Abstract class

Abstract Class & abstract method

- Abstract classes are classes that cannot be instantiated directly
- Cannot create objects .
- They may contain a mix of abstract methods (methods without a body) & concrete methods (methods with an implementation)

Some points :-

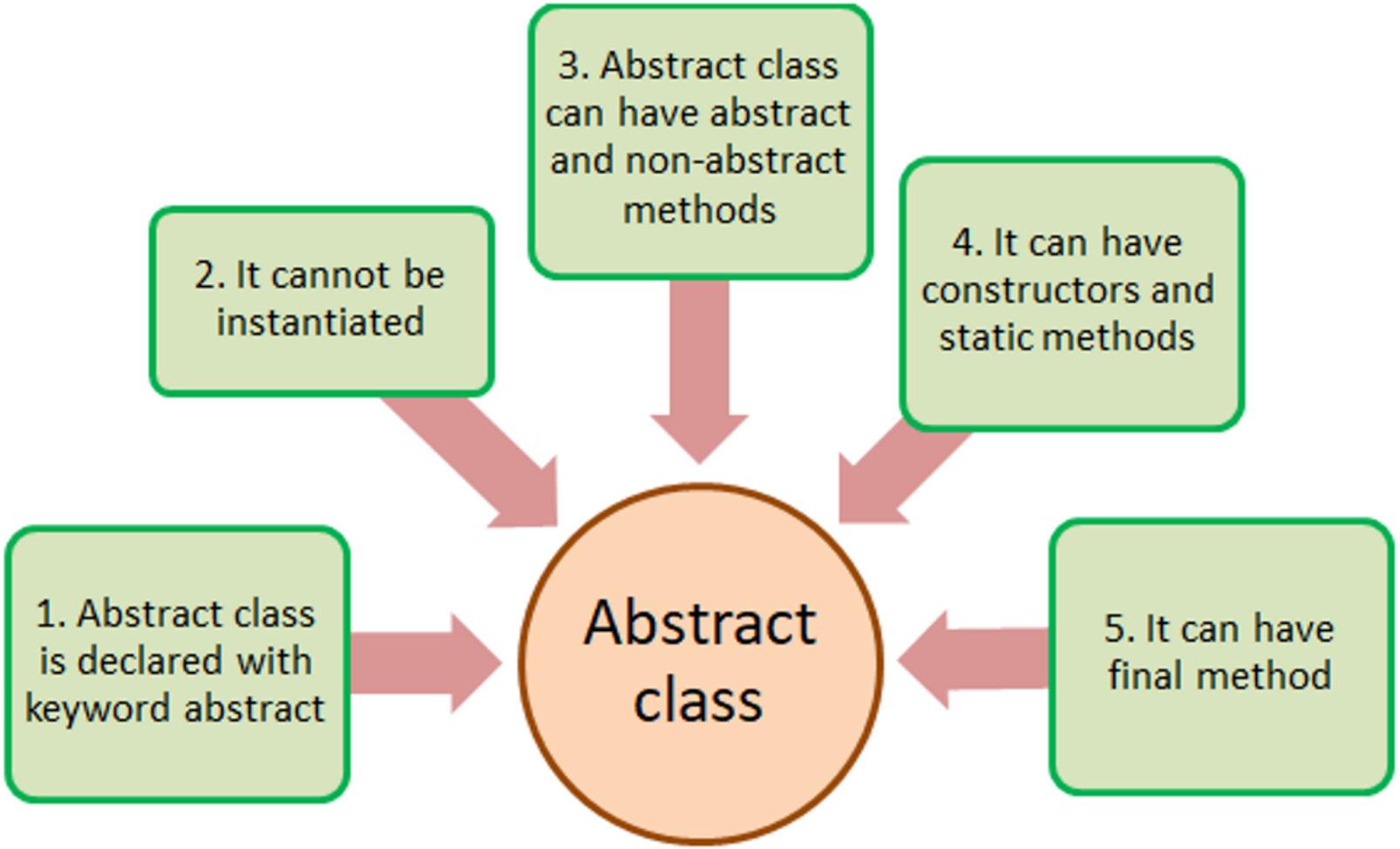
a. declared using the `abstract` keyword before the class definition.

b. can have constructors, which are called when an instance of subclass is created.

c. contain fields (instance variables) & concrete methods with implementation.

d. also contain abstract methods, which are declared without a body and must be implemented by the subclasses.

e. A class that extends an abstract class must implement all the abstract methods defined in the abstract class.


