INFORMATION HIDING
#LECTURE-15

* ACCESS MODIFIERS IN JAVA.
  → Public → Protected → default → Private
* They "specify accessibility (scope)" of a data member, method, constructor or class.

Access modifiers

default → Visible to "same package"
Public → Visible "anywhere"
Protected → Visible to "INHERITED CLASS"
Private → Visible to the class only.

◎ If NOT specified ⇒ set default
Then that member will be accessible to any class that belongs to same directory, where the class belongs or it is basically belong to same package.

"SCOPE OF DATA MEMBER"

Access levels

Modifier	CLASS	PACKAGES	SUB-CLASS	EVERYWHERE
Public	✓	✓	✓	✓
Protected	✓	✓	✓ + To inherited classes...	×
default	✓	✓	×	×
Private	✓	×	×	×

DEFAULT

→ default is set by default.
→ It is accessible only within Package.

* Package is a set of files, that belong to a common directory.

◉ if 2 classes are default lie in the same directory, then they can be accessed ✓
(VICE-VERSA) NO COMPILE TIME ERROR.

PUBLIC

→ Public has the widest scope among these 4 modifiers.
→ It is accessible everywhere.
#also in any other dir.

```
Package P1;
Public class A{
    Void msg(){
    }
}
```

```
import P1.*;
Package P2;
class B{
    Public static void main(){
    }
}
```

Public doesn't matter whether it belongs to same directory or program file.

If same file, set it default than public.
else we will get COMPILEATION ERROR.

GOOD PRACTISE ⇒ to save all files in single class.

(X) when a class is public all its members with default access specifiers are also public. ! FALSE

PRIVATE

→ Accessible only within class
→ Can't be accessed outside the class.

(X) when a class is private all its members within class are also private. ! FALSE

CONSTRUCTOR + PRIVATE ⇒ If u make any Constructor private we can't create instance of that class outside that class.

COMPILE TIME ERROR.

↖ NOT ADVICED

NOTED

PROTECTED

It is accessible within package or from outside package only via INHERITANCE.

→ It can't be applied on a "CLASS"

Diff sub-dir ⇒ Package ⇒ Then extend to use private modifiers.

* overloading.

* If you are "overriding" it should NOT be more restrictive than that method

$X = 2+3 = 5$

$Y = 5+3 = 8$

$X = 11$

$Y =$

DEMONSTRATION - 7
#LECTURE - 16

* A good programmer should segrate a classes in different files of same dir.

* WKT, default allows us to create an instance of a class if they are in SAME DIRECTORY.
  ↳ How WITH DIFF DIR? → Then error occurs.

— DIRECTORY ≡ PACKAGE.
  Package pack/dir name.
  import pack.*;

* Constructor can't be private.
* We can override the higher specific access specifier by lower ones.
  ↳ Public → Protected → default → Private ×
  → We cannot "reduce the restrictness/visiblity of a method in a class".

Higher protected → Less protected ✓
Less protected → Highly protected ✗

PACKAGES
LECTURE - 17

There are 2 unique features of JAVA
* Packages ✓
* Interface ✓

WHAT'S A PACKAGE?
* A package is a "container" for the classes that are used to keep a class name specialised.

Eg: you can contain all classes related to sorting programs in your own package.

* JDK → has many packages (API)

WHY A PACKAGE?

→ It allows flexibility to name → avoids name space collision.
→ It allows partioning the class name space into manageable chunks.
→ It controls * naming
* visibility
It support * reusability
* maintainability

* Hierarchy fashion.

* Packages are nothing more than we ORGANISE FILE into DIFF DIR. based on FUNCTIONALITY, USABILITY.

java.lang → default one, I/P, O/P, math.
java.swing → gui.

* java has a no. of packages called API.
* Packages are collection of classes & interface.

Apart from built-in we can have our own packages. IK....

↳ CORE PACKAGES ⇒ java.lang
java.io
java.util
java.net.

How TO INCLUDE A PACKAGE?

"import" → used at beginning.
import java.lang.string;
↳ with fully quantified class name → access a particular class.
with default quantification
↳ no. of class from a package
import java.util.*;

* Instead of 1 by 1, it can be used as in single statement.
Eg: java.util.date d = new java.util.date();

DEFINING YOUR OWN PACKAGES IN JAVA

* User can maintain their own packages. Java use file system directories to store packages.

