by more than aggregated object → Aggregation (not filled diamond)
- if ~~the~~ every owned object can be owned by one aggregated object → Composition. (filled diamond)

↓

For your own knowledge

- Processing Primitive Data Types values as objects:-
- wrapper calss (Integer, Double, character)---etc
  ↓ ↓ ↓ ↓
  int double char

we using for processing primitive data type value as objects and making operations on them.

↓

java.lang

- Converting a primitive data type to object (boxing)
- " an object to a primitive data type (unboxing)

int x; → Primitive } converting
Integer y; → object. } (automatic)

Integer y = x; → auto boxing from Primitive to

- BigInteger class is used to deal with very big integer numbers.
- BigDecimal class is used to deal with very big floating point values.

↓

then are in ~~java.Math~~ java.math

↓

arithmetic operation are methods in these classes so we invoke them.
(add, subtract, multiply, divide, remainder)

- BigDecimal ← إذا قسمنا رقمين كل بعض وكان الناتج دوري وما حددنا scale بين عدد الأرقام بعد الفاصلة ← ArithmeticException

- Inheritance is a relationship between classes (super classes and subclasses)

Properties of a class = datafields
behaviors of a class = methods

- subclasses will inherit all the properties (common) or methods from the super (generalized class).

- in subclasses we use:-
  extends SuperclassName

- to use or access private datafields from Super-class we use set and get methods.

- Subclass > superclass

- Private datafields is superclass cannot be used directly in subclass

- the supclass can only extend from one class.

- Super keyword (to invoke constructor from super class)
- Constructors are not inherited.

Super (datafields to set)

- Super must be the first statement in the constructor.

- if we don't invoke a constructor from the super class using (super keyword) of from the same sub class using (this keyword) then the compiler will add super(); by default and use the default values for the datafields.

- overriding → define a method in the superclass and redefine it in the subclass with the same header.

super.method();

↓

to invoke the method from superclass after overiding.

super keyword

Call a superclass constructor

super() → no-arg constructor

super(arguments)
parameters

must be the first statement in the constructor

Call a superclass method

we can use the super keyword to invoke any method from the superclass by :-
super.method
(overriding method)

- if we call an overriding method without super keyword then we will have → logical error.
- Private method cannot be overridden. but (No syntax error)
- static method cannot be overriden
  ↓
  Methods in subclass will hide the methods in superclass → hidden
- if we want to call static Methods from subclass
  ↓
  SuperClassName.staticMethodName.

- default method is not accessible from another package so it cannot be overridden.

- override Annotation → before the overriding method, to make sure that you are making overriding correctly.

@Override

if you make any mistake while overriding → syntax error.

- by default every class in java extends object class.

- every class in java will inherit the (toString) method.

- override the toString method in subclass

- System.out.println(object); equivalent

System.out.println(object.toString());

- in Java there are 3 important Concepts :-

1) encapsulation 2) Inheritance 3) Polymorphism

- Polymorphism depends on inheritance

: any object of type subclass is an object of type super class Also.

- you can define an object of subertype and create it with the subtype

ex:

```
GeometricObject g = new Circle();
```

GeometricObject → super → declared type (of type super)

Circle() → sub → actual type (of type sub)

```
GeometricObject g = new Circle(1);
System.out.Println(g)
```

searching for ToString Method in Circle first. (from actual type)

Dynamic Binding

declared type — actual type

```
GeometricObject[] g = {new GeometricObject(),
new Circle(5), new Rectangle(1.2, 3.4)};
```

↓

Polymorphism.

```
for (int i = 0; i < g.length; i++)
    System.out.println(g[i]);
```

out :- first element → toString in GeometricObject class (actual type)

second element → toString in Circle class.
third element → toString in Rectangle class.

- Searching for the toString method in the actual type class.
- if there is no toString method in the actual class → searching in declared type class.

↓

Dynamic Binding.

