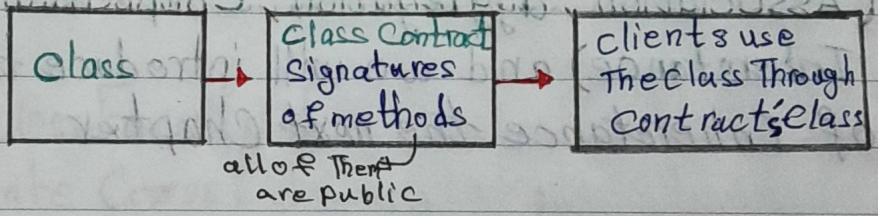
Chapter 10 :-

Class Abstraction and Encapsulation :

Def: The Class Abstraction is the separation of Class implementation from the use of a Class

Def: The Class Encapsulation is The Details of implementation are hidden from the user



Def: we named The object Oriented Programing as a Paradigm which is a way to programming any program and before OOP you were using The Procedural Programming Paradigm as an action driven and data are separated from actions for one time Because it's in the main method but OBject Oriented Programming Paradigm focuses on object which we can reuse it many times

Thinking In Objects :

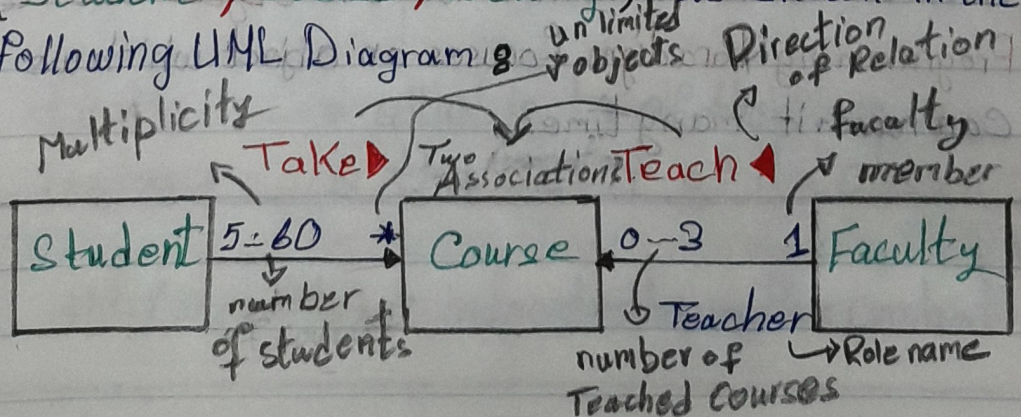
Def: Object Oriented Programming Paradigm Couples Data fields and methods into objects together and you can use it at any Program (at any main Method)

Class Relation Ships :

we have some Relationships between Classes (Association, aggregation, Composition, ~~Interface~~ Inheretance) and we will introduce the Principle of Inheretance the next Chapter

1 Association :-

Def: is a general binary Relationship that describes an activity between ~~Classes~~ Two Classes Suppose that we have Three classes which are (Student, Course, faculty) as shown in the following UML Diagram :



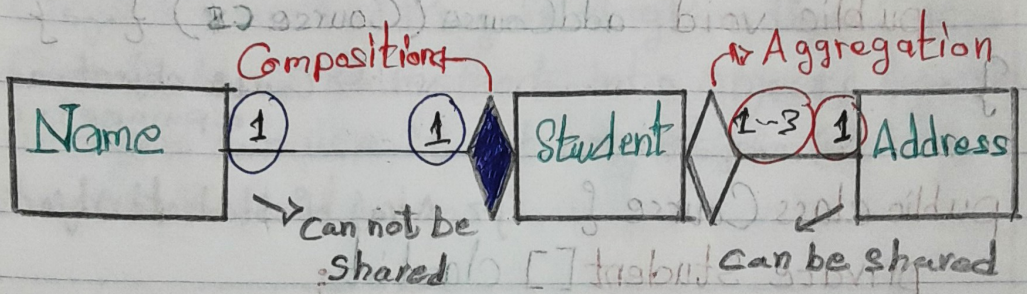
ex:

```
public class Student {          → Array of Courses objects
    private Course[] courseList;
    public void addCourse(Course c) {--}
}                                   ↳ Course object as a parameter

public class Course {           → Array of Students Objects
    private Student[] classList;
    private Faculty faculty;
    public void addStudent(Student s){--}
    public void setFaculty(Faculty faculty){--}
}

public class Faculty {
    private Course[] CourseList;
    public void addCourse(Course c){--}
}
```