"Package" is a keyword. !

* They are usually defined by "hierarchial naming pattern". ↙
* Level is seperated by "." [PERIOD] → distant file.

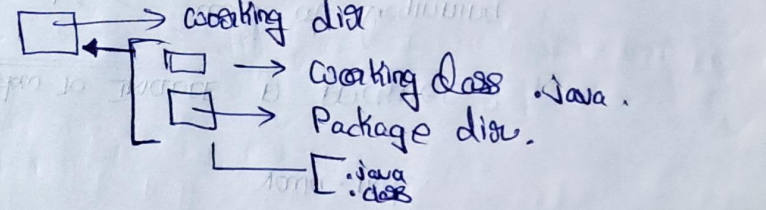
⊗ "Sub packages" are lower level inside the package.

PACKAGE NAME

→ lowercase preffered
→ No spaces
→ Main domain is mentioned.

⊗ The class that should be used, ~~while~~ is declared PUBLIC.

* The package must contain class file if it is going to be used/imported as package.
* When [package declaration is absent] ⇒ in a file, then all the classes contained in the file belong to unnamed Public package.
* A class in named package can be reffered in 2 ways. ⊗



NOTE: * We CANNOT put 2 or more public classes in .java file → otherwise there will be difficulty/ ambiguity in "NAMING" the .java file. ← NOTE

How PACK contains multiple class then? (public)
By multiple class files

Pack P
Public class A{
}

Pack P
Public class B{
}

P ▷ A. Java
B. Java
A. class
B. class

PACKAGES - II
#LECTURE - 18

ACCESING A PACKAGE:

"classpath"

① DEFAULT: all packages in your working directory
② Setting the classpath environmental variable.
* ③ you can use -classpath option Java & Javac to specify path to your classes.

-classpath ⇒ LOCATION OF THE CLASS FILE IN THE DIR.

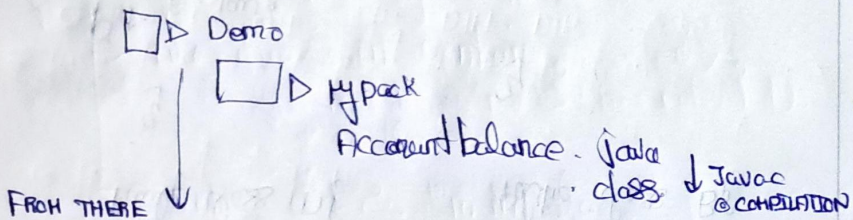
Package mypack;

For a program to find mypack →
- Program can be executed from a dir immediatley above mypack ✓
- classpath must be set to include path to mypack ✓
- -classpath option must specify the path when program runs via java. ✓

In the 2nd 2 methods / option, the class path must NOT include mypack itself.

↳ It should just specify the path to mypack.

⊗ It is a good practise the package contains only non-main classes. Yup....



IMPORTING A PACKAGE

* import entire package

ACCESS PROTECTION FOR PACKAGES

3 main
* Public
* Private
* Protect

all access specifiers are applicable to packages. JAVA addresses 4 category for class members....

① Sub class in same package
② Non-sub class in same package
③ Sub class in diff package
④ class that are either in same package or sub package.

	PRIVATE	NO-MODIFIER	PROTECTED	PUBLIC
Same class	✓	✓	✓	✓
Same pack Subclass	x	✓	✓	✓
Same pack Non-subclass	x	✓	✓	✓
Diff pack Subclass	x	x	✓	✓
Diff pack Non-subclass	x	x	x	✓

DEMONSTRATION - VIII
#LECTURE - A

* all classes are imported when we use "import"

→ Sqrt is a static method.
∴ It can be called without creating a object.

* We can include/import the class what we actually want to.

We can actually access or use classes without importing (OR)

⇒ Java.util.ArrayList Liste = new ...;

import myPackage.myClass;

While creating a package ⇒ all the methods INSIDE Public class SHOULD BE PUBLIC.

* We can also inherit the Class what we imported.

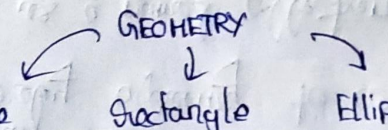
* The classes that shares a common package can extend one another.

! only public is accesible even from different packages.....