Signature Matching :- Compiler makes sure that the method is existing, if not → syntax error according to the method name and its parameters.

- Compiler starts searching from the declared type

ex) GeometricObject g = new Circle(5);

System.out.println(g._____);

- we can't access methods in the sub class (Circle), we can only access declared type class methods and data fields.

so we use downcasting

↓ (من نوع عام لنوع خاص)

System.... (((Circle)g). method in Circle);

Circle mycircle = new Circle(1);

GeometricObject g = mycircle;

upcasting

- ClassCastException → when you cast a class into a class that cannot be cast to.

* Polymorphism is upcasting.

Dialog boxes — message
input

import javax.swing.JOptionPane;

① JOptionPane.showMessageDialog(null, String to show)
static ↵ (void method) (Parent component)

default icon → information icon (!)
" title → Message
} to change them

JOptionPane.showMessageDialog(null, string, "title", JOptionPane.chooseicon);

②
JOptionPane.showInputDialog("string");

JOptionPane.showInputDialog(null, "string", "title", JOptionPane.chooseicon);
↓
returns string

- we can't compare strings using equality operator, so we invoke equals method.

- if the method can't convert the string which is return from JOptionPane.showInput---- to the ~~primitive~~ required data type, then, runtime error will occure

(NumberFormatException)

The Date class

+ Date() ⟶ no args. constructor → date object with date and time ~~when the object created~~. (current time)

+ Date(elapseTime:long) ⟶ milliseconds elapsed since January 1, 1970, GMT.

+ toString(): String → Contains date object details

+ getTime(): long

+ setTime(elapseTime:long): void.

```
import java.util.Date;
```

→ current date

```
Date currentDate = new Date();
System.out.println( currentDate.toString) ;
```

returns current date details from the device

equivalent to :-

```
System.out.println (currentDate);
```

The Random class → generates random value

```
import java.util.*;
```

or,

```
import java.util.Random;
```

Generics

- Generics let you parameterize type. → change type of Parameters in methods.

- you can define a class or method with generic types, then the compiler can replace with Concrete types

, an example of a generic class is the ArrayList class

java.util.ArrayList;

- an ArrayList object can be used to store a list of objects.

. ArrayList has unlimited number of objects.

→ Generic type

java.util.ArrayList < E >

↓

بنحوض نكان E بالنوع الأولي لل objects داخل ال ArrayList

و بالتالي تعويض النوع بكل ال methods .

- Abstract Method → without implementation (without body)
(keyword → abstract)

ex: Public abstract double getArea();

- إذا احتوى ال class على abstract method ← abstract class

- declared type بين عادي ، actual type

- إذا كان ال actual type غير abstract class

(cannot be used to create an object)
(cannot create instance using "new" operator)

بين ال array عادي

- لتغيير تعريف ال constructors ← protected

- Abstract classes and methods in UML diagram → italicized ( )

- abstract methods are non-static (can be overriden

- interface → it is like constructors
it has only constants data fields
and abstract methods.

- interface → تعريف صفة او سلوك
تعرف مشتركة بين classes بيعضم علاقة
او ما في بيعضم علاقة.

- interface → يتعامل معاملة ال class من قبل
ال compiler

عدد الـ files.class = عدد الـ classes + عدد الـ interfaces

- any file can only have one public class or one public interfaces

- all the methods in the interface are abstract methods.

keyword :- interface

- ال object باخذ الصفة من ال interface عن طريق
implement keyword
ال class اللي بعمل implement لازم يعمل override لكل
ال methods بال interface

interface inheritance :- The relationship between an interface and a class that implements it.

- interface is abstract.

<<interface>>
in UML diagram ↓

- interface inheritance → <<interface>>
UML diagram ↓

- interfaces used exactly the same how we use the abstract classes.

- data fields in the interface are static
(public static final)
↓
because they are constants.