[2] Aggregation and Composition :-

Defn: Aggregation is a special form of association that represents an ownership Relationship between two objects. This Relation called "has a" RelationShip. The owner called aggregating Object and the Subject is called aggregated Object and it's Class called aggregated Class

Def: The Composition is a relationship between object and it's aggregating object



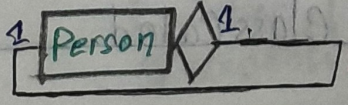
ex:

```
public class Name { // Aggregated Class
}

public class Student { // aggregating Class
    private Name name;
    private Address Address;
}
```

Composition Relationship ← owner Ship

```
public class Address { // aggregated Class
}
```

note: Aggregation may be exist between objects of The same as following UML Diagram ex:



Person may have a supervisor

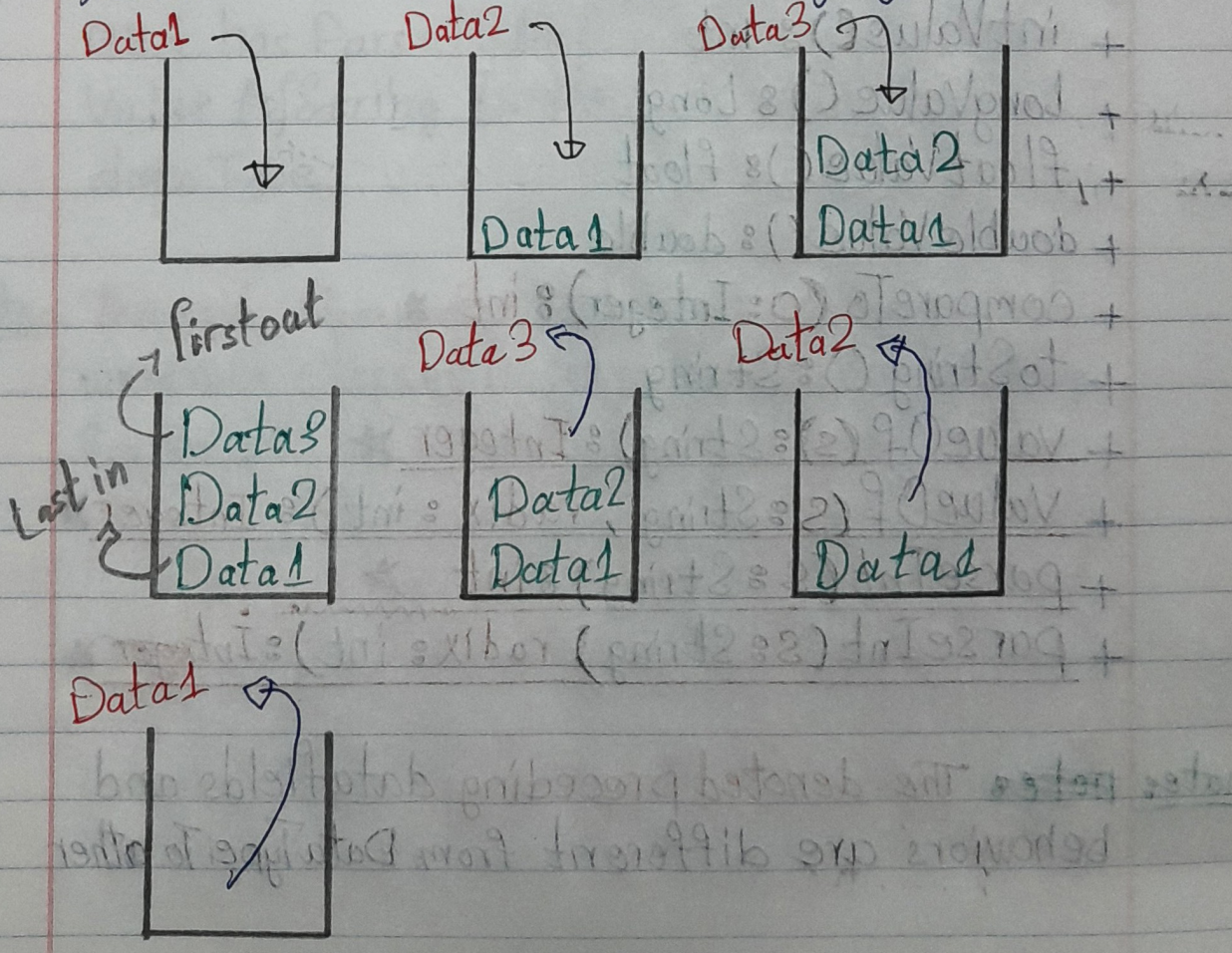
exe
```
public class Person {
    private Person supervisor;
```

