Dynamic Dispatch:

Dynamic dispatch is the process of assigning subclass object to superclass reference variable.

```
Base b = new Derived();
```

↑ Base class reference

↑ derived class object

→ Hiding the internal implementation and just highlight the set of services or set of functionalities what are offering is the concept of Abstraction.

→ i.e showing only essential feature to user rather than internal implementation.

→ Ex: ATM machine GUI, online application, desktop application,

Abstraction is achieved in two ways

1. Abstract class (0 to 100%)
2. Interface (100% abstraction).

Advantages

1. Security
2. Enhancement become easy
3. maintainability of an application is improved
4. easiness to use system/application.

Abstract keyword (Abstract modifier)

- abstract is the modifier applicable for classes and methods but not for variable
- class defined without abstract modifier are called as concrete class. concrete class can be instantiated.
- classes defined with abstract modifier class are called as abstract class and are not instantiated.
- method defined without abstract modifier are called as concrete method. concrete method must have body (implementation)
- method defined with abstract are called as abstract method. Abstract method must not have body (implementation)

Abstract method

- method with abstract modifier is called abstract method.
- abstract method should not have implementation.

Syntax:

```
abstract void methodName();
```

- Child classes should implement the abstract method
- abstract with → final, static, synchronized, native, strictfp, private → not allowed

abstract class : Partial implementation of class

- If you don't want to create an object / instanciation then we have to declare that class as abstract.
- abstract class can contain concrete or abstract classes or both

Syntax:

```
abstract class abstractDemo {

}
```

```
abstractDemo ad = new abstractDemo(); X
                                  CTE
```

- If a class contains atleast one abstract method then compulsary we have to make class as abstract class else CTE
- Eventhough class doesn't contain any abstract method still we can make class as abstract class
- abstract method should not have final, combination is not valid, as abstract method should be implemented (body has to be write in derived class) in derived class.
- abstract class can contain final method, But final class can't contain abstract method.

- Abstract class can have constructor
- To use abstract method of class, we must extend abstract class with concrete class and method must be Provided with body in concrete class.
- Multiple inheritance are not allowed in abstract class but allowed in interfaces.
- why does the abstract class have a constructor even though you can't create object?

Ans: Abstract class can only be extended by class and that class will have its own constructor and that constructor will have super(); which is calling constructor of abstract class, if the abstract class doesn't have constructor then the error flashes.

Therefore, abstract classes have constructors.

- adapter class is one which is defined with dummy methods implementation.
- abstract class can extend concrete class.
- abstract class can contain static methods.
- abstract class can contain main method
- abstract class contains Both instance and static variables.
- static members of abstract class can be accessed with classname. Instance member are accessed with subclass object only.

Full

- Abstract class contains all the members that can be written in Concrete class.

Example (1)

```
abstract class Animal {
    abstract void sound of Animal ();  // it is just idea.
}
class cat extends Animal {
    void sound of Animal ()
    {
        SOP ("meow");
        // implementation of idea as per subclass requirement
    }
}
class Dog extends Animal {
    void sound of Animal ()
    {
        SOP ("Bow Bow");
        // implementation of idea as per subclass requirement
    }
}
```

```
Animal a;
a = new cat();
a.sound of Animal();
a = new Dog();
a.sound of Animal();
```

Example (2)

```
abstract class shape {
    abstract double area();
}
class Rectangle extends shape {
    double length = 45.65;
    double width = 48.78;
    public double area() {
        return (length * width);
    }
}
class Triangle extends shape {
    double base = 5.2;
    double height = 7.4;
    public double area() {
        return (0.5 * base * height);
    }
}
```

Triangle array

```
shape s;
s = new Rectangle();
double d = s.area();
sop(d);
```

```
s = new Triangle();
double t = s.area();
sop(t);
```

Example (3)

```
abstract class Bank {
    abstract int getRateOfInterest();
}
class SBI extends Bank {
    int getRateOfInterest() {
        return 7;
    }
}
class PNB extends Bank {
    int getRateOfInterest() {
        return 8;
    }
}
class TestBank {
    psvm(-)
    {
        Bank b;
        b = new SBI();
SOP(b.getRateOfInterest());   } SBI
        b = new PNB();
SOP(b.getRateOfInterest());   } PNB
    }
}
```

Interface

- 100% pure abstract class is nothing but an Interface or any service requirement specification is nothing but Interface. (SRS)
- Interface considered as contract b/w client and service Provider.
- Every method present in the Interface should be abstract method So it is called as 100% Pure abstract class.

Syntax:

```
interface <InterfaceName>
{
    void m1();
    Void m2();
}
```

// By default Public abstract

- In industry, Architect level People create interfaces and then it is given to the developers for writing classes by implementing interfaces provided.