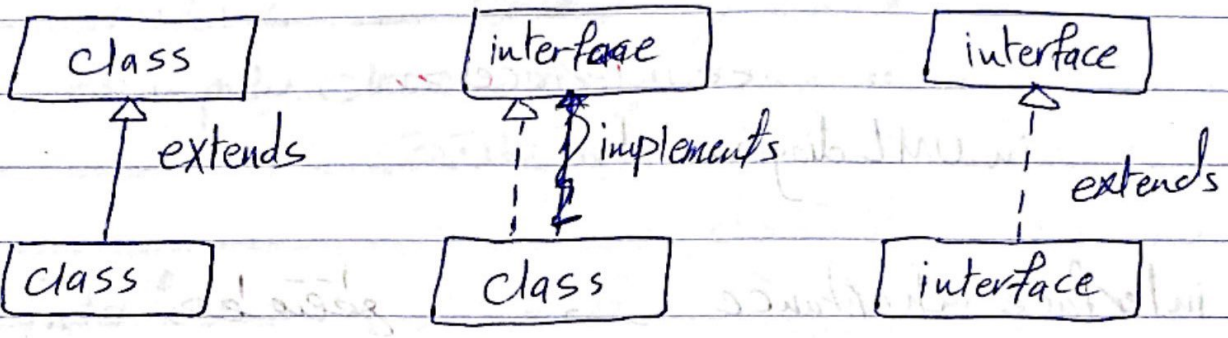
- Class can implement more than one interface

- inheritance between two interfaces is possible using (extends) keyword

- We have sub and super interfaces

↓

Can ~~implements~~ extends more than one interface



- A class implementing interface must implement the abstract methods defined in the all interfaces, to be concrete.

↓

abstract method وإلا لازم تكون abstract method

- overriding بدون

- for classes the single root is the object class, but for interfaces there is no single root

- strong is-a relationship → class inheritance (type)
- weak is-a relationship or is-kind-of → interface (Property)

lab 10 tutorial   overriding لما في عادي abstract لأنه.
→ لازم rectangle circle بس بال howToColor لـ

Public abstract class GeometricObject implements
Comparable <GeometricObject>, colorable {

it has 2 abstract methods

```
@override
public int compareTo(GeometricObject o) {
    dateCreated.compareTo(o.dateCreated);
```

. in Circle Class

Geo..... extends circle لأنه

```
@override
    public int compareTo ( GeometricObject c) {
if ( c instance of Circle) {
        if (radius>((circle)c).radius) return 1;
    else if (radius ==((circle)c.radius) return 0;
    else return -1;
 }return else super.compareTo(c);
}
```

compareTo of Geometric object.

Scanned with CamScanner

. CompareTo in geometricObject (Area)

```
@Override
Public int CompareTo ( GeometricObject o) {
        if (getArea() > o.getArea()) return 1 ;
   else if (getArea < o.getArea()) return -1;
   else 0;
}
```

ال Public interface بكون ب file لحاله .

```
Public interface colorable {          → Calarable.java
        Void howToColor();   ≡  Public abstract Void
                                      howToColor();
```

. in Circle class :-

```
@Override
   Public Void howToColor(){
if( filled ) System.out.println("Color . . . . . "+ Color);
else System.out.println (" . . . . . . ");
}
```

↓

Same in rectangle class

- Comparable c1 = new Rectangle (1, 2);
- Comparable [] c2 = {new Rectangle (2,3), new circle(), new Date(), }

any class that implements Comparable

declared type يكون interface ال استخدام معناه .

