It is the container which is used to store logically related types(classes, interfaces, enums).

Create package:-

package packagename; ← keyword

Advantages:-

→ good readability
→ good maintability (if any error is there we can easily rectify it).
→ to avoid naming conflicts.
→ to control access

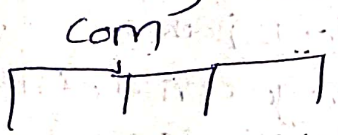
Rules:-

→ you should give reverse of company domain name.
www.techpalle.com
Package name be:-
com.techpalle → if we want anything we can write.
→ Small letters [start with]
→ No spaces
→ No predefined keywords.
→ Package statement should be first line.

Deployment:-

If you create any application like facebook, insta etc, after creating we will definetely upload in server, that uploading the creating ones in server is called deployment.

Avoid naming conflicts



→ If we want to access placement details in kadnest.
→ Then if we mention placements, here it will get conflicts like, from which i have to take. so By using package this conflicts will be reduced.

Accessing members of one package to other package:-

→ By creating object of class along with package name.
→ By importing the particular class from that package.
→ By importing all the classes from that package.

Student

registration.java.

```
Package: com.techpalle.student;
public class Registration.
```

trainers.

subject.java

```
package com.techpalle.trainers;
public class sub {
    public void func() {
```

Packages < built-in ——→ [java, lang, awt, javax, swing, net, io, util, sql, etc]
user-defined

→ import package.*;
import package.classname;] → access package from another package.
fully qualified name

1) using packagename.*

import keyword is used to make the classes and interface of another package accessible to the current package.

```
Package pack;
 Public class A {
 Public void msg() { S.o.P("Hello"); }
Package mypack;
import pack.*;
Class B {
 Public Static void main (String() args) {
  A obj = new A();
  obj.msg();
 }
}
```
o/p:- Hello

2)
```
Package pack;
 Package public class A {
  Public void msg() {
   S.o.P("Hello"); }
Package mypack;
import pack.A;
class B {
public static void main (String args()) {
 A obj = new A();
 obj.msg();
}
}
```

```
Package mypack;
class B {
 P.S. void main(String() args) {
 pack.A obj = new
      pack.A();
 obj.msg();
 }
}
```

3)
```
package pack;
public class A {
 public void msg() {
  S.o.P("Hello"); }
}
```

```
public void func() {
com.techpalle.student.Registration
r = new com.techpalle.student.
        Registration();
r.display();
}
}
```

By importing particular class from the package:-

```
student
registration.java.
package com.techpalle.student;
public class Registration {
  public void display() {
    =
  }
}
```

```
trainers
subject.java.
package com.techpalle.
    trainers;
import com.techpalle.
    trainers.student.
Registration
```

By importing all classes from that package:-

```
student
registration.java.
package com.techpalle.student;
public class Registration {
  public void display() {
    =
  }
}
```

```
trainers:
subject.java.
package com.techpalle.
    trainers;
import com.techpalle.student.*;
}
public class sub {
  public void func() {
  Registration r = new Registration();
  r.display();
  }
}
```

Ambiguity in packages:-

i.e when multiple packages have same name, then class ambiguity happens in 2nd and 3rd approach.
→ At this one 1st one is preferable.

int a = 10 Unicode = 62536

long a = 10 ↓ 16 → public abstract void func();

Abstract method: - a method without body, is called as an abstract.

public abstract void m2(); → abstract method

↓ Here semicolon mandatory.

abstract keyword is mandatory.

→ A class which have atleast one abstract method that class should be declared as abstract class.

```
class sample {
    public void m1() {
        ==
    }
```

→ concrete method

public abstract void m2(); → abstract method

→ we cannot create object for abstract class. → Why we can't create objects for abstract.

→ Then we need to inherit the abstract, implementing all the abstract methods in child class.

```
public class sample {
    public void fun() {
        S.O.P("Hello");
    }
public abstract void m1();
public abstract void m2();
}
```

→ Here it get error, because if in a class, atleast one abstract class is there we have to mention abstract class.

→ Inheriting:-

```
Abstract public class sample {
        public void func() {
            S.O.P("Hello");
        public abstract void m1();
        public abstract void m2();
class childsample extend Sample {
    public void m1() {
        S.O.P("Body of m1"); }
    public void m2() {
        system.out.println("Body of m2"); }
```

Abstract class object:-

```
abstract class Test {
    public void fun1() {
        S.O.println("welcome to palle"); }
    public abstract void for2(); }
public static void main(String[] args) {
        Test t=new Test(); → this will give error
    }                        we can't create object
                             for abstract class.
```

→ abstract classes can be extended/inherited to other classes.

→ all abstract methods present in abstract class must be implemented/given body in derived classes, otherwise derived class must be declared as abstract class.

how to use abstract class?

the only way to use abstract class is by inheriting to a child class. because we can't create object for abstract class.

2-ways:- of inheriting abstract classes:-

providing body/functionality for all inherited abstract methods in the child class (while implementing remove abstract keyword.)

