→ Interfaces

→ Abstract classes.

Object → state
→ behaviours.

Inheritance

Animal

Dog Cat

logically connected to each other. ↓

is a relation

Dog
↓
Run()

RoboticDog
↓
Run()

Organise a race

List < > — ;
Animal
Dog
RoboticDog.

Interface → to group entities based on behaviours.

→ Categorize on the basis of behaviours.

Abstract fn → A function which doesn't have a defn.

```
Interface Runner {
    void run();
}
```

mandatory to give defn.

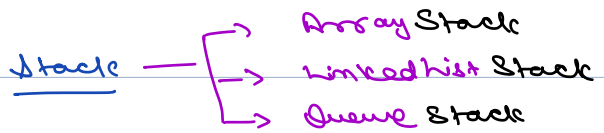
```
Class Dog implements Runner {
    void run() {
        "Dog is running";
    }
}
```

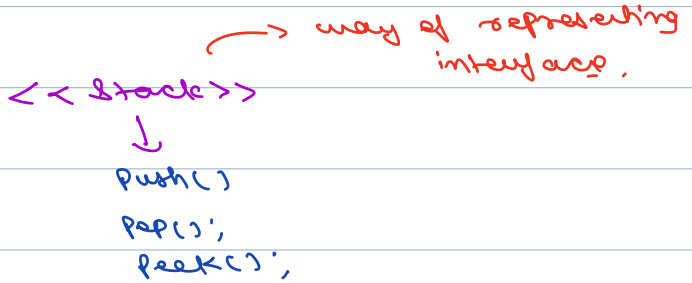
```
Class RoboticDog implements Runner {
    void run() {
        "Robotic Dog is running";
    }
}
```

Interfaces as data types:

```
Runner r1 = new Dog();
Runner r2 = new RoboticDog();
r1.run();
r2.run();
```

Everything is public in interfaces.






```
Stack s = new ArrayStack();

void doSomething (Stack s) {
    s.push();
}
```

PhonePe ← yesBankAPI

↳ getBalance
moneyTransfer

ICICI
~~yesBankAPI~~ yb = new ~~yesBankAPI~~();

ICICIAPI();

doSome (ICICI yb) {
balanceCheck
yb.getBalance();
}

ICICI
↓
balanceCheck();
transferMoney();

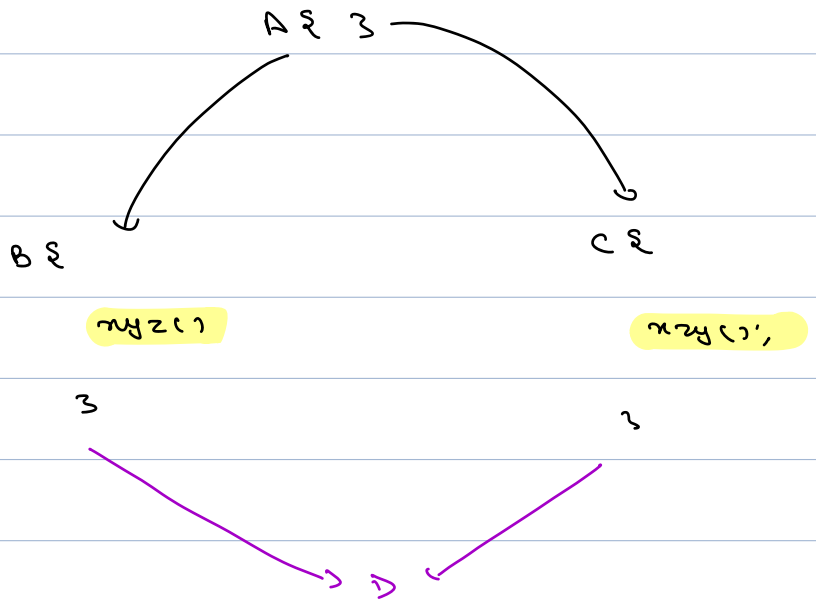
<< bankAPI >>
↳ getBalance();
↳ transferMoney();

yesBankAPI

ICICIUPI()

→ Code to interface not implementation.

—> class can implement many interfaces but they can extend only 1 class.



