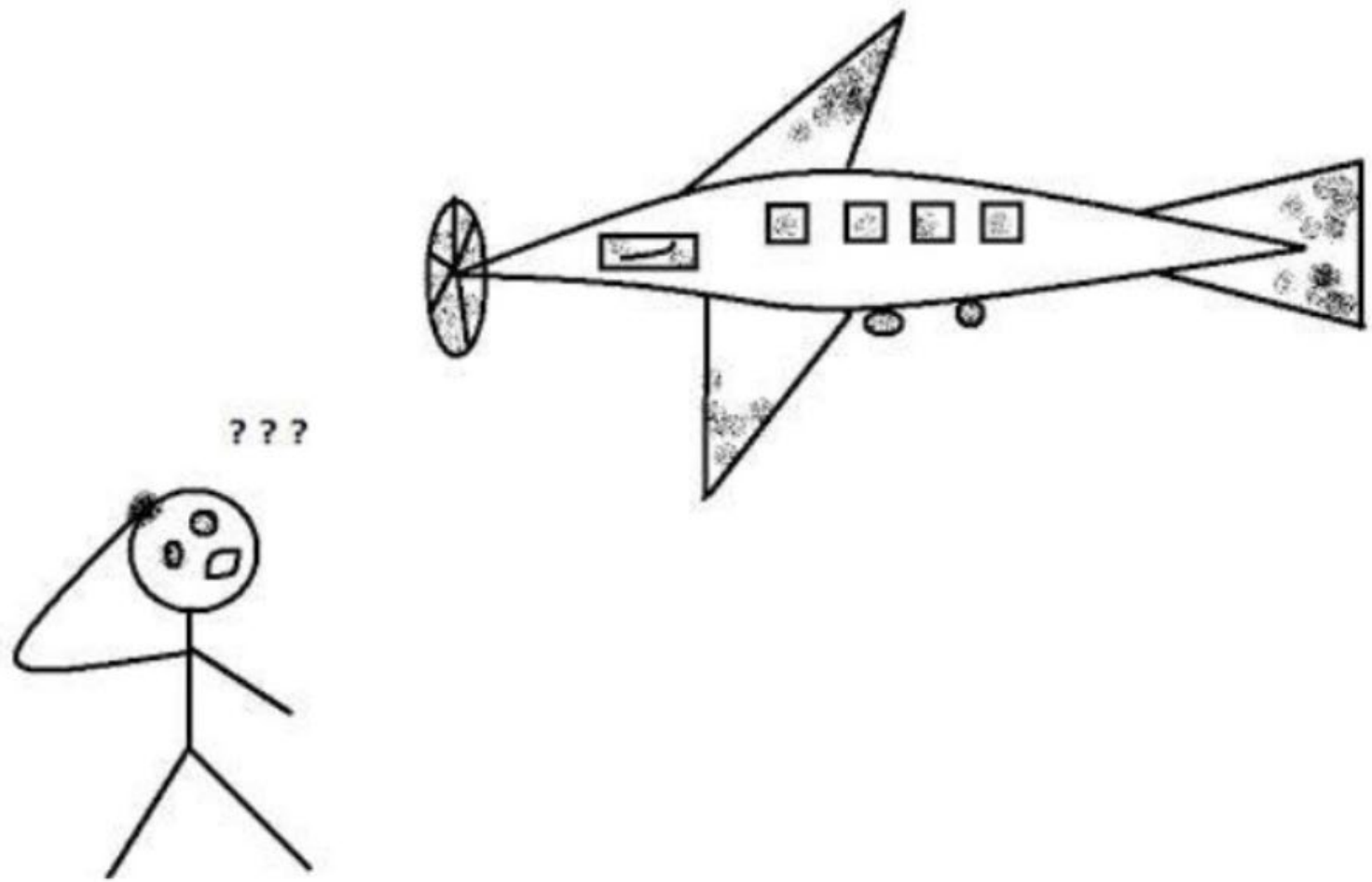
Abstract class and interface in java

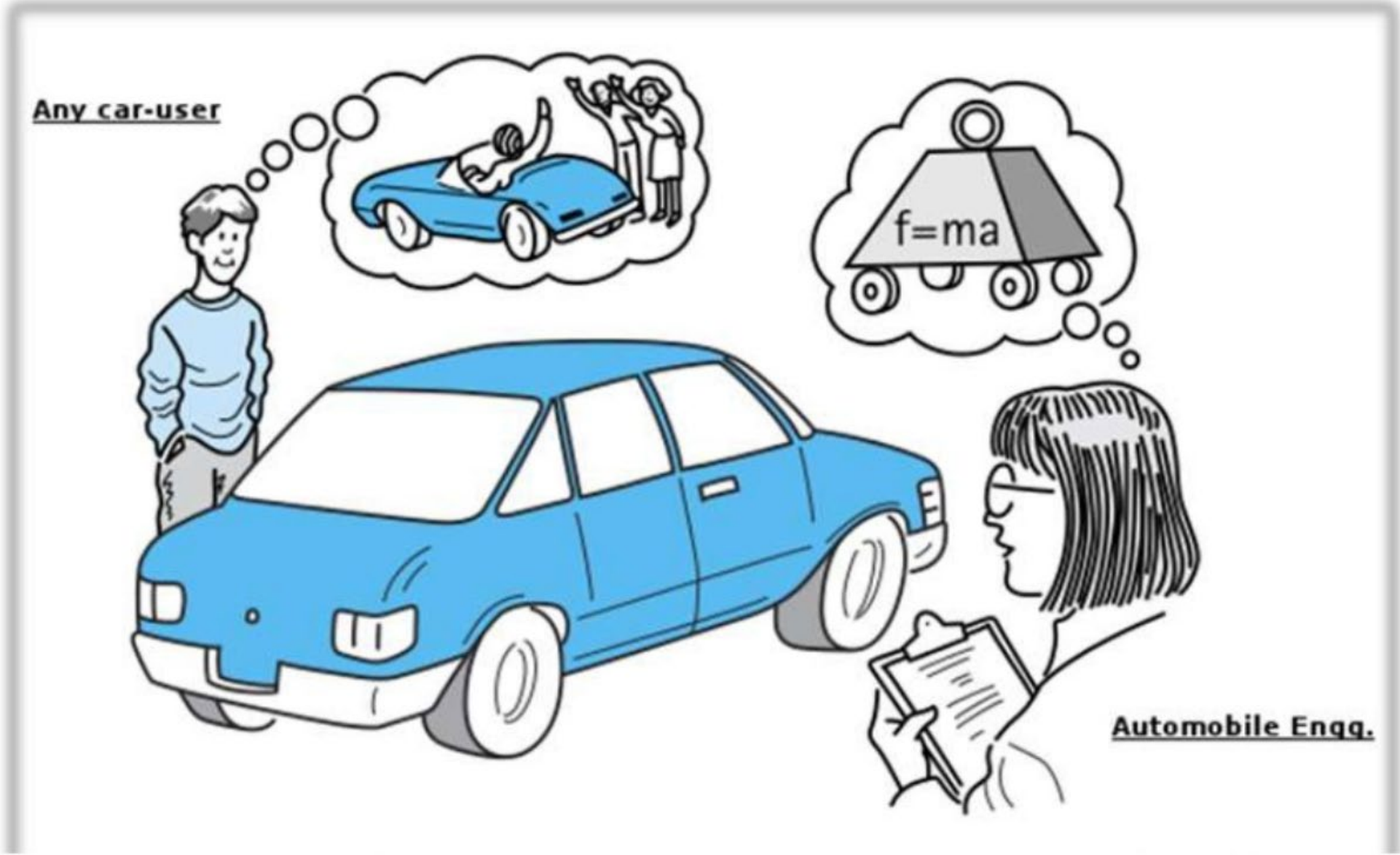
Abstraction in Java



Abstraction in Java

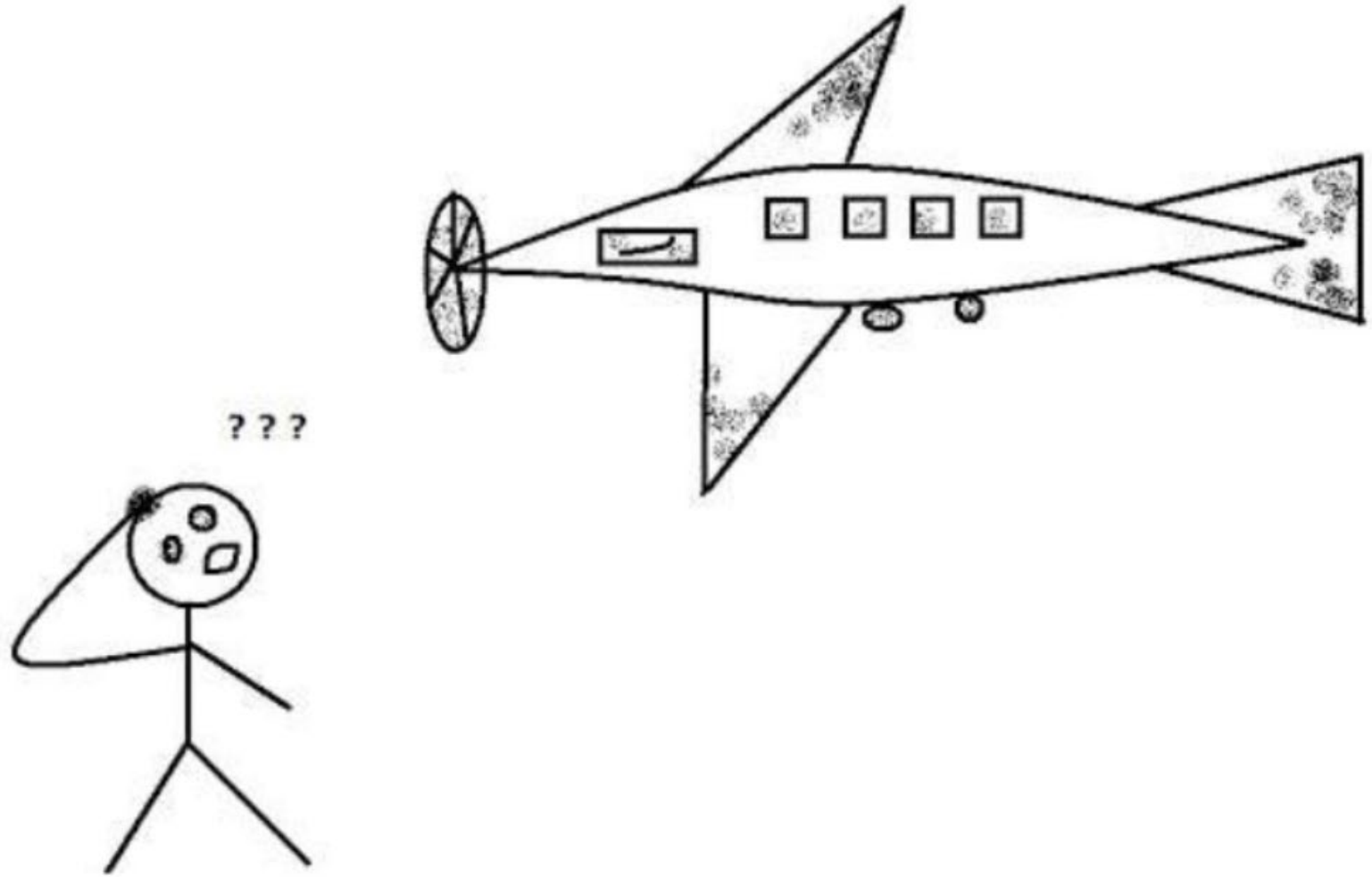
- Another good feature of OOPS is that we can hide the complex and unnecessary details of any object and can display only the required details which are necessary to be displayed so that a person can use that object.
- **Abstraction** is the concept of exposing only the required essential characteristics and behavior with respect to a context.
- Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

Abstraction in Java



An includes the essential detail relative to the perspective of the viewer

Abstraction in Java



