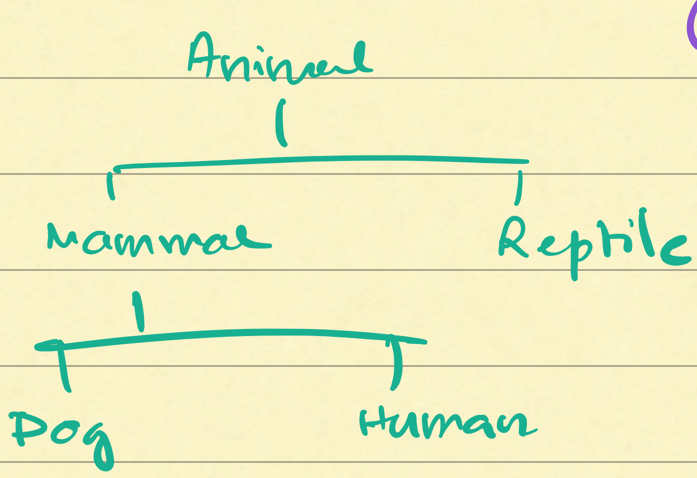
Agenda

- Interfaces
- Abstract classes
- Final & Static keyword.

Interfaces :

Classes / Inheritence → Categorise on the basis of physical / logical similarity.



Animal Machine

Dog Robotic Dog.

(can run) (can run)

A race is organised → you want to get all entities that can run

Runner
```
void participateRace(List< ~~?~~ > runners);
{
    runner.run();
}
```

Animal? → Robotic Dog miss out

Dog? — " + Other animals who can run

Robotic Dog?

Machine?

We also need to categorise based on behaviours

↳ Interface.

Interface

```
Interface Runner {

    void run ( ) ;      → there is no
                          defination
}
```

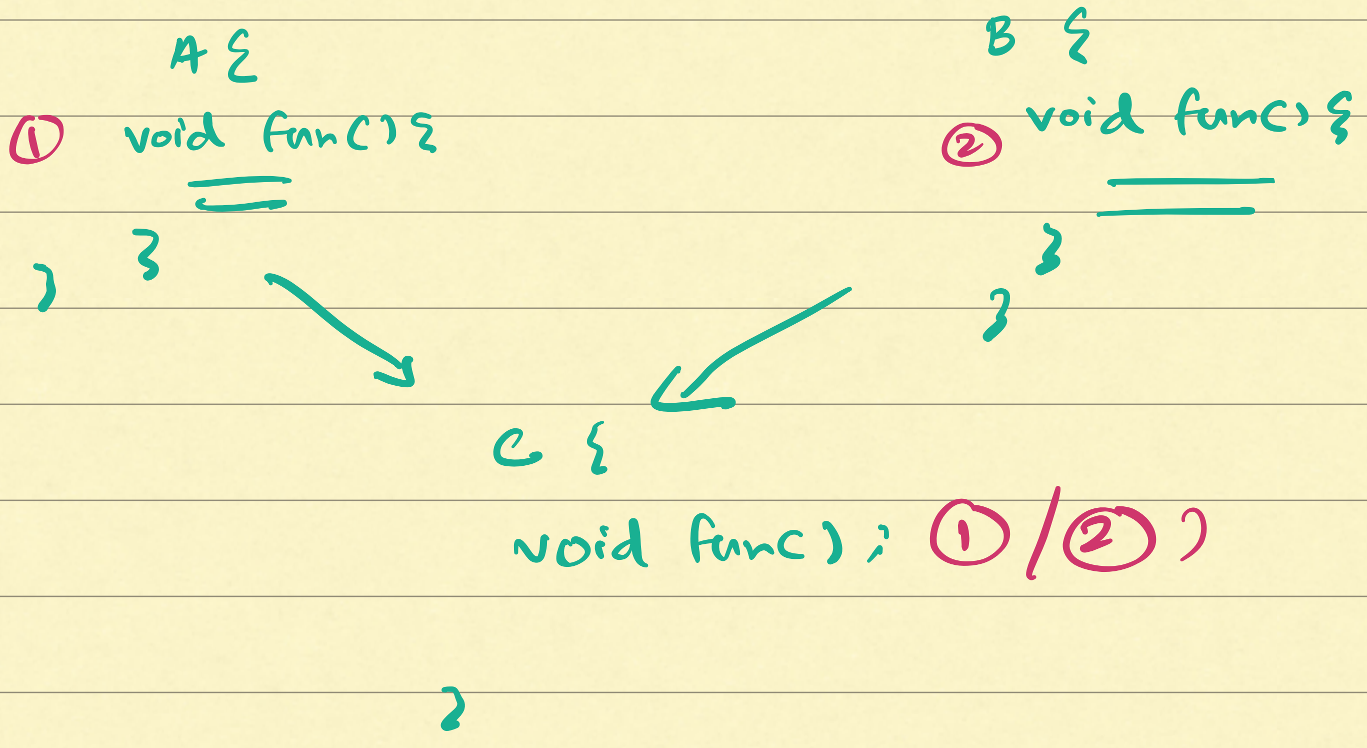
```
void fun();
    ↓
there is no
defination

void func(){
    ═══
};
    ↓
Defination &
    declaration.
```

```
Class Dog implements Runner{
    // It's mandatory to give defination to
    the methods of the interface.

    void run ( ) {
        ───────────
        ───────────
        ───────────
    }
}

Class RoboticDog implements Runner {

    void run () {
        ───────────
        ───────────
        ───────────
    }
}
```