notes Since aggregation and Composition RelationShips are represented using Classes in the same way we will not Differentiate them and call both Compositions for Simplicity

---

The Stack DataStructure:

Def: is to Hold data in a last-in, first-out fashion as shown in the following figure:



Processing Primitive DataType Values as Objects:

java.lang.Integer;
(→ wrapper class)

- _Value : int
+ MAX_VALUE : int
+ MIN_VALUE : int
+ Integer(Value : int) ★
+ Integer(s : String)
+ byteValue() : ~~short~~ byte
+ shortValue() : short
+ intValue() : int
+ longValue() : long
+ floatValue() : float
+ doubleValue() : double
+ compareTo(o : Integer) : int ★
+ toString() : String
+ valueOf(s : String) : Integer ★
+ valueOf(s : String, radix : int) : Integer ★
+ parseInt(s : String) : int ★
+ parseInt(s : String, radix : int) : ~~Integer~~ int ★

note: ~~note:~~ The denoted preceding datafields and behaviors are different from DataType To other

note: The wrapper Classes donot have no-arg constructors and The instances of it are immutable

note: Each numeric wrapper class has the constants (MAX_VALUE, MIN_VALUE) and has Conversion Methods (doubleValue(), floatValue(), intValue(), longValue(), shortValue()) and has CompareTo Method That returns 1>, 0=, -1< and has valueOf Static Method which creates a new object Initialized To the value represented by the Specified String and has Parse Method which Converts the Value of String to the Specified numeric data Type

note: The radix Parameter in the Parse Methods of wrapper Classes is like the following Counting Systems:

2 Binary 8 Octal 16 Hexadecimal

Automatic Conversion Between Primitive Types and Wrapper Class Types :

Def: Converting a primitive Types to a wrapper object is called Boxing and the reverse Conversion is called Unboxing and java make it by the Context Value if it's requiring an Object Autoboxing and if it requires a primitiveValue Autounboxing

ex:

```
Integer intObject = new Integer(2) ;
        <=========>
Integer intObject = 2 ; -> Autoboxing
```

ex:

```
Integer[] intArray = { 1, 2, 3} ;
                     Autoboxing
System.out.println(intArray[0] + intArray[1] +
                   Autounboxing <- intArray[2]);
```

The BigInteger and BigDecimal Classes :

Def: These ~~classes~~ make you able to compute very large capacity of numbers than The Primitive Types from java.math Package and both of Them are immutable Classes

note: ArithmeticException Occurs when the result can not be Terminated and we can use this Method to avoid This exception divide(Bigdecimal d, int Scale, int RoundingMode);

Scale → Maximum Digits Number after the decimal Point

The String Class :

Def: is an immutable Class and every String represents Object (Immutable Object) and has 13 Constructors and more than 40 methods and the strings are ubiquitous