Suppose there is an airplane flying in the sky. We know that it is flying, but we don't know how exactly it is flying and what are the underlying mechanisms, working and all.

Abstraction in Java

- **Note:** Data abstraction can be used to provide security for the data from the unauthorized methods.
- **Note:** In java language data abstraction can be achieve using class.

- **How to Achieve Abstraction ?**
- There are two ways to achieve abstraction in java
- Abstract class (0 to 100%)
- Interface (Achieve 100% abstraction)

Abstract class in Java

- We know that every java program must start with a concept of class that is without classes concept there is no java program perfect.
  In java programming we have two types of classes they are

- Concrete class (also call normal class)
- Abstract class

Abstract class in Java

- **Concrete class in Java**
- A concrete class is one which is containing fully defined methods or implemented method.

```
class Normal
{
 void  msg()   //method
 {
   System.out.println("Hello Students");
 }
public static void main(String args[]){
 Normal n= new Normal();    //creating an object
 n.msg();
 }
}
```

Abstract class in Java

- **Concrete class in Java**
- Every concrete class have specific feature and these classes are used for specific requirement but not for common requirement.
- If we use concrete classes for fulfill common requirements than such application will get the following limitations.
- Application will take more amount of memory space (main memory).
- Application execution time is more.
- Application performance is decreased.
- To overcome above limitation you can use abstract class.