- **Features of Interface:-**
  - * We can not instantiate Interface but we can create reference variable.
  - * By default all methods in Interface are Public abstract and all variables are static final Public
  - * interface does not have static block, main method, instance block, Constructor, instance variables,

- we can't create object for Interface.
- Interface not allowed with concrete methods.
- Interface doesn't contain constructor as it contains static final variables and not contains instance variable.
- Interface must be implemented by using 'implements' keyword. the implemented class will provide the implementation for all the abstract methods in the subclass.
- when you are accessing interface members by interface reference variable which contains subclass object then that members should be aviable in interface.
- Interface variables can be accessed with Interface name also because of static.
- Interface can extend interface. (any number)
- Interface can extend Multiple interface.
- ~~Interface can extend~~
  - class extends class implements Interface.
  - class implements Interface.

- class extends class implements Interface1 and Interface2.
- when you are implementing two or more interfaces which contains common methods then in subclass you have to provide the implementation only once and single implementation will be valid for all interfaces.
- what about different return type? impossible.
- If two interfaces have same variable names while using it follow the syntax:

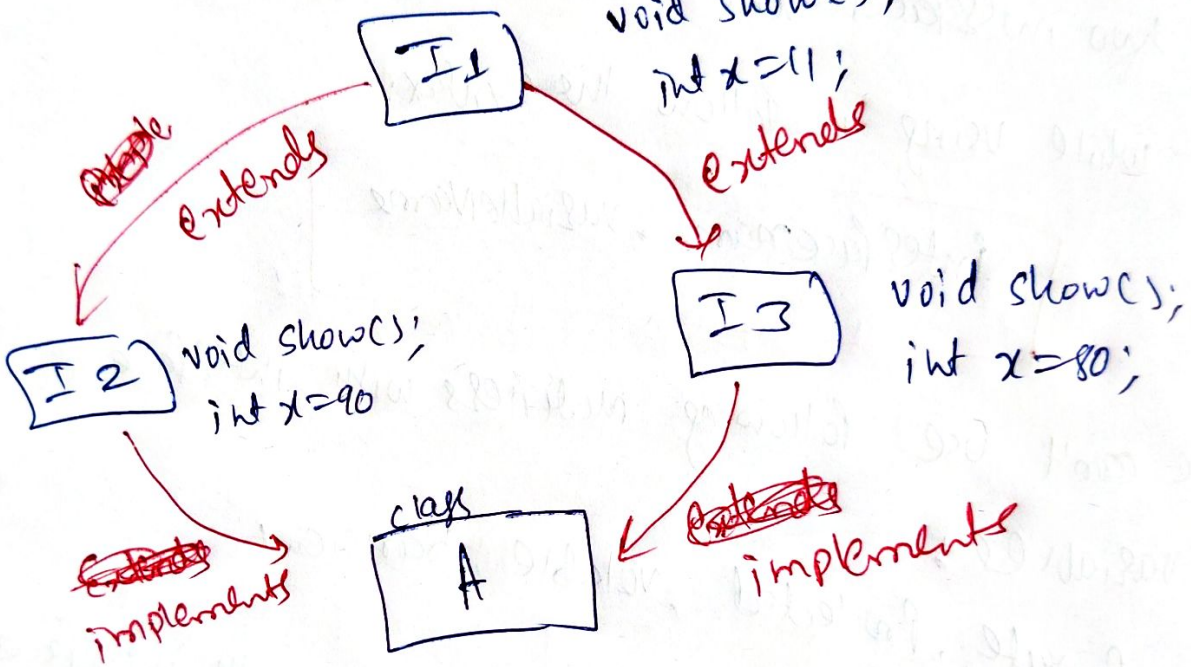
```
interfacename.variableName
```

- we can't use following modifiers with interface variables:
  private, protected, volatile, transient
- we can't use following modifiers with interface methods:
  private, protected, static, final, strictfp, synchronized, native.
- we can't define following members in Interface:
  1. instance variable
  2. static method
  3. constructors
  4. initialization Block.

- when you define an interface without any members then it is called marker interface (or) Tagged interface

  Ex: Serializable, Clonable, RandomAccess, SingleThreadModel, Remote

- Multiple Inheritance using Interface




```
void show() {
// super.show();  ← can't be used for Interface
    I1.x;
    I2.x;
    I3.x;
}
```

} Because of static variable we can access with interface name.

```
A() {
    super();
}
```

} Interface don't have constructor

Interface	Abstract
① don't know about implementation just requirement specification we have Interface	If we have Partial implementation of class, we have Abstract class
② Every method is Public and abstract. (100% Pure abstract class)	Every method present in Abstract class need not be Public and abstract & have concreate methods also
③ we don't use following modifiers with methods i.e private, Protected, default, final, static, Synchronized, Native, Strictfp	No restriction method modifiers.
④ Every var is Public, static final we declare ⊗ not.	need not be Public static final
⑤ private, Protected, transient, volatile are not allowed for variables	No restriction on var modifiers
⑥ should perform initialization at the time of declaration only	No need at the time of declaration can be done later also
⑦ Static Block & Instance Block not allowed main	Allowed
⑧ Constructor not allowed	Constructor are allowed
⑨ supports multiple Inheritance	doesn't ✗
⑩ After implementation object creation is faster	After implementation object creation slower as with Interface.