4__10-07-2022

Sunday, 10 July 2022 10:30 PM

Topics to cover –

→ Inheritance

→ Types of inheritance

→ Interface

→ Default methods.

Inheritance

class — member variables ✓
      └ member methods ✓

Parent class | super class | Base class

↓

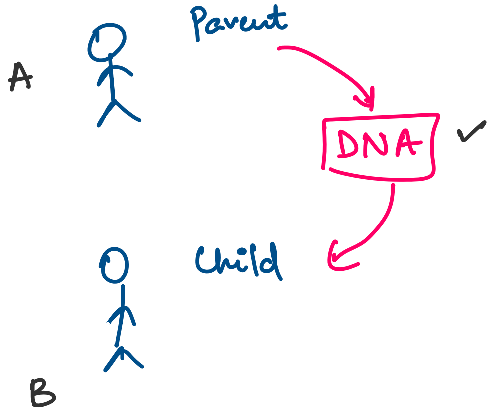
Child class | Sub class | Derieved

OOP — State, Behavior

— Relationships

↓

Inheritance



user A is a parent of user B

Eg- A is a parent of B

Dog is an animal.

Inheritance

Extends

↓

(Classes)

↓

Implements

↓

(Interface)

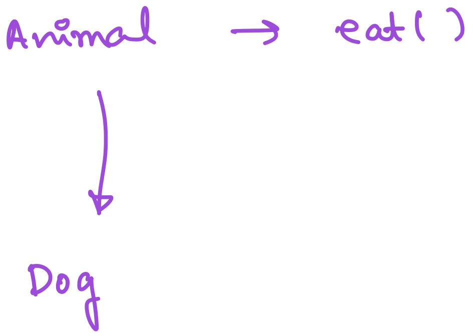
↓

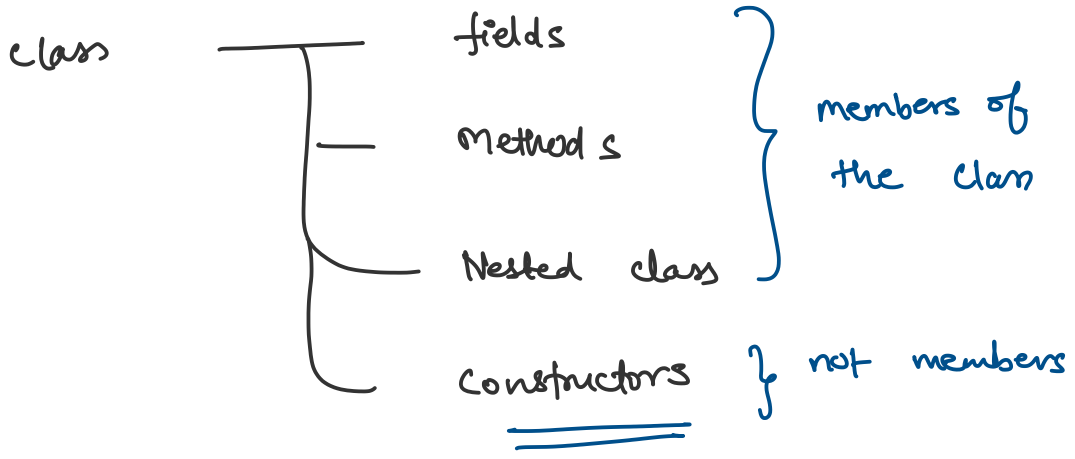
Parent is a class

Parent is an interface

```
public class Animal {
   public void eat()
   {
      System.out.println("Eating food");
   }
}


public class Dog extends Animal {
   /*
   Dog is an animal
   Dog is a child class of Animal, it will get all the stuff of Animal class
   * */
}


public class Program {
   public static void main(String[] args) {
      Dog d = new Dog();
      d.eat();
   }
}
```