Abstract class in Java

- A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).
- An abstract class is never instantiated.
- It is used to provide abstraction.
- **Syntax :**

```
abstract class class_name { }
```

- **Example :**

```
abstract class A { }
```

Abstract class in Java

- An abstract class is a class that is declared abstract
- Abstract classes cannot be instantiated
- Abstract classes can be sub-class
- It may or may not include abstract methods
- When an abstract class is sub-class, the subclass usually provides implementations for all of the abstract methods in its parent class
- If subclass doesn’t provide implementations then the subclass must also be declared abstract.

Abstract Method in Java

- Method that are declared without any body within an abstract class are called abstract method.
- The method body will be defined by its subclass. Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.
- **Syntax :**

```
abstract ReturnType methodName(List of formal parameter);
```

- **Example :**

```
abstract void run();    // No definition
abstract int sum(int a, int b); // No definition
```

Abstract class in Java

- In the computer science perspective, Abstraction is the process of separating ideas from their action.
- Yes, In the computer science, Abstraction is used to separate ideas from their implementation. Abstraction in java is used to define only ideas in one class so that the idea can be implemented by its sub classes according to their requirements.
- Let see an example in next slide how it is true.

Example of Abstract class in Java

```
abstract class Animal {
   abstract void soundOfAnimal();  // It is just an idea
}
class Cat extends Animal {
   void soundOfAnimal() {
      System.out.println("Meoh");
      //Implementation of the idea according to requirements of sub class
   }
}
class Dog extends Animal{
   void soundOfAnimal() {
      System.out.println("Bow Bow");
      //Implementation of the idea according to requirements of sub class
   }
}
```

Example of Abstract Class that has Abstract Method

```
abstract class A {
  abstract void msg();
  }
class B extends A
{
   void msg(){
   System.out.println("Hello My Name is Adil");
 }
 public static void main(String[] args) {
    B = new B();
    b.msg();
 }
}
```

Example of Abstract Class that has Abstract Method

```
abstract class A {
  abstract void msg();
  }
class B extends A
{
   void msg(){
   System.out.println("Hello My Name is Adil");
 }
 public static void main(String[] args) {
     B = new B();
     b.msg();
 }
}
```

Abstract class with Concrete(Normal) Method

```
abstract class A {
  abstract void msg();
  public void normal(){
   System.out.println("This is Concrete Method"); }
}
class B extends A {
  void msg() {
   System.out.println("This is Abstract Method");
 }
 public static void main(String[] args) {
   B b = new B();
   b.msg();
   b.normal();
 }
}
```

Points to Remember

- Abstract classes are not Interfaces. They are different, we will study this when we will study Interfaces.
- An abstract class may or may not have an abstract method. But if any class has even a single abstract method, then it must be declared abstract.
- Abstract classes can have Constructors, Member variables and Normal methods.
- Abstract classes are never instantiated.
- When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.
- Abstract method must be in a abstract class.
- An abstract method must be overridden in a subclass.

Understanding the Real Scenario of Abstract Class

- In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**.
- A **factory method** is the method that returns the instance of the class. We will learn about the factory method later.
- In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

```
abstract class Shape{
  abstract void draw();
  }
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
   void draw()   {  System.out.println("Drawing Rectangle"); }
}
class Circle extends Shape{
    void draw()   {  System.out.println("Drawing Circle");  }
}
//In real scenario, method is called by programmer or user
class Test{
public static void main(String args[]){
 //In real scenario, object is provided through method e.g. getShape() method
   Shape s=new Circle();
   s.draw();
  }
}
```

Output is:
Drawing Circle

Abstract Class Without Any Abstract Method

- In Java, we can have an abstract class without any abstract method. This allows us to create classes that cannot be instantiated, but can only be inherited.
- Abstract classes cannot be instantiated, means we can't create an object to Abstract class.
- We can create Subclasses to Abstract classes.