3. Abstract class can have abstract and non-abstract methods
2. It cannot be instantiated
4. It can have constructors and static methods
1. Abstract class is declared with keyword abstract
Abstract class
5. It can have final method

Example -

```
abstract class Vehicle {
      abstract void start();    // no body
      void stop(){              // has body
          System.out.println("method in abstract class.");
      }
}
class Car extends Vehicle {
     void start() {       // provide implementation    i.e override
              System.out.println("abstract class method is overrided in child class Car");
    }
}
public class Main {
      public static void main(String[] args) {
             Car myCar = new Car();
              myCar.start();
              myCar.stop();
      }
}
```

Interface (implementing multiple interfaces)

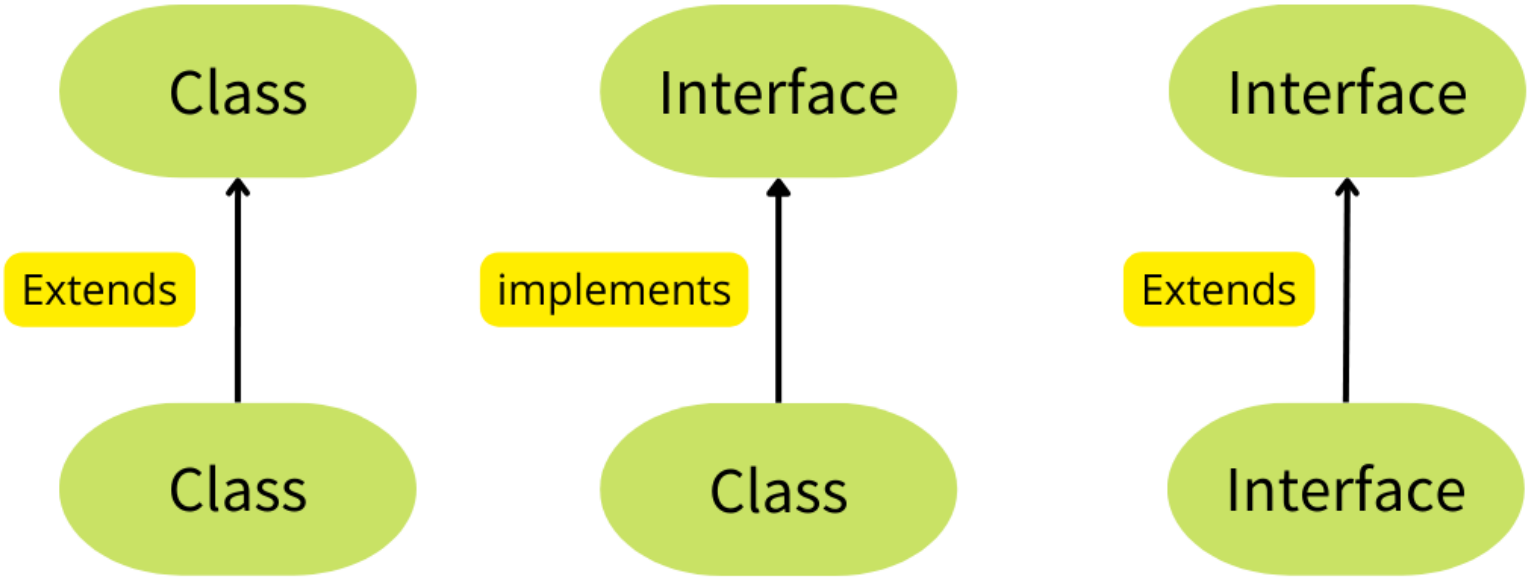
- Inside interface every method is always abstract whether we are declaring or not. Hence, interface is considered as 100% pure abstract class.
- Syntax of declaring interface :–

```
interface interface_name
{
    data member;
    methods();
}
```

Where we can use extends or implements keyword ?

- A class can extend only one class at a time.
- An interface can extend any no. of interface simultaneously .
- A class can implement any number of interface at a time .
- A class can extend one class and can implement any number of interface simultaneously.
- Example – (Interface variable)

```
interface I1
{
   int x =10;
 }
```


Class
Extends
Class
Interface
implements
Class
Interface
Extends
Interface

Some case

Case I – If two interface contain a method with same signature and same return type then in the implementation class , we have to provide implementation for only one method.

Case II – If two interface contains a method with same name but different argument type the in the implementation class , we have to provide implementation for both methods and these methods acts as overloaded methods .