An interface can also act as a reference data type.

```
Runner r = new Dog();
Runner r = new RoboticDog();

Dog d = new Runner(); X
r.run();
```

- Multiple inheritence is not possible in java but you can implement multiple interfaces.



```
A {
①  void fun(){

    }
}

B {
②  void func(){

    }
}

C {
    void func(); ①/②)

}
```

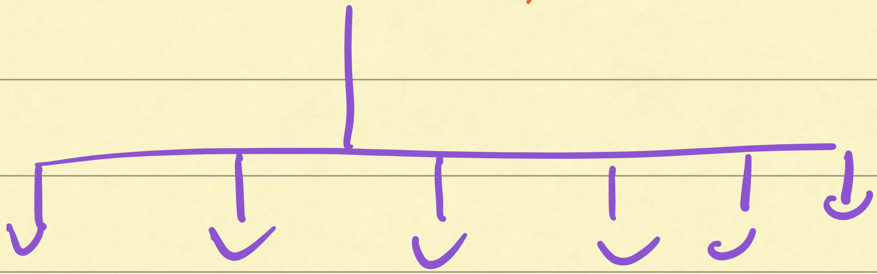
```
<<A>> {                      <<B>> {
  void fun();                  void func();
}                            }
```

implements both

```
C implements A, B {
  void fun() {
  
  }
}
```

```
Interface Work {
  void focus();
  void prepare();
}
```

→ adding new function



50 classes

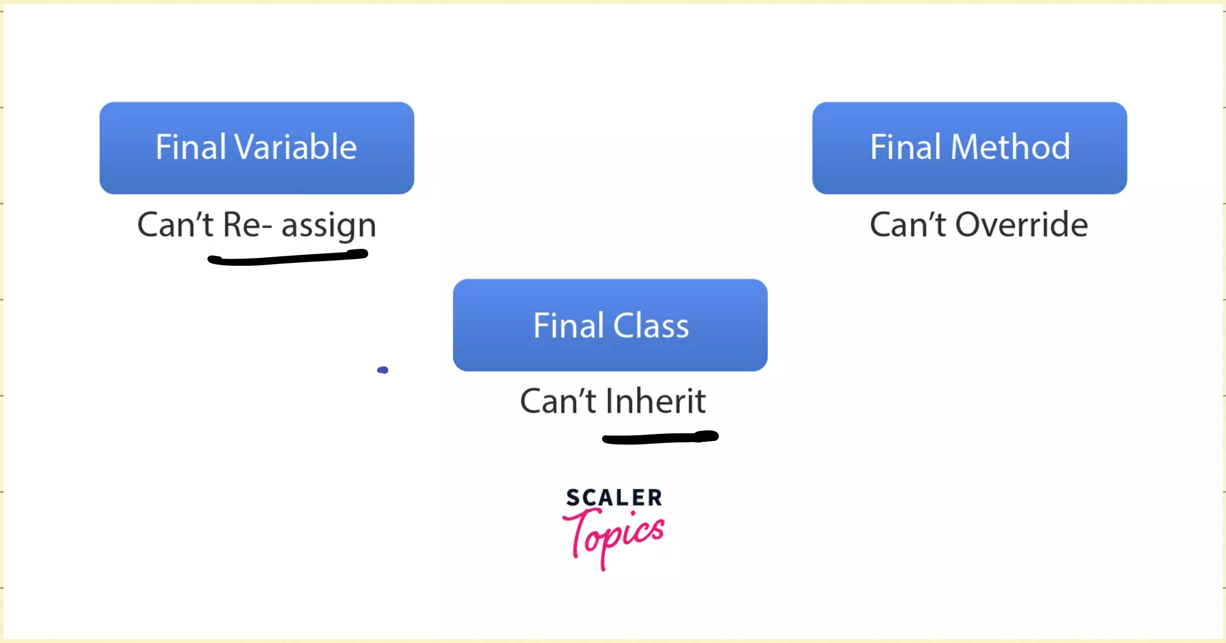
```
① Interface WorkPlus extends Work {
        void prepare;


    Class A implements WorkPlus {
        void focus() {
            =
        }
        void prepare() {
            =
        }
    }
}
```