Abstract Class Without Any Abstract Method

```
abstract class Base {
    void fun() {
     System.out.println("Base Class Method"); }
}
class Derived extends Base {


 public static void main(String args[]) {
    Derived d = new Derived();
    d.fun();
    }
}
```

Abstract Class Without Any Abstract Method

```
abstract class Base {
    void fun() {
     System.out.println("Base Class Method"); }
}
class Derived extends Base {


 public static void main(String args[]) {
    Derived d = new Derived();
    d.fun();
    }
}
```

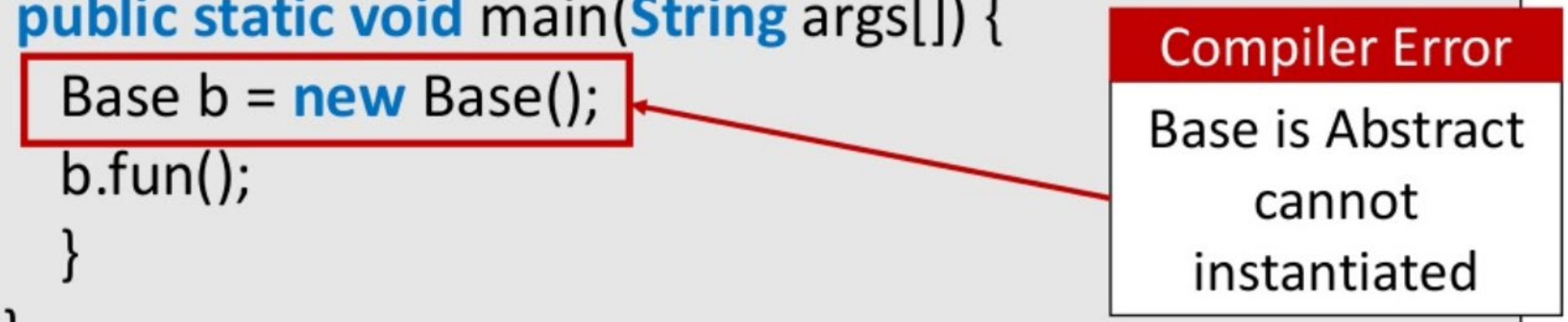
There is no Abstract Method

Abstract Class Without Any Abstract Method

```
abstract class Base {
    void fun() {
     System.out.println("Base Class Method"); }
}
class Derived extends Base {



 public static void main(String args[]) {
    Base b = new Base();
    b.fun();
    }
}
```



Abstract Class Without Any Abstract Method

```
abstract class Base {
   void fun() {
    System.out.println("Base Class Method"); }
}
class Derived extends Base {



 public static void main(String args[]) {
 //Reference is of parent class and object of child class
  Base d = new Derived();  //Upcasting
   d.fun();
    }
}
```

Abstract Class with Abstract Methods

```
abstract class Parent {
    abstract int sum(int a,int b);
 }
class Child extends Parent
{
   int sum (int a,int b) {
      return a+b;
   }
public static void main(String args[]){
   Child c=new Child();
   System.out.println("Sum is: "+c.sum(10, 20));
}
}
```

Output is:

Sum is: 30

Abstract class in Java

- An abstract class can have data member, abstract method, method body, constructor and even main() method.
- A constructor of abstract class is called when an instance of a inherited class is created.
- Let see an example in next slide.

An Abstract Class With Constructor

```
abstract class Base {
   Base() {
      System.out.println("Base Constructor Called"); }
   abstract void fun();
}
class Derived extends Base {
   Derived() {
      System.out.println("Derived Constructor Called"); }
   void fun() {
      System.out.println("Abstract Method"); }
}
class Main {
   public static void main(String args[]) {
     Derived d = new Derived();
   }
}
```

An Abstract Class With Constructor

```
abstract class Base {
   Base() {
      System.out.println("Base Constructor Called"); }
   abstract void fun();
}
class Derived extends Base {
   Derived() {
      System.out.println("Derived Constructor Called"); }
   void fun() {
      System.out.println("Abstract Method"); }
}
class Main {
   public static void main(String args[]) {
     Derived d = new Derived();
   }
}
```

Constructor of Abstract Class

Output is:

Base Constructor Called
Derived Constructor Called

Abstract Class Complete Example-1

```
abstract class Person {
        private String name;
        private String gender;
  public Person(String nm, String gen){
        this.name=nm;
        this.gender=gen;
        }
  //abstract method
  public abstract void work();
  public String toString(){
        return "Name="+name+"::Gender="+gender;
  }
  public void changeName(String newName) {
        this.name = newName;
  }
}
```