Output:
Eating food



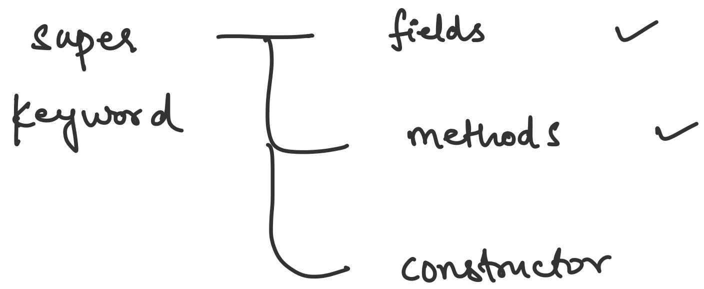


class

- fields
- methods
- Nested class

} members of the class

- constructors } not members



super keyword

- fields ✓
- methods ✓
- constructor

For variable

```
super. variableName;
```

For methods

```
super. methodName();
```

For constructors

Default

constructors —
- Parameterized
- Copy
- Private

Default constructor

↳ This will invoked with the base class constructor

Parameterized constructor

↳ For this we will have to use super Keyword.

```
class A
{
  public A (int num)
   {
       =

   }
}
```

✓

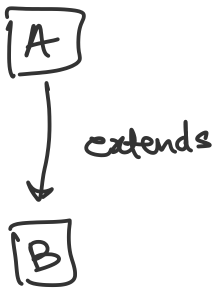
```
 class B extends A
```

```
{
    public B ( int value)
    {
        =
    }
}
```

To invoke parent class constructor

super (5);

single level inheritance



//default class constructor

```
public class Parent {
   public Parent()
   {
      System.out.println("Parent class constructor");
   }
}

public class Child extends Parent {
   public Child()
   {
      System.out.println("Child class constructor");
   }
}

public class Program {
   public static void main(String[] args) {
      Child obj = new Child();
   }
}
```

```
Output:
Default ctor : Parent class
Default ctor : Child class
```

```
public class Parent {
   public void Print()
   {
      System.out.println("Parent class print function");
   }
}


public class Child extends Parent {
   public void Print()
   {
      System.out.println("Child class print function");
   }
}

public class Program {
   public static void main(String[] args) {
      Parent parentObj = new Parent();
      parentObj.Print();

      Child childObj = new Child();
      childObj.Print();
   }
}
```

```
Output:
Parent class print function
Child class print function
```

```
public class Parent {
   public void Print()
   {
      System.out.println("Parent class print function");
   }
}


public class Child extends Parent {
   public void Print()
   {
      super.Print();
      System.out.println("Child class print function");
   }
}


public class Program {
   public static void main(String[] args) {
      Child childObj = new Child();
      childObj.Print();
   }
}
```

```
Output:
Parent class print function
Child class print function
```

```
public class Parent {
   public Parent()
   {
      System.out.println("Default ctor");
   }

   public Parent(int value)
   {
        System.out.println("Parent class ctor : " + value);
    }
}
```

```
public class Child extends Parent {
    public Child(int num)
    {
        System.out.println("Child class ctor : " + num);
    }
}
```

```
public class Program {
    public static void main(String[] args) {
        Child childObj = new Child(10);
    }
}
```

Output:

```
Default ctor
Child class ctor : 10
```

```
public class Parent {
    public Parent()
    {
        System.out.println("Default ctor");
    }

    public Parent(int value)
    {
        System.out.println("Parent class ctor : " + value);
    }
}
```