② Default methods

```
interface Work {
    void focus();
default void prepare() {      → default methods
    ≡                           must be defined.
}
}

    Class A implements Work {
        void prepare() {   } — Can define
            ≡                but even if
                             you don't
```

} }   no errors.

final keyword :



```
final int x = 10 ;
final int x ;  }
   x = 10 ;    }
  x = 12; X
```

```
class A {
   final void something(){

   }
}
class B extends A{
    void something(){
         X
  }  }
```

```
final Class A {

}
```

```
class B extends A {  X

}  X
```

Can't extend a final class.

Break till : 8:07.

PhonePe → YesBank (RBI banned YesBank)

↳ ICICI Bank

How much time ? 24 hours

Always code to interface and not classes.

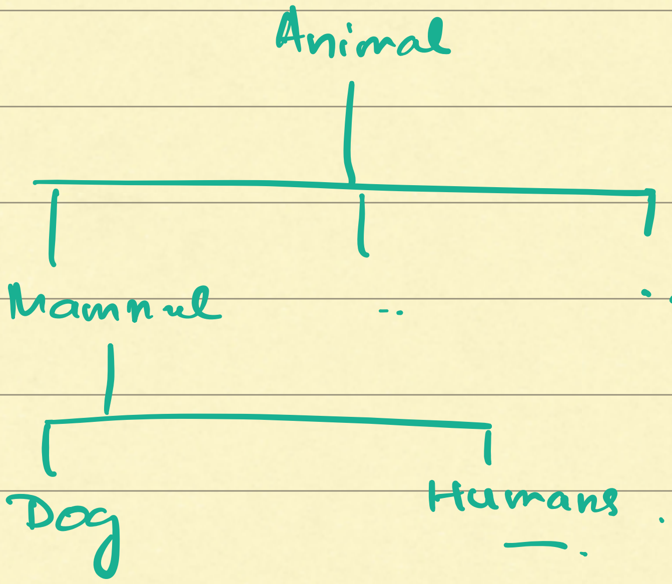
```
UnifiedInterface upi = new ____ ();

upi.__
upi.__
```

sully → Captain sullen berg
↳ water landing on Hudson river.

You can achieve anything if you are not in a hurry.

Abstract Classes.

```
class Animal {
  void move(){
     ===  ✗        ← don't wanna define
  }
  String type;
  int weight;
  int colour;

  void checkDeadAlive(){
     ===
  }
}
```



Animal a = new Dog();
new Lion();
new Animal()

```
abstract class Animal {

    abstract void move();   → no defination
                              only declaration

}
    Animal a = new Animal(); ✗
        a. move() ;
```

You can extend abstract class

```
class Dog extends Animal {
 // Mandatorily define abstract methods
  of parentclass
      void move() {
}         }

abstract class Maman extends Animals {

    }
```

Abstract class can exist without an abstract method

but abstract method can only be written in a abstract class

```
void    checkAllAnimalAlive(List<Animal>
                                          animals)
    { for(Animal a : animals){
                  a.checkDeadAlive();
             }
        }
)
```