Abstract Class Complete Example-2

```
public class Employee extends Person {
    private int empId;
  public Employee(String nm, String gen, int id) {
        super(nm, gen);
        this.empId=id;
        }
  //implementation of abstract method
  public void work() {
        if(empId == 0){
        System.out.println("Not working");
        }
    else{
        System.out.println("Working as employee!!");}
        }
```

Abstract Class Complete Example-3

```
public static void main(String args[])
{
    Person s = new Employee("Hina","Female",0);
    Person  e= new Employee("Aslam","Male",22);
    s.work();
    e.work();
    //using method implemented in abstract class - inheritance
    e.changeName("Adil");
    System.out.println(e.toString());

  }
}
```

Output is:

Not working
Working as employee!!
Name=Adil::Gender=Male

Key Points

- If there is any abstract method in a class, that class must be abstract.
-

```
class Base{
abstract void run();
}
```

```
abstract class Base{
abstract void run();
}
```

Important Points about Abstract Class

- Abstract class of java always contains common features.
- Every abstract class participate in inheritance.
- Abstract classes definitions should not be made as final because abstract classes always participate in inheritance classes.
- An object of abstract class can not be created directly but it can be created indirectly.
- All the abstract classes of java makes use of polymorphism along with method overriding for business logic development and makes use of dynamic binding for execution logic.
- A private method can’t be abstract. All abstract methods must be public.

Abstract Classes

- **Advantage of abstract class**
- Less memory space for the application
- Less execution time
- More performance
- **Why Abstract class have no Abstract static Method?**
- In abstract classes we have only abstract instance method but not containing abstract static methods because every instance method is created for performing repeated operation where as static method is created for performing one time operations in other word every abstract method is instance but not static.
- Because "abstract" means: "Implements no functionality", and "static" means: "There is functionality even if you don't have an object instance". And that's a logical contradiction.

Abstract Method with Multilevel Inheritance

```
abstract class Parent {
  abstract int sum(int a, int b);
}
abstract class Child extends Parent {
}
class Child1 extends Child {
   int sum (int a, int b) {
      return a+b;
   }
public static void main(String args[]){
    Child1 c = new Child1();
    System.out.println("Sum is: "+c.sum(10, 20));
  }
}
```

Implementation of sum method in child most class

Output is:

Sum is: 30

- Create Multiple Abstract Methods within a Abstract Class

Multiple Abstract Methods within a Abstract Class-1

```
abstract class Base {
        abstract void get(int a, int b);
        abstract void add();
        abstract void display();
}
class SubClass extends Base {
        int x, y, z;
        void get(int a, int b) {
                x = a;
                y = b;   }
        void add() {
                z = x + y; }
        void display() {
                System.out.println("The Addition is : " + z); }
}
```

Multiple Abstract Methods

Multiple Abstract Methods within a Abstract Class-2

```
class MainClass {

        public static void main(String args[]) {

                SubClass obj = new SubClass();

                obj.get(10, 20);
                obj.add();
                obj.display();
        }
}
```

- Create Multiple Abstract Class & Abstract Methods

Multiple Abstract Class & Abstract Methods-1

```
public abstract class Super {
        abstract void getName(String name);
        abstract void getSex(String sex);
}
abstract class AbstClass extends Super {
        abstract void getCountry(String country);
}
class SubClass extends AbstClass {
        String name, sex, city, country;
        void getName(String name) {
                this.name = name;  }
        void getSex(String sex) {
                this.sex = sex;  }
        void getCountry(String country) {
                this.country = country;  }
```

Multiple Abstract Class & Abstract Methods-2

```
void display() {
                System.out.println("Name : " + name);
                System.out.println("Sex : " + sex);
                System.out.println("Country : " + country);
        }
}
class MainClass {
        public static void main(String args[]) {
                SubClass obj = new SubClass();
                obj.getName("Adil");
                obj.getSex("Male");
                obj.getCountry("PAK");
                obj.display();
        }
}
```

Output is:

Name : Adil
Sex : Male
Country : PAK

Difference Between Abstract class and Concrete class

Concrete class	Abstract class
Concrete class are used for specific requirement	Abstract class are used for fulfill common requirement.
Object of concrete class can be create directly.	Object of abstract class can not be create directly (can create indirectly).
Concrete class containing fully defined methods or implemented method.	Abstract class have both undefined method and defined method.