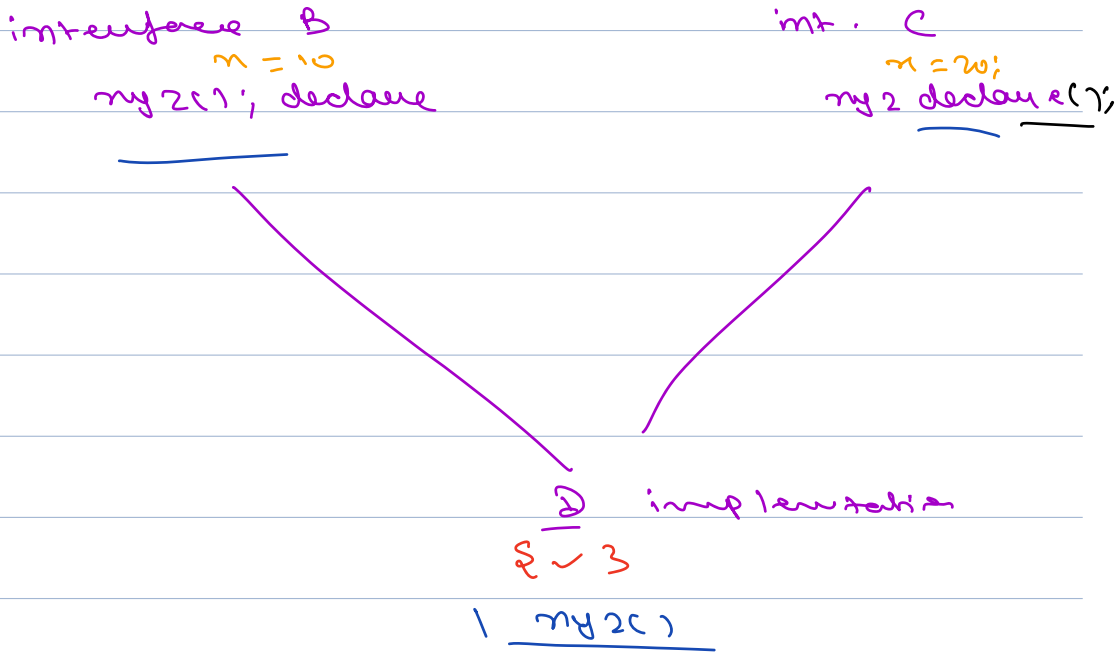
D implements B, C {

D.myz();

}

D d = new D();

~~d.myz();~~



private in Interfaces

```
interface abc {
    default void fnc() {
    };
    void xyz();
}
```

↓

So classes implementing it.

→ Two Solns

1st Soln.

```
interface abcExtended()
    void fnc();
    void xyz();
}
```

```
ins abc {
    default xyz() {
    }
}

class def implementing abc {

}

abc x = new def();
```

① Interfaces → static functions

// utility method.

```
}
```

```
Interface {
```

data member → public static final

void method1();

default void method() {

}

static void method2() {

}

final keyword

final variable → can't re-assign.

final class → can't inherit / extend it.

final method → can't override it.