(or)

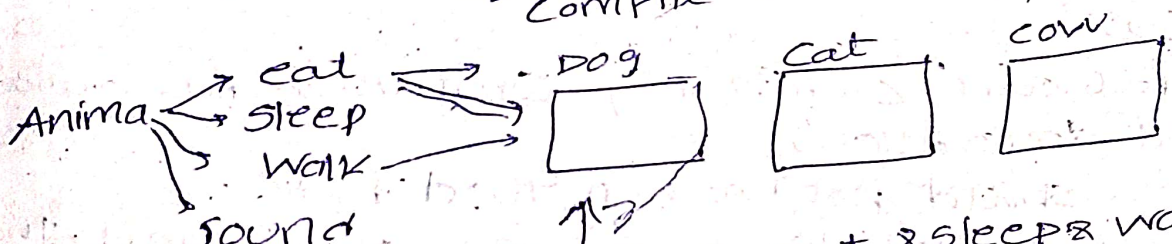
making child class as abstract class, if you can't give body for all the abstract methods.

examples:

① public abstract class A
{
public abstract void f1();
public abstract int f2(int x);
}
public class B extends A {
public void f1() {
S.O.P("Hi");}
public int f2(int x) {
S.O.P("Bye");
return 10;
}
}

public static void main (String[] args) {
B b1 = new B();
b1.f1();
b1.f2(4);

② public abstract class A {
public abstract void f1();
public abstract int f2(int x);
}
public abstract class B extends A {
public void f1() {
S.O.P("Hi");}}
public static void main(String[] args) {
B b1 = new B(); }
→ compile time error.



so here we have to create concrete methods for eat, sleep, walk. and we have to create abstract method for sound.

Here eat & sleep & walk are here both animals by mouth closing eyes, by legs. But sound will be different → The method that which are same for all the child class we have to create it by concrete method.
→ which is different for that we have to create abstract Method.

```
abstract class Animal {
    public void eat() {
        S.o.p("eat with mouth"); }
    public void walk() {
        S.o.p("walks with legs"); }
    public void sleep() {
        S.o.p("sleeps by closing eyes"); }
    public abstract void makesound();
}
class Dog extends animal {
    public void makesound() {
        S.o.println("Bow Bow"); }
class Cat extends Animal {
    public void makesound() {
        S.o.p("meow meow"); } }
abstract class Giraffe extends Animal {
    // Here I don't know the how giraffe make
        sound so i make this is abstract [that
        someone [developer] they will execute].
```

Interface:-

Interface looks exactly like a class with below properties..

Properties:-

→ Interface can contain only constants/final variable and method declarations.

→ Interface should not have method bodies.

→ by default all methods in interface are abstract. that why method bodies are not allowed in the interface.

→ you can't create object to interface (because abstract method is there).

```
public interface MyInterface {
    public final int j=10;
    public void f();
}
```

Interface

→ In interface by default every method is default.
→ All members of interface by default it is public.
→ It will follow pascal case convention.

can I create final abstract method:-?
No, Because we will fill modify this in child class, so not use final in abstract. → or access

can I create static abstract method?
yes.

can I override abstract?
yes

can I overload abstract method?
→ No, it is not possible. overload the abstract method in same class because it does not have implementation

→ Interface will always create outside the class
→ we are not able to create objects for interfaces and abstract, because abstract method is incomplete, if we create the memory will be wasted.
→ If we want use the interface, then we have to implement all this in one class [we are not able to create objects and access].
→ using implements keyword [we can interface in a class].
→ In a one program we can create only one public class, one public interface, and one ~~et~~ public enum.

```
public interface I {
    public final int x=10;
    public abstract void m1();
}
class A implements I {
    public void m1() {
        S.o.P("Implemented method of interface");
    }
}
```

→ you have to implement all methods in class OR → use abstract class. [a

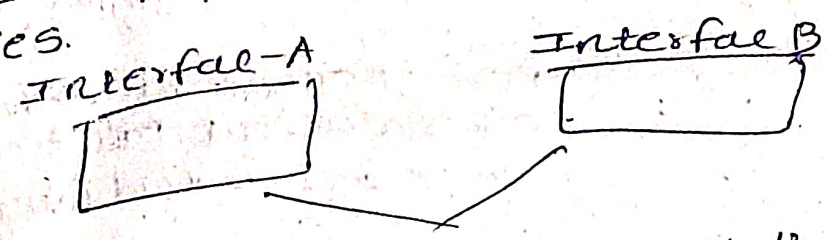
OR

```
abstract class A implement I
```

```
package abstraction;
public class Mainclass {
    public static void main(String[] args) {
        A a1 = new A();  a1.m1();
    }
}
```

Can I implement multiple interfaces?

yes.