Abstract class: Some Key Points with Example

- Inside abstract class, we can keep any number of constructors. If you are not keeping any constructors, then compiler will keep default constructor.

```
abstract class AbstractClass{
   AbstractClass()
   {
      //First Constructor
   }

   AbstractClass(int i)
   {
      //Second Constructor
   }

   abstract void abstractMethodOne();  //Abstract Method
}
```

Abstract class: Some Key Points with Example

- Abstract methods can not be private. Because, abstract methods must be implemented somehow in the sub classes. If you declare them as private, then you can't use them outside the class.

```
abstract class AbstractClass
{
   private abstract void abstractMethodOne();
   //Compile time error, abstract method can not be private.
}
```

- Abstract methods can not be static.

```
abstract class AbstractClass
{
   static abstract void abstractMethod();
   //Compile time error, abstract methods can not be static
}
```

Abstract class: Some Key Points with Example

- Constructors and fields can not be declared as abstract.

```
abstract class AbstractClass
 {
    abstract int i;
    //Compile time error, field can not be abstract
    abstract AbstractClass()
    {
    //Compile time error, constructor can not be abstract
   }
}
```

Some Question About Abstract Class

- **Q) What is Abstract class in Java? Or Explain Abstract classes?**
  - A class with **abstract** keyword in class declaration is known as abstract class in Java. Unlike class, an abstract class can contain both abstract methods as well as concrete methods (i.e.; methods with braces and method body or method implementation)
- **Q) What is an Abstract method in Java?**
  - A method declaration preceded/prefixed with abstract keyword with no body or no implementation detail which ends its method signature with semicolon(;) is known as abstract method

Some Question About Abstract Class

- **Q) Whether abstract class compiles successfully, if it contains both concrete & abstract methods together?**
  - Yes, abstract class compile successfully as it can contain both abstract methods and concrete
- **Q) What happens, if sub class extending abstract class doesn't override abstract methods?**
  - Compiler throws error to implement all abstract methods
- **Q) Can abstract class implements interface?**
  - Yes, an abstract class can implement interface and this is allowed

Some Question About Abstract Class

- **Q) Can an abstract class be defined without any abstract methods?**
  - Yes, a class can be declared with abstract keyword even if it doesn't got 1 abstract method
  - But vice-versa is not true; means if a class contains abstract methods then class has be declared with abstract keyword
- **Q) Can we define abstract class without abstract keyword in class declaration?**
  - No, abstract keyword is required at class declaration to declare abstract class

Some Question About Abstract Class

- **Q) Whether it is mandatory to have abstract methods in abstract class? If not, why such design is required?**
  - It's not mandatory to have abstract methods in abstract class
  - Even without a single abstract method in a class can be declared as abstract
  - This is to flag compiler that this class is not for instantiation
- **Q) Whether class compiles successfully, if class contains abstract methods and no abstract keyword at class declaration?**
  - Compiler throws error

Some Question About Abstract Class

- **Q) Can we define constructor inside abstract class?**
  - Yes, we can define constructor inside abstract class
  - Both default & parametrized constructors are allowed inside abstract class
- **Q) Can abstract class be instantiated?**
  - No, abstract class cannot be instantiated
  - Instantiating abstract class throws compile-time error
- **Q) Can we declare abstract method with *static* modifier inside abstract class?**
  - No, an abstract class cannot be ***static***
  - **Compile-time error** will be thrown: The abstract method display in type <abstract-class-name> can only set a visibility modifier, one of public or protected

Some Question About Abstract Class

- **Q) Can an abstract class be *final*?**
  - No, an abstract class cannot be ***final***
  - Abstract methods needs to be implemented; therefore it is overridden in the sub class
  - But by marking ***final***, we are restricting it to override
- **Q) Can we declare abstract method with *private* modifier inside abstract class?**
  - No, an abstract class cannot be declared with private accessibility
  - Compilation error will be thrown with below error

Some Question About Abstract Class

- **Q) Can we declare concrete (non-abstract) method with *final* modifier inside abstract class?**
  - Yes, concrete method can be declared with ***final*** modifier
- **Q) Can we declare abstract method with *private* modifier inside abstract class?**
  - No, an abstract class cannot be declared with private accessibility
- **Q) What are all modifiers allowed for abstract method declaration?**
  - pubic and protected access modifiers are allowed for abstract method declaration

Some Question About Abstract Class

- **Q) Why modifiers such as *final*, *static* & *private* are not allowed for abstract methods declaration in abstract class?**
  - **Final:** as sub-class needs to provide method implementation for all abstract methods inside abstract class, therefore abstract cannot be marked as ***final***
  - **Static:** abstract methods belongs to instance not class, therefore it cannot be marked as ***static***
  - **Private:** abstract methods needs to be overridden in the sub-class for this we need more wider accessibility
  - By marking abstract method declaration with ***final*** or ***static*** or ***private*** modifier –> results in compilation error