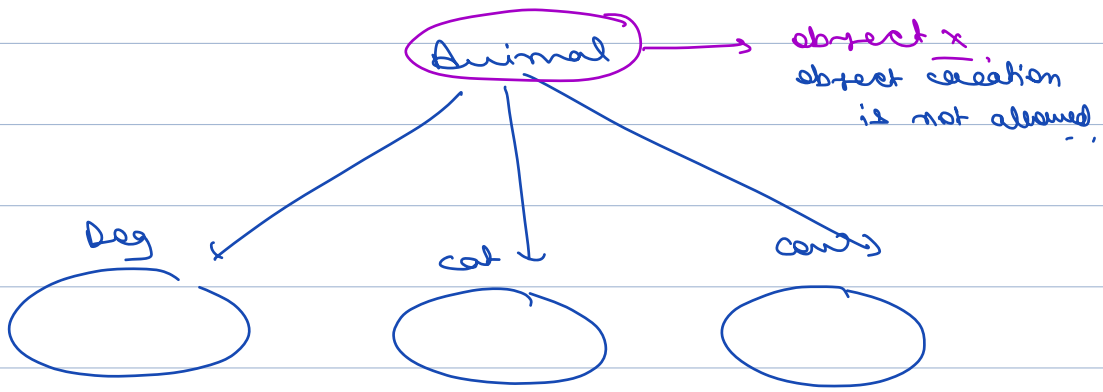
Abstract classes

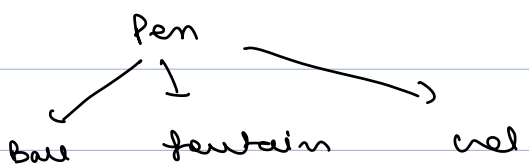
↓

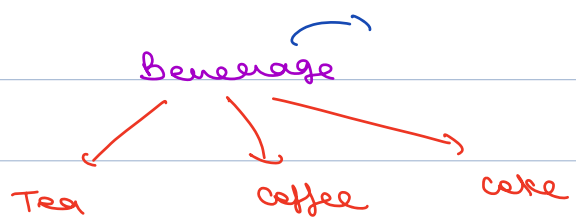
class in which you don't have to define every function.

Interface → to group behaviours.

Abstract classes → logical relationship.







Abstract classes

→ normal data
→ normal functions

- abstract method(),
  ↓
  don't have defn
  'inheriting class will
  be implementing it.

Abstract class might not have abstract method.

Abstract methods will always have
abstract class.

```
Int Runner {

    default walk () {

    }
}
```

```
abstract class Animal {

    public walk () {

    }
}
```

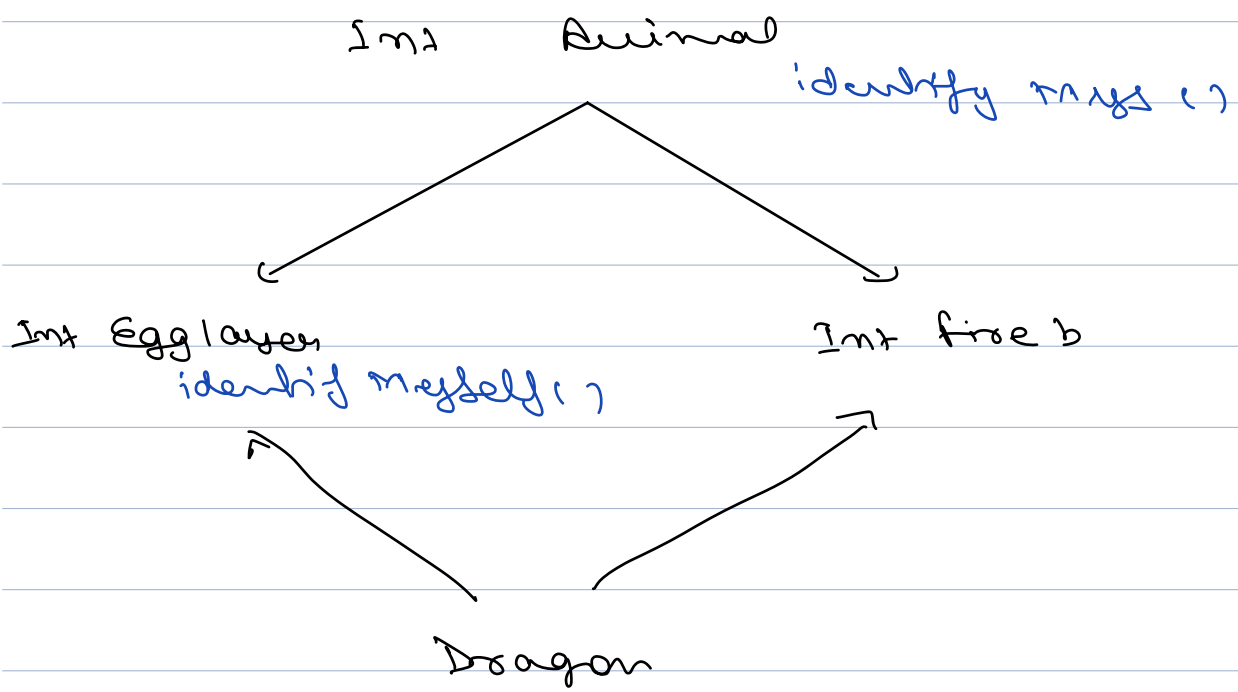
```
Dog d = new Dog ();
d.walk ();
```

```
Dog extend Animal implent Runner {

}
```