Class I implement A, B

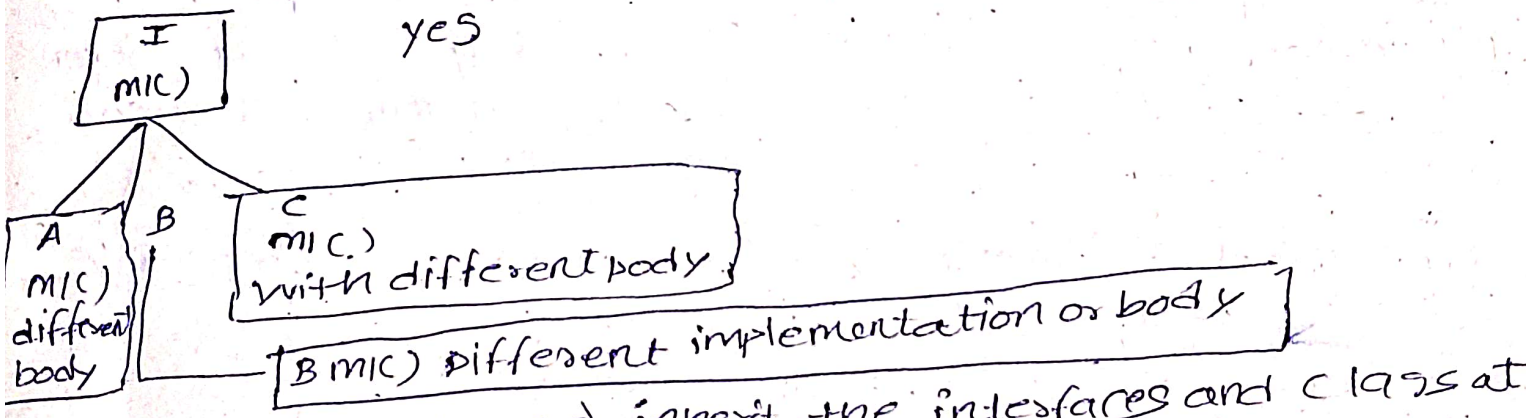
```
interface A {
public abstract void fun1();
}
interface B {
  public abstract void fun1();
}
class C implements A, B {
  public void fun1() {
    S.o.p("fun1");
  }
}
```

→ No function ambiguity in these multiple interface, because method overriding happens in child class.

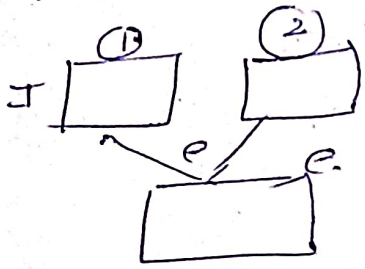
→ Here body is not mentioned in interface,

→ Body is implemented in class only so function ambiguity not occurs.

Can I implement one interface in multiple class?



Can I implement and inherit the interfaces and class at a time?




```
class child extends parent
    implements I. {
}
```

Uses of interface:

- Behaving as a API [Application programming interface]
- → when we have dependency [i.e when 2 or 3 more developers dependency on same code].

```
m1.fun1();
String i = m1.fun2(); S.o.p(i);
```

Tap academy

(2) when we have a dependency
i.e when we consider 2 employees, if we 1-employee is fast and other one not done the work
→ But here both works depend on solution.
→ If one done and can't wait, then here 1st one will create interfaces of and one also and complete the task
→ So here 2nd person will create the code based on the interface which will provided by 1st one.

→ In both this cases a interface is most preferable.

Palle technologies

Here both want to find calculator: But palle technolog is very fast it find calculato But tap academy is very lazy it can't find calculator, bu these both are friends, here palle technologies and tap academy both are friends, so it's not possible to [here tap academy asked to share code for calculator], so actually they are friends and can't say no so palle technologies will give code [i.e here complete code we can't give]. so here we are sharing API, this will use, and here, this will give to tap academy [i.e here palle technolog -es will give only interface, depen on that tap academy will create and execute.

Can I ...

yes

Inheriting & Implementing at a time:

a class can inherit (extend) only one class.
a class can implement multiple interfaces at a time.
a class can inherit/extend a class & implement interfaces at a time.

```
Class A {          interface one {          interface two {
}                  }                        }

Class B extends A implements one, two {
    Implement one & two
}
```

Dif b/w abstract classes & interfaces

Abstract class	Interface
you can have methods with body + abstract methods + and any type of variables.	you can have only - constant/final variable - only method declarations
abstract class members can be private/protected/ default/public.	every member inside interface should be public.
If you want to share some common functionality to a family of logically related classes & wanted to enforce child classes to implement some specific functionality then use abstract class.	if you want to share some common functionality to some logically unrelated classes. then use interface.

we mainly use 2 ways
→ when we don't know the row do exceuting.

→ we can create static method in interface.
→ implements by giving body to the method. we can't implement one interface in other interface.
→ In extern and implement we will give first priority to extern.

```
public static void main(String[] args){
    System.out.println("Hi");
    try{
        int res=10/0;
        S.O.P(res);
        S.O.P("Hi");  → this will not execute
    Catch(Exception name any variable e){
        ArithmaticException ae){
        S.O.P("we can't divide any number with zero");
    }
```

→ o/p:- Hi
we can't divide any number with zero

→ If exception occurs in try, and exception name does not match with catch block, the JVM terminate program abruptly. And remaining lines present after catch block will not be executed further

```
public static void main(String[] arg){
    S.O.P("Hi")
    try:
        int[] arr={10,20};
        S.O.P(
```