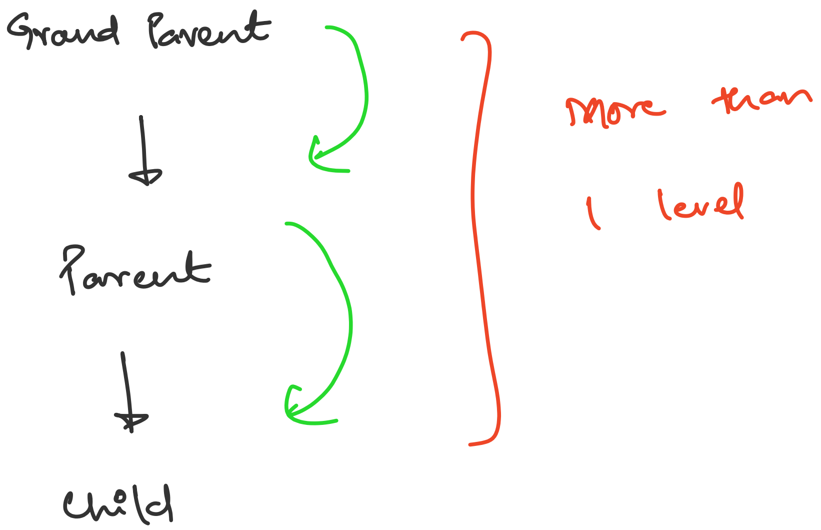
```
public class Child extends Parent {
    public Child(int num)
    {
        super(20);
        System.out.println("Child class ctor : " + num);
    }
}


public class Program {
   public static void main(String[] args) {
      Child childObj = new Child(10);
   }
}
```

```
Output:
Parent class ctor : 20
Child class ctor : 10
```

```
public class Parent {
   public Parent()
   {
      System.out.println("Default ctor : Parent class");
   }

   public Parent(int value)
   {
      System.out.println("Parent class ctor : " + value);
   }
}


public class Child extends Parent {
   public Child()
   {
      System.out.println("Default ctor : Child class");
   }

   public Child(int num)
   {
      super(20);
      System.out.println("Child class ctor : " + num);
   }
}


public class Program {
    public static void main(String[] args) {
        Child obj = new Child();
        Child childObj = new Child(10);
    }
}
```

```
Output:
Default ctor : Parent class
Default ctor : Child class
Parent class ctor : 20
Child class ctor : 10
```

Multilevel inheritance



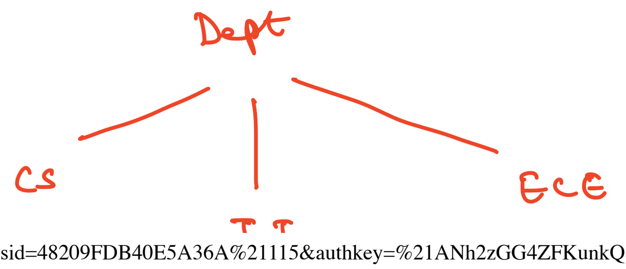
```
public class GrandParent {
    public GrandParent()
    {
        System.out.println("Default ctor : Grandparent class");
    }
}
```

```
public class Parent extends GrandParent {
   public Parent()
   {
      System.out.println("Default ctor : Parent class");
   }
}


public class Child extends Parent {
   public Child()
   {
      System.out.println("Default ctor : Child class");
   }
}


public class Program {
   public static void main(String[] args) {
      Child obj = new Child();
   }
}
```

Output:

```
Default ctor : Grandparent class
Default ctor : Parent class
Default ctor : Child class
```

Hierarchical inheritance

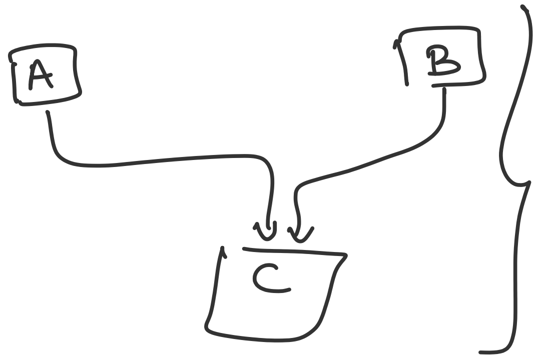


```
public class Dept {
   public Dept()
   {
      System.out.println("Dept ctor");
   }
}

public class CS extends Dept {
   public CS()
   {
      System.out.println("Computer science");
   }
}


public class ECE extends Dept {
   public ECE()
   {
      System.out.println("Electronics");
   }
}


public class Program {
   public static void main(String[] args) {
      CS cs = new CS();
      System.out.println();
      ECE ece = new ECE();
   }
}
```