ArrayList <GeometricObject> go = new ArrayList();
go.add(new Rectangle());
go.add(new Circle());

object

~~GeometricObject~~ [] array = go.toArray();

ArrayList ل Array تحول ال

object

~~GeometricObject [] array = (GeometricObject[]) go.toArray~~

Arrays.sort(array); ← ترتيب تصاعدي بناءً على Area (Comparable ال في)

هل هو موجود بال array

```
Circle c = new Circle (4, "Red", false);
System.out.println(go.contains(c));
```

c.equals (Array element)

false

لأنه ما في object go له radius = 4

```
Circle c = new Circle (10, "Red", false);
System.out.println(go.remove(c));
System.out.println (go);
```

موجودة بال array من قبل

to string method for each element

true

Array elements without C

```
for (int i = 0; i < go.size(); i++)
if (go.get(i).getArea() < 10) { go.remove(i);
    i--; }
```

بديل لـ

```
Iterator goI = go.iterator();
while (goI.hasNext())
    if (((GeometricObject)goI.next()).
        getArea() < 10) goI.remove();
```

نوع ال ArrayList

The String class

. The string is not a primitive data type

. String message = new String ("Welcome to Java");

= equivalent

. String message = "Welcom to Java";

. we can convert an array of characters to String

① Char[] charArray = {'G', 'o', 'o', 'D', ' ', 'D', 'a', 'y'};

② String message = new String (charArray);

output ↳ Good Day

. methods in String Class :-

1) Length → عدد الأحرف
2) CharAt(index) → حرف بـ index محدد
3) Concat(s1) → s0.concat(s1) → يرجع string ناتج عن دمجهم
4) toUpperCase() → يحول الـ string لـ uppercase → s1 = s0.toUpperCase();
5) toLowerCase() → lowercase لـ " " " → s1 = s0.toLowerCase();
6) trim() → يرجع string بدون spaces بالبداية والنهاية

```
String [] tokens = "Java#HTML#Perl".split("#");

    for(int i=0; i<tokens.length; i++)
      System.out.print(tokens[i]+ " ");
```

returns array with ↲
size = 3
tokens[0] = "java"
tokens[1] = "HTML"
tokens[2] = "Perl"

helps to see if the string belongs to a pattern

"java".matches("java"); → True
"java".equals("java"); → True

"Java is fun".matches("Java.*") → anything

java string ل بدءاً إذا كانت هدف بال java string ↲

- format method is a static method → class نستدعيها باسم ال (String)

format

String s = String.format("%7.2f %6d %-4s", 45.556, 14, "AB")

floating point number ↓ ٧ ارقام يسار الفاصلة و رقمين عاليمين

↓ 6 digist integer

↓ ٤ فراغات بعد الرقم

System.out.print(s);

↳ _ _ _ _ 45.55 _ _ _ _ 14 _ _ _ AB

string ↲

- بتشبه Printf بس بدون طباعة

- String class → immutabe
- StringBuilder, StringBuffer classes → mutable

↓

more flexible

- Exception = Runtime error
- Exception Handling → error معالجة ال

- InputMismatchException → Type رقم غير ال

- إذا كان ~~الـ~~ النوع double مسموح القسمة على صفر
  الجواب ← Infinity

0/0 → double → NaN (not a number)
-3/0 → double → -infinity

- في حال قسمة integer على صفر
  ↓
  ArithmeticException: / by zero

Scanned with CamScanner

java.lang.Throwable → root class for all exceptions classes in java.
Also, it extends object.

1) error class → directly extend Throwable → Contains errors caused by the system
LinkageError → the JVM can't link the code with the java library.
VirtualMachineError → errors in JVM

2) RuntimeException → errors caused by the code itself.

3) other exceptions → ClassNotFoundException, IOException, Many more classes
↓
سبب خارجي بقراءة الكود
مثل قراءة الفايل

- System errors] unchecked → try, catch يتم اجبار
- Runtime Exceptions]

- other exceptions → { IO, notFound }] checked
↓
catch, try
↓
Syntax error ← غير صحيح

- in Catch blocks we must put the sub exception first then the super

```
try {
  ...
}
catch (RuntimeException) {        ← sub
  ...
}
catch (Exception ex) {            ← super
  ...
}
```

IO exception → X

```
catch (Exception Ex) {            ← super
  ...
}
catch (RuntimeException) {        ← sub  → unreachable
  ...
}
```

Syntax error