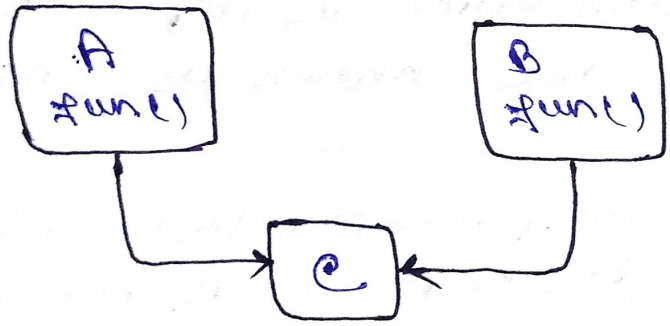
Object-Oriented-Programming #LEC-5

Multiple Inheritance:

- Java doesn't support Multiple inheritance.

Suppose we have class A contain a function with name fun(), also we have class B which also contain function with name fun(). and we have child class which inherits both class A and B. Now if call the function with name fun() in class C, it will confused which one to call, that's why Java doesn't support multiple inheritance.



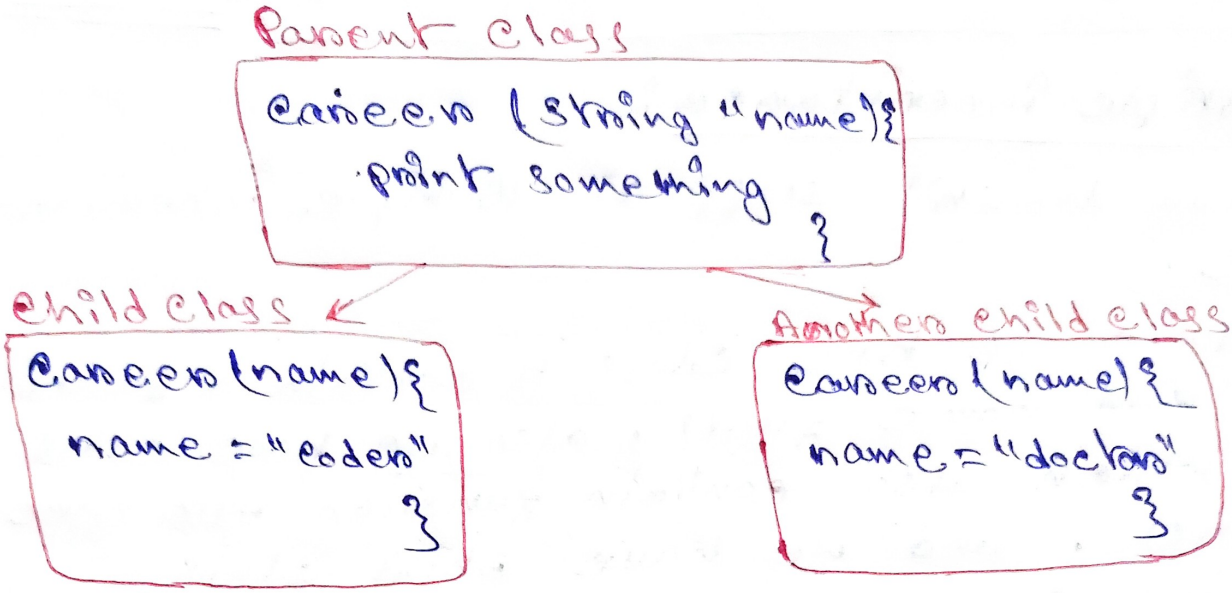
C.fun() [this will get confused which one to call]

Abstract classes:

- sometimes you want to create a parent class that gonna tell us what to do and not how to do it.
- The parent class will give us a generalize form that gonna be shared by all the child classes, and the child class can fill in the

Scanned with ACE Scanner

details by it self.



- here, the super class or parent class going to give us the generalized form of the function not the body, it's not implementing any method but just giving definition of it, such ~~class~~ methods are known as abstract methods.
- ~~When~~ So, in your parent classes when your function doesn't have the body it totally depends on your child classes to provide the body for that function
- Using the override method we can change the body of function
- you wanna make sure that the child class must override all the necessary methods, these are known as Abstract methods

Scanned with ACE Scanner

Syntax:

```
abstract void canEat(String name);
```

- Any class that contain one or more Abstract methods, must also be declared as abstract.

Note: Some important properties about Abstract classes:

1. We can not create object out of Abstract classes, for that we have to use override methods to implement object out of Abstract classes
2. You also can not create Abstract constructors

but why?

→ Because Abstract class is an incomplete class that contains abstract methods without any implementation, therefore it cannot be instantiated directly

3. we also can not create, Abstract static methods.

→ as, Abstract methods are need to be overriden but static methods can not be. So, there is no point creating Abstract static methods. but, we can create static method in Abstract classes.