Difference b/w interface & abstract class

Interface	Abstract class
If we don't know about implementation then we should go for interface	If we know implementation but not completely (partial implementation) then we should go for abstract class.
Inside interface every method is always public & abstract, whether we are declaring or not . Hence , it is 100% abstract class.	Every method present inside abstract class need not be public & abstract.
Every variable present inside interface is always public static final	Every variable present inside abstract class need not to be public static final.
For interface variable compulsory we should perform initialization at the time of declaration (otherwise we get error)	For abstract class variable we are not required to perform initialization at the time of declaration .

Final variables, final methods and final class

- **final variable**: Its value cannot be changed once assigned.
- **final method**: Cannot be overridden by subclasses.
- **final class**: Cannot be extended by any other class.

Example –

```
final class Utility {
        final int MAX_USERS = 100;
        final void showMessage(){
            // MAX_USERS =200 ; can no be reassigned
            System.out.println("This is a utility class.");
        }
 }
// class ExtendedUtility extends Utility {  } // error: cannot extend final class
 public class Main {
        public static void main(String[] args) {
            Utility utility = new Utility();
            System.out.println("MAX Users:"+ utility.MAX_USERS);
        }
}
```

Functional interface

- A **functional interface** is an interface with exactly one abstract method. (Single-method interfaces)
- **Key Points**:

➢ Declared using @FunctionalInterface annotation.

➢ Can have default and static methods.

➢ Commonly used with lambda expressions.

Example -

```
@FunctionalInterface
interface Interface1{
        void say(String msg); // abstract method
 }
```

New interface features (Java 8)

- **Default Methods:**

  Introduced in Java 8, allows interfaces to have methods with a default implementation.

- **Static Methods:**

  Introduced in Java 8, static methods can be called directly from the interface.

- **Private Methods:**

  Introduced in Java 9, allows interfaces to have private helper methods.

Default Method

- A default method must have a body.
- The access modifier of default methods are implicitly public.
- A default method is defined using default keyword.
- **Syntax –**

```
interface interfaceName {
    default return_type methodName() {
        // method definition
    }
}
```

Static Method

- A static method must have a body.
- access modifier ->**public**
- called using interface name(because it is static method)
- method is static, we cannot override them in implementing class.
- Syntax –

```
interface interfaceName {
        static return_type methodName() {
            // method definition
         }
}
```

Private Method

- A private method must have a body.
- defined using **private** access modifier.
- These method can be accessed within the interface only.
- Since these methods are private, we cannot override them in implementing class.
- Syntax –

```
interface interfaceName {
        private return_type methodName() {
                // method definition
         }
 }
```

Functional Interface vs Interface

Functional Interface	Regular Interface
An interface with exactly one abstract method.	An interface can have multiple abstract methods.
Can have default and static methods, but only one abstract method.	Can have multiple default and static methods.
Example- Runnable, Callable, Comparator, Function, etc.	Example- List, Set, Map, custom interfaces, etc.

Assignments

1. Write an abstract class '**Shape**' with an abstract method calculateArea() and Create two subclasses, **Circle** and **Rectangle**, that implement the calculateArea() method.

2. Write an example program to demonstrate how a constructor in an abstract class can be used.

3. Design an abstract class '**Vehicle**' with an abstract method fuelEfficiency() to calculate mileage and a normal method startEngine() that prints "Engine started." Create subclasses **Car** and **Bike** and implement fuelEfficiency().

4. Create an interface **Calculator** with methods add(int a, int b) and subtract(int a, int b), Implement this interface in a class **SimpleCalculator**.

Assignments

5. Create two interfaces , **Printer** with a method print() and **Scanner** with a method scan().Now Create a class **AllInOneDevice** that implements both interfaces.[Multiple Interfaces]

6. Create an interface **Greeting** with a default method sayHello() that prints "Hello!“ and Create a class **EnglishGreeting** that overrides this default method.

7. Create an interface **MathUtils** with a static method square(int x) that returns the square of a number , after that call the method from the main method without creating an instance.

Practice Questions

1. Is java 100% pure object-oriented language ? Why Java is not 100% object-oriented ?

2. Does constructor return any value?

3. How to call one constructor from the other constructor ?

4. Can we declare constructor as a private?

5. What is the IS-A relationship in java?

6. Can we override the overloaded method?

7. Can we override static method?

8. Can we override private methods in java?

9. Can we override main ( ) method?

10. Can we declare constructor as a final?

THANK YOU!!

C-DAC Patna