Agenda

1) Interfaces
2) Abstract classes
3) static keyword

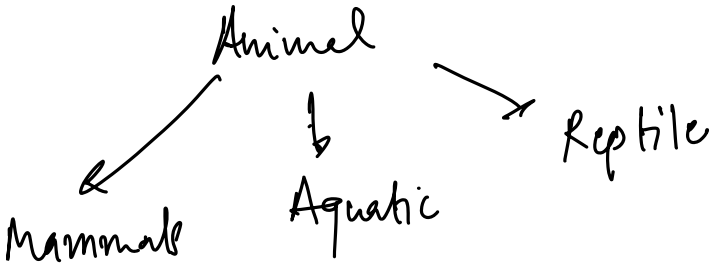
Interfaces

When we talk about classes → attributes, behaviours.

What is a class → blueprint

Concept → does not have attributes
→ No definition inside the methods.

concept is categorized by the type of behaviour it supports



e.g. anyone who can walk, eat and run is an animal.

every animal will have it's own way of walking, running, eating.

eat, run, walk.

→ Animal.

Using Interfaces.

→ If classes are blueprints of entities.
→ Interfaces are blueprints of behaviours

```
interface Animal {
    void eat();
    void walk();
    void run();
}
```

→ function declarations.

Note:- * Interfaces have method declaration and not definition

* There are no attributes in interfaces.

```
class cat implements Animal {
    void eat(){
        Soplm("Cat is eating");
    }
    void walk(){
        Soplm("Cat is walking");
    }
    void run(){
        Soplm("Cat is running");
    }
}
```

Soplm → System.out.println( )

```
void meow(){
    Soplm("meow");
}
```

```
class Dog implements Animal {
    void eat(){ Soplm("Dog is eating"); }
    void run(){ Soplm("Dog is running"); }
    void walk(){ Soplm("Dog is walking"); }
    void bark(){ Soplm("Bark"); }
}
```

* You can't skip any method declared in Animal interface if you implement it in your class.
* There are more specific methods in the classes.

Stack → LIFO

↳ push, pop, isEmpty, peek

How to implement stack.

→ Using Array

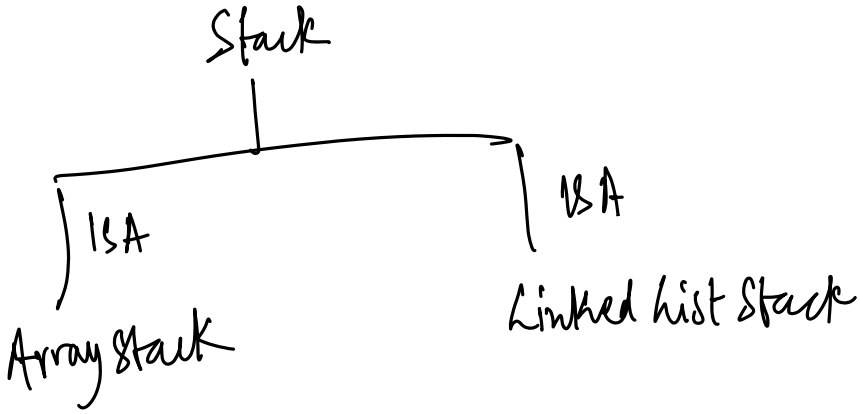
→ Using linked list

```
Interface Stack {
    int pop();
    void push(int val);
    boolean isEmpty();
}
```

```
class ArrayStack implement Stack {
    int top;
    int[] arr;
    int maxSize;
    void push(int val) {
        arr[top] = val;
        top+=1;
    }
    int pop() {
        return arr[top--];
    }
```

```
class LLStack implement Stack {
```

```
boolean isEmpty(){
|
y
}
```




```
Stack s1 = new ArrayStack();
Stack s2 = new LinkedListStack();
```

Code to interface

Principle:- Program to Interface, not an implementation

To
the Principal, → person who has authority to grant leave.

Best Regards

Don't code to specific class (implementation), code to an interface.

Phonepe

↳ Servers went down.
YesBank moratorium.

Phonepe was using YesBank's api.

↳ was down for an entire day.
↳ shifted it's backend APIs from YesBank to ICICI Bank.

How?
↳ Efficient and clean coding using Interfaces.

```
class Phonepe {
    YesBankApi api = new YesBankApi();
    addToWallet() {
        api.addToWallet();
    }
    api.checkBalance(long accountNo.);
    api.pay();
}

interface BankAPI {
    double checkBalance();
    transferMoney(String from, String to);
    registerAccount(acc.No, Name);
}
```

```
class YesBankApi {
    int checkBalance(long ac.) {
        ...
    }
    boolean addToWallet() {
        ...
    }
    String pay() {
        ...
    }
}
```

```
class ICICIBankApi {
    double getBalance(acc.No) {
    }
    int pay(from acc, to acc) {
    }
}
```

```
interface BankAPI {
    double checkBalance();
    transferMoney(String from, String to);
    registerAccount(acc.No, Name);
}
```

YesBankApi implements BankApi

che
fru
req.

ICICI Bank implements Ban

```
class Phonepe {
    BankApi api = new ICICIBankApi();
                  ~~new YesBankApi();~~

    api.checkBalance();
    api.transferMoney();
}
```

Systems should be loosely coupled.

Abstract Classes.

Entity
- attributes and behaviours.
- But I don't know how these behaviours will work.

```
abstract class Animal {                    → Abstract class
    String name;
    int age;
    abstract void walk();
    abstract int noOfLegs();               → All child classes of Animal will need to implement these abstract methods.
}
```

```
abstract class Tiger extends Animal {
    void walk() {
        print("Tiger is walking");
    }
}
```

* We cannot create objects of Abstract classes.

Animal a = new Tiger(); → ✗ incomplete

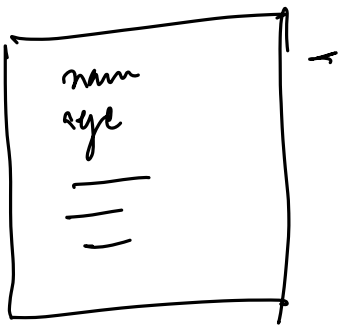
```
class Cat extends Tiger {
    int no.of legs () {
        return 4;
    }
}
```

Animal a = new Cat();

a.walk(); → "Tiger is walking"

a.no.of legs(); → 4.

a = @700



User user = new Student()

List list = new ArrayList<>();

List list = new LinkedList<>();

2 things:-

→ Why multiple inheritance not allowed in Java

→ Static Keyword.