4. Abstract class can contain normal methods

Scanned with ACE Scanner

5. Even though we can no create object out of Abstract class but we can use it as reference variable while creating object.

```
Parent daughters = new Daughters(28);
```

6. 'final' keyword can not be used before the Abstract classes

✗
```
final public abstract class Parent {
    ... some code
}
```
// it will not work ↑

But 'final' keyword can be used inside Abstract classes like this. ↓

✓
```
public abstract class Parent {
    final int value;
    ... some more code
}
```
// will work

7. Incase of Abstract classes too multiple inheritance is not allowed.

✗
```
public class Son extends Parent, Parent2 {
}
```
// this will not work

Scanned with ACE Scanner

Interfaces :

- Interfaces contain Abstract functions, Abstract functions are functions where no body of the functions are allowed
- Interfaces are like Class only, but not completely
- In the Interfaces classes are Abstract and public, So as classes are Abstract we can not make object out of it.
- As we can not create object, So the variables will be static and final by default, as we know that static stuff doesn't depend on object.
  variables will be static and final by default in interfaces

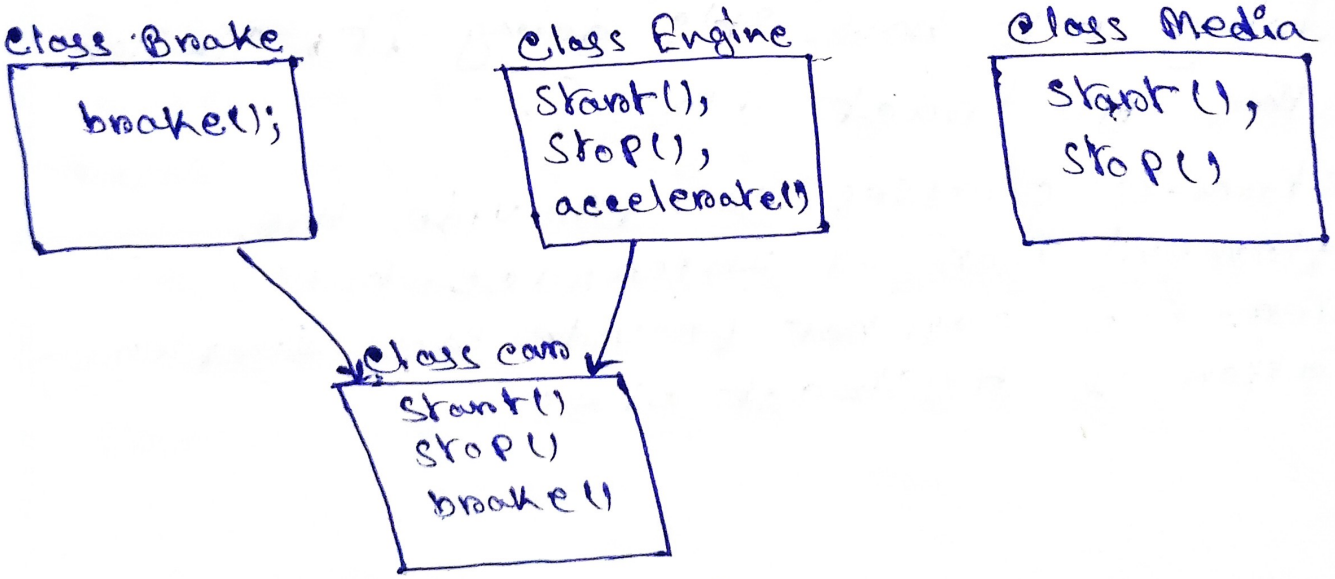
- Interfaces specify what the class is doing not how it's doing it. Similar to the Abstract classes.
- Abstract classes can provide the implementation of Interface, but Interface can not provide the implementation of Abstract class.

Scanned with ACE Scanner

- Some key difference between Abstract class and Interface.

1. Java Interface can be implemented using the 'implement' keyword. where as incase of Abstract class we use 'extends' keyword
2. we can implement multiple Interface, means a Interface can extends another Java Interface, but a Java class can extends only one Abstract class can not do multiple.
3. Abstract class can provide the implementation of interface, but Interface can not provide the implementation of Abstract class.

- So, ~~if~~ previously we unable to do this ↓



where, A class inheriting the properties of multiple classes, but in Interfaces

Scanned with ACE Scanner

now we can do that. Means multiple inheritance is allowed in Interface

- So, here we can see two classes, which are unrelated to each other they can implement the same interface.
  the classes doesn't have to be related to each other like, child and parent manner.
  both or multiple classes can be totally distinguised from one another, may have no relation among them but still can implement same interface.

- Separate classes in Same Interface:

By implementing an interface, a class agrees to implement all of the methods defined in the interface,.
So, if multiple classes implement the same interface, they will all have the same methods with the same signatures, which allows them to be used interchangeably in code that relies on the interface.

- we can also extends interfaces

- Annotations: Annotations are also internally an interface.

Scanned with ACE Scanner

- **default methods within interface :**

Suppose if we have default methods within a interface.

example :

```
public interface A {
default void fun() {
System.out.println("I am in A");
}  }
```

the primary motivation for this default methods was to provide a means by which interface would be expanded without breaking the existing code

for example : if you add another method without an body in interface, we would have to provide the body of that method in all the classes that implements that interface.

- Now suppose in the below code ↓

```
public class Main implements A, B {
@Override
public void greet() {
}
}
```

if A and B both consit default method the code will not work

Scanned with ACE Scanner

- So, we can expand the interfaces without breaking the existing code.

Note:

- static methods are the ones that should be used in interfaces only. they should have a body.

As, you can not override static methods, that's why static interface methods should always have a body.

- we must implent the static methods by the interface name

example:

```
public interface A {
    static void greeting() {
        print("I am in static method");
    }
}
```

Main class

```
public class Main implements A, B {
    public static void main(String[] args) {
        Main obj = new Main();
        A.greeting();
    }
}
```

- Another point, Suppose in the parent class if a method is protected, than in another class when we override the protected method from parent to

Scanned with ACE Scanner

child class, the method should be protected or better than that like, public .. It can not be more restricted than protected, it has to be less restricted than protected, can not be more restricted than that like private.

- **Nested Interface:**

  - Nested Interface can be declared as public, private & protected.
  - Top level interface has to be declared public or default