Output:

```
Dept ctor
Computer science

Dept ctor
Electronics
```

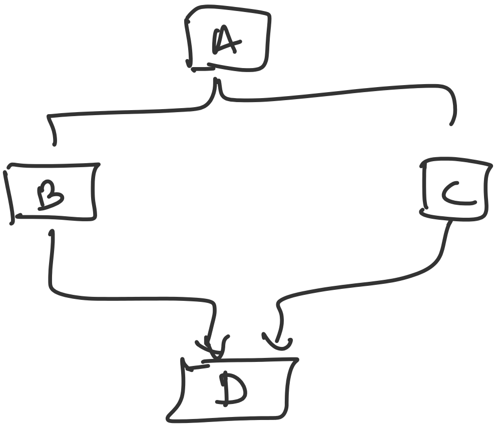
Multiple inheritance



Java does not support this

Hybrid inheritance

Hier + Multiple → Not supported by Java



Java does not allow multiple inheritance where one class inherits from more than one class.

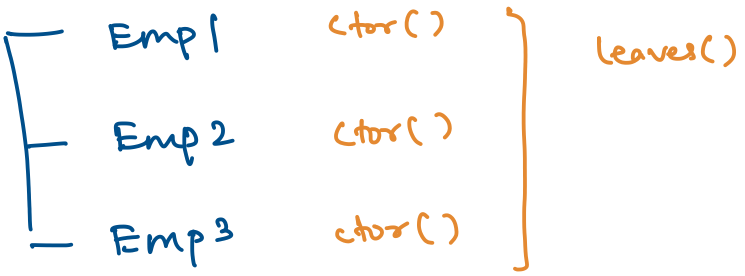
This is also known as diamond

problem.

Sol^n for diamond problem —

Use default methods and interfaces.

Abstract



```
abstract   class   className
{
    methods
    variables
}
```

Abstract class → F^n declaration

→ F^n definition

```
abstract class A
{
    void print();          → f^n declaration

    void show()            } f^n definition
    {
        =
    }
}


class B extends A
{
        =
}
```

For those functions which have only declaration in the abstract class, we will be providing definition in the derived class.

Purely abstract class

↓

If the abstract only has the f^n definition then we call it as purely abstract class.

In such a scenario it will behave similar to interface.

```
abstract class A
{
    public void func1();
    public void fun2();
}
```

Purely abstract class

Interface

↳ using "implements" Keyword.

↳ Interfaces came to solve the

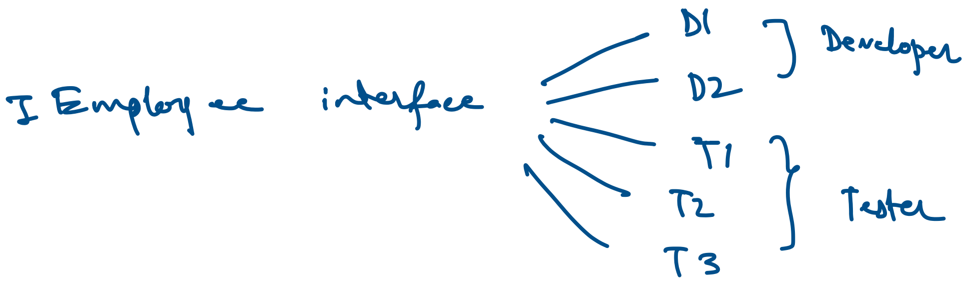
diamond problem.

```
interface IA
{
    =
 }

interface IB
 {
      =
 }

public class C implements IA, IB
 {
     =
 }
```

We can implement more than interface in our class.