INTERFACE - I
#LECTURE - 20

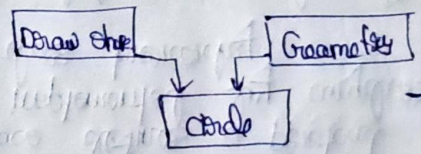
ABSTRACT CLASS

→ abstract is a keyword.
→ abstract class that allow class to create its own object.



* abstract methods ⇒ no body → inherited and written / "override"

MULTIPLE INHERITANCE IN JAVA



This is NOT POSSIBLE in JAVA.

This is possible by ABSTRACT INTERFACE.

INTERFACE

→ Abstract & interface is very similar.
→ It is hard to find diff between them, BUT THE DIFF IS
  → abstract class allows ⇒ single INHERIT.
  → interface ⇒ SUPPORT multiple inheritance.

Interface is basically a kinda class.
  → Like classes it contains,
    * members
    * methods.
  → unlike classes in interface,
    * all members are final. → ∴ Can't be changed
    * all methods are abstract → only changed/used/overrided by inheritance then used.

* For both abstract class & interface NO OBJ. CAN'T BE INSTANTIATED....

INTERFACE CONCEPT

* An interface defines a protocol of behaviours that can be implemented by any class anywhere in the class hierarchy.
* Interface defines set of methods BUT DOES NOT implement them.
* A class that implements the interface agrees to implement all the methods defined in the interface, thereby agreeing to certain behaviour.
* "An interface is a named collection of method definitions, without implementation."
* Interface reserves behaviour for class that implements them.
* "Template/framework for the class"

* All methods in an interface are always PUBLIC & ABSTRACT.
↳ if they are NOT there, then Compile time error happens.

(*) static method can't be declared in interface.
↳ these methods do not / are not never abstract and do not express behaviour of objects.

* Keyword ⇒ INTERFACE

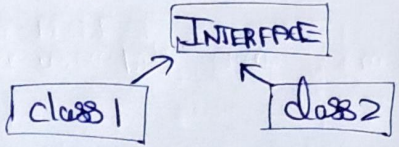
* Interface are syntatically similar to classes, but they lack instance variables & their methods are defined without any body.

```
interface call Me {
    Void call(int p); // no def.
}
```
(*)

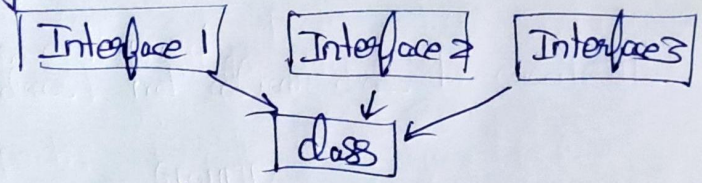
** It is by default abstract and public. **

once an interface is defined,

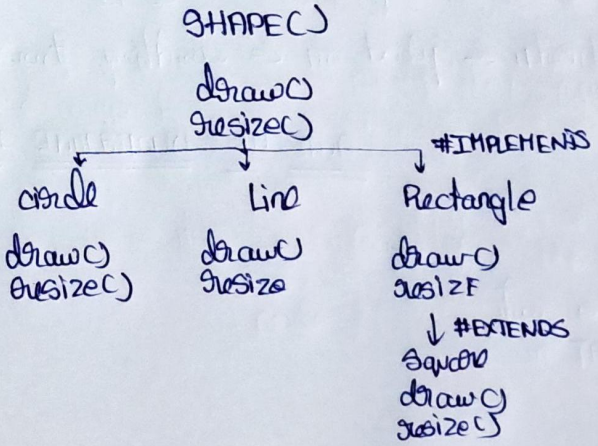
① * any no. of classes can implement an interface.



② * one can also implement any no. of interface using a class



* Interface is a structure / framework / protocol.



PROPERTIES OF INTERFACE

* KEYWORD → Interface
* all methods are implicitly public & abstract.
* It's method can't be static
* It's methods can't be final, since abstract.
* A interface extends one or more interface.
* (*) A interface CANNOT implement another interface or class ---- OBVIOUSL BLUD...
* (*) Interface types can be used POLYMORPHICALLY

Variable declared,

static final <type> <name> = <val>;

method, <ret type> <methodname> (Parameter list);

class <name> implements <interface> {
}.

<Interface name> <name>;