Def: See the following example:-

```
String s1 = "Welcome to Java";
String s2 = new String("welcome to Java");
String s3 = "Welcome to Java";
System.out.println("s1 == s2 is " + s1==s2);   // as objects
System.out.println("s1 == s3 is " + s1==s3);   // as Variables
```

output: S1 == S2 is false → by References
S1 == S3 is true → by Values

~~java.util.String~~;
java.lang.String;

+replace(oldChar : char, newChar : char):String
+replaceFirst(oldString : String , newString : String)
: String
+replaceAll(oldString : String , newString : String):
String
+split(delimiter : String):string[]

note: Replace Methods returns a new derived String Obj. without changing in the Original String Object

note: The Split Method in String Class can extact tokens from the String and return it as following ex:

```
String[] tokens = "Java#HTML#Perl"
    .split("#"); -> Split("[tokens]");
for (int i = 0 ; i < tokens.length ; i++)
    System.out.print(tokens[i] + " ");
```

output: JavaHTMLPerl

note: The String Class Contains the matches Method which is most Powerful and ~~Usful~~ Useful Defence for inputs

String.matches(Regex);

The Regex (Regular Expression) is a Template or Pattern for Inputs of users and it's written as following Syntax:

"____.*" => Regex

ex: "Java is fun".matches("Java.*"); True

Pattern Start with Java

we can write Patterns with numbers or digits as a date or Id or PlateNumber of Car like following Syntax:

"440-02-4534".matches("\\d{3}-\\d{2}-\\d{4}");

1 digit, 4 digits

True Pattern with Specified digits

we Can use Regex with all replace Methods and Split Method in String Class as following Syntax:

replaceAll(Regex);, split(Regex);

toCharArray();

The method toCharArray In String Class Converts a string into array

getChars(); -> Copy Characters

which has Parameterlist int SrcBegin, int SrcEnd, char[] dst, int dstBegin

Constructor -> String(char[]);

which Converts Array of characters to a String or use the valueOf(Char[]); Method

Turn anyData Type into String -> + valueOf(Symbol:DataType):String

+ format(format:String, items:String) :String

(S) reference

ex:
```
String.format("%7.2f %6d %-4S", 45.556, 14, AB);
System.out.println(S);
```

output: □□45.56□□□14AB□□ -> space

The StringBuilder and StringBuffer Classes:

These Classes aren't immutable or more flexible than String Class You can add, insert or append new contents into Their Objects. The Difference between These Two Class is That The StringBuffer Class has a Synchronized methods Than The StringBuilder Class for MultiTasks Operations on a StringBuffer Obj.

java.lang.StringBuilder;

+ StringBuilder()
+ StringBuilder(Capacity:int)
+ StringBuilder(String:String)
+ append(data:char[]):StringBuilder
+ append(data:char[], offset:int, len:int)
  :StringBuilder
+ append(v:aPrimitiveType):StringBuilder
+ append(s:String):StringBuilder
+ delete(StartIndex:int, endIndex:int):StringBuilder
+ deleteCharAt(index:int):StringBuilder
+ insert(index:int, data:char[], offset:int, len:int):
  :StringBuilder
+ insert(offset:int, data:char[]):StringBuilder
+ insert(offset:int, s:String):StringBuilder

+insert(offset:int, b:aPrimitiveType):StringBuilder
+replace(startIndex:int, endIndex:int, s:String)
:StringBuilder
+reverse():stringBuilder
+setCharAt(index:int; ch:char):Void

note: All methods except setCharAt method are doing 2 things

① Change the contents of the StringBuilder
② Return reference of the StringBuilder

+toString():String
+Capacity():int
+charAt(index:int):char
+length(~~newlength~~):int
+setLength(newlength:int):Void
+subString(startIndex:int):String
+Substring(startIndex:int, endIndex:int):String
+trimToSize():Void

note: In setLength(newlength) method If the new length is less than the original, The StringBuilder is truncated to contain exactly newlength of characters which is argumented to the method