Abstract class	Interfaces
① It can have both- f^n definition and f^n declaration.	It can have only f^n declaration.
② It can have constructor and destructors	It cannot have.
③ we can have different access modifiers	we have only public access modifies.
④ we can extend only one abstract class.	We can implement more than one interface.
⑤ It does not solve diamond problem.	It does solve.
⑥ Here we can have constants, fields,	It can have only methods.

methods, getters, setters, etc.	
(7) It can be fully, partially or not implemented in the derieved class.	It has to be fully implemented.



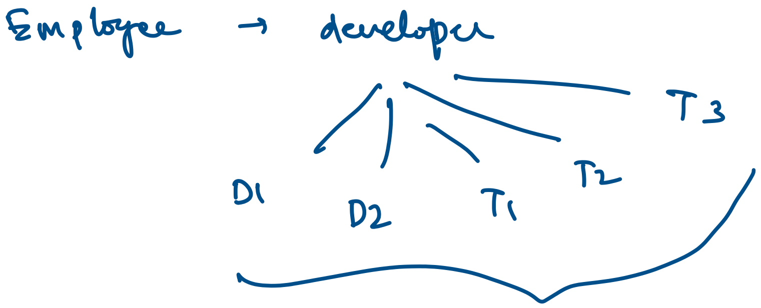
class Employee implements I Employee

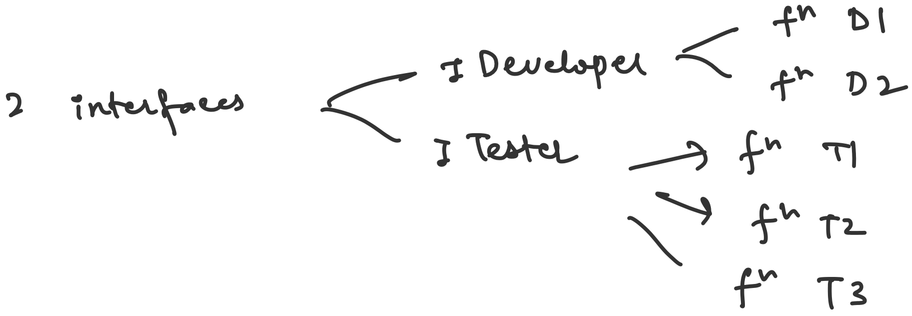
{

You will be forced to provide implementation for

}

all of those 5 f^n, even if the employee is a developer.





```
class DevEmployee implements IDeveloper
{
        Here you will provide
        implementation only for D1
        and D2
}
```

class TesterEmployee implements ITester

{

—

—

—

}

Provide implementation only for T1, T2, T3

```
public interface IDemo1 {
   public void Print();
}


public interface IDemo2 {
   public void Display();
}


public class Demo implements IDemo1, IDemo2 {
   @Override
   public void Print() {
      System.out.println("Print function");
   }

   @Override
   public void Display() {
      System.out.println("Display function");
   }
}


public class Program {
   public static void main(String[] args) {
     Demo d = new Demo();
     d.Print();
     d.Display();
   }
}
```

Output:
Print function
Display function

```
public interface IDemo1 {
   public void Print();
}


public interface IDemo2 {
   public void Print();
}


public class Demo implements IDemo1, IDemo2 {
   @Override
   public void Print() {
      System.out.println("Print function");
   }
}

public class Program {
   public static void main(String[] args) {
     Demo d = new Demo();
     d.Print();
   }
}
```

Output:
Print function

```
public interface IDemo1 {
   default void Print()
   {
      System.out.println("Demo 1 print");
   }
}

public interface IDemo2 {
   default void Print()
   {
      System.out.println("Demo 2 print");
   }
}


public class Demo implements IDemo1, IDemo2 {
   @Override
   public void Print() {
      IDemo1.super.Print();
      IDemo2.super.Print();
   }
}


public class Program {
   public static void main(String[] args) {
     Demo d = new Demo();
     d.Print();
   }
}
```

Output:

```
Demo 1 print
Demo 2 print
```