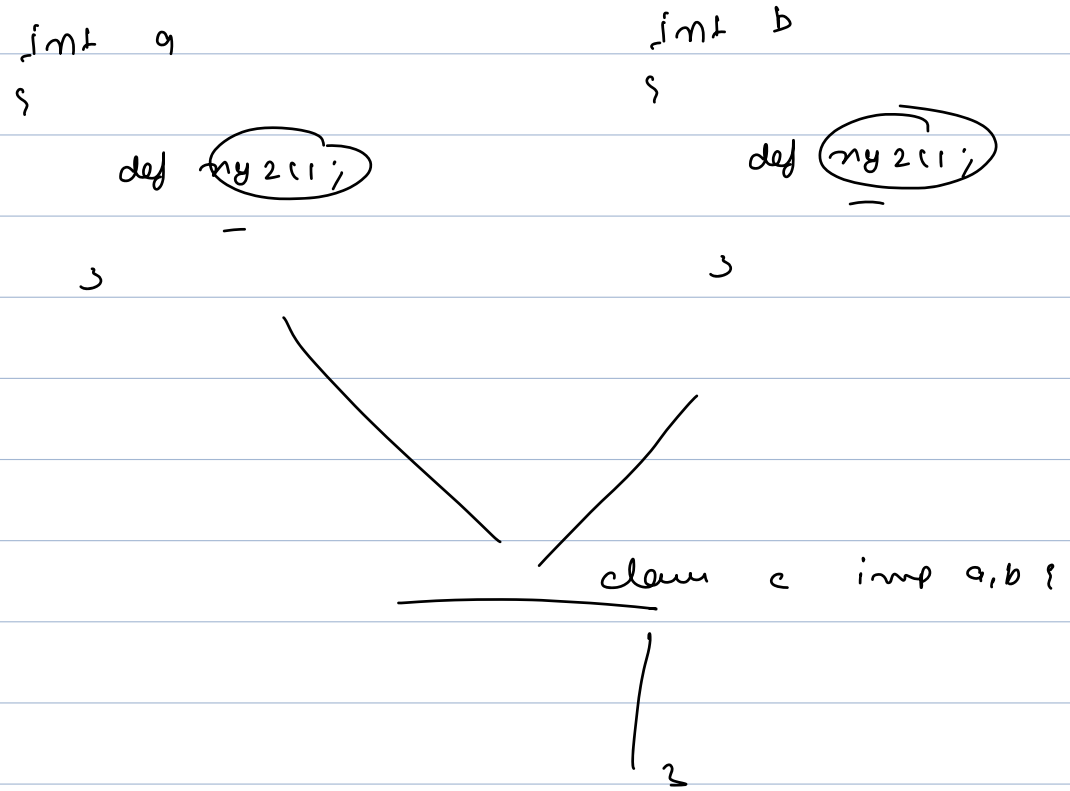
(1) Instance methods are always preferred over interface default method.

(2)

```java
public interface Animal {
    default public String identifyMyself() {
        return "I am an animal.";
    }
}

public interface EggLayer extends Animal {
    default public String identifyMyself() {
        return "I am able to lay eggs.";
    }
}

public interface FireBreather extends Animal { }

public class Dragon implements EggLayer, FireBreather {
    public static void main (String... args) {
        Dragon myApp = new Dragon();
        System.out.println(myApp.identifyMyself());
    }
}
```



1. Instance methods are preferred over interface default methods.
2. Methods that are already overridden by other candidates are ignored. This circumstance can arise when supertypes share a common ancestor.



```
public class Horse {
    public String identifyMyself() {
        return "I am a horse.";
    }
}
public interface Flyer {
    default public String identifyMyself() {
        return "I am able to fly.";
    }
}
public interface Mythical {
    default public String identifyMyself() {
        return "I am a mythical creature.";
    }
}
public class Pegasus extends Horse implements Flyer, Mythical {
    public static void main(String... args) {
        Pegasus myApp = new Pegasus();
        System.out.println(myApp.identifyMyself());
    }
}

The method Pegasus.identifyMyself returns the string I am a horse.
```