Some Question About Abstract Class

- **Q) Can we define *private* constructor inside abstract class?**
  - Yes, it is allowed to have private constructor inside abstract class
- **Q) Is it ok to declare abstract method inside non-abstract class?**
  - No, it is not allowed to have abstract method inside concrete class
  - If there any abstract method, then class must be marked with abstract modifier
- **Q) Can we declare static fields inside abstract class?**
  - Yes, static fields and static methods are allowed to declare inside abstract class

Interface in Java

Interface in Java

- Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). Also, the variables declared in an interface are public, static & final by default.
- The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**.
- It cannot be instantiated just like abstract class.
- Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Interface in Java

- **Why we use Interface ?**
- It is used to achieve fully abstraction.
- By using Interface, you can achieve multiple inheritance in java.
- It can be used to achieve loose coupling.
- Since methods in interfaces do not have body, they have to be implemented by the class before we can access them. The class that implements interface must implement all the methods of that interface.

Interface in Java

- **Properties of Interface**
- It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

Interface in Java

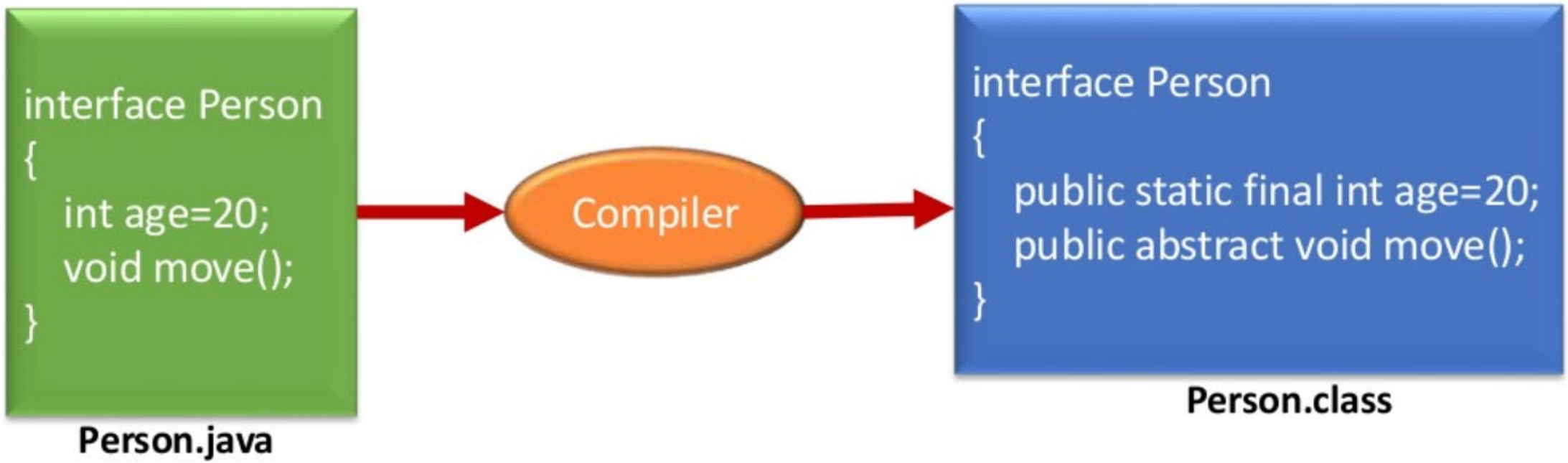
- **How interface is similar to class ?**
- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- Whenever we compile any Interface program it generate .class file. That means the bytecode of an interface appears in a .class file.

Interface in Java

- **How interface is different from class ?**
- You can not instantiate an interface.
- It does not contain any constructors.
- All methods in an interface are abstract.
- Interface can not contain instance fields. Interface only contains public static final variables.
- Interface is can not extended by a class; it is implemented by a class.
- Interface can extend multiple interfaces. It means interface support multiple inheritance

Behavior of Compiler with Interface Program

- In the below image when we compile any interface program, by default compiler added public static final before any variable and public abstract before any method. Because **Interface** is design for fulfill universal requirements and to achieve fully abstraction.



Behavior of Compiler with Interface Program

- **Declaring Interfaces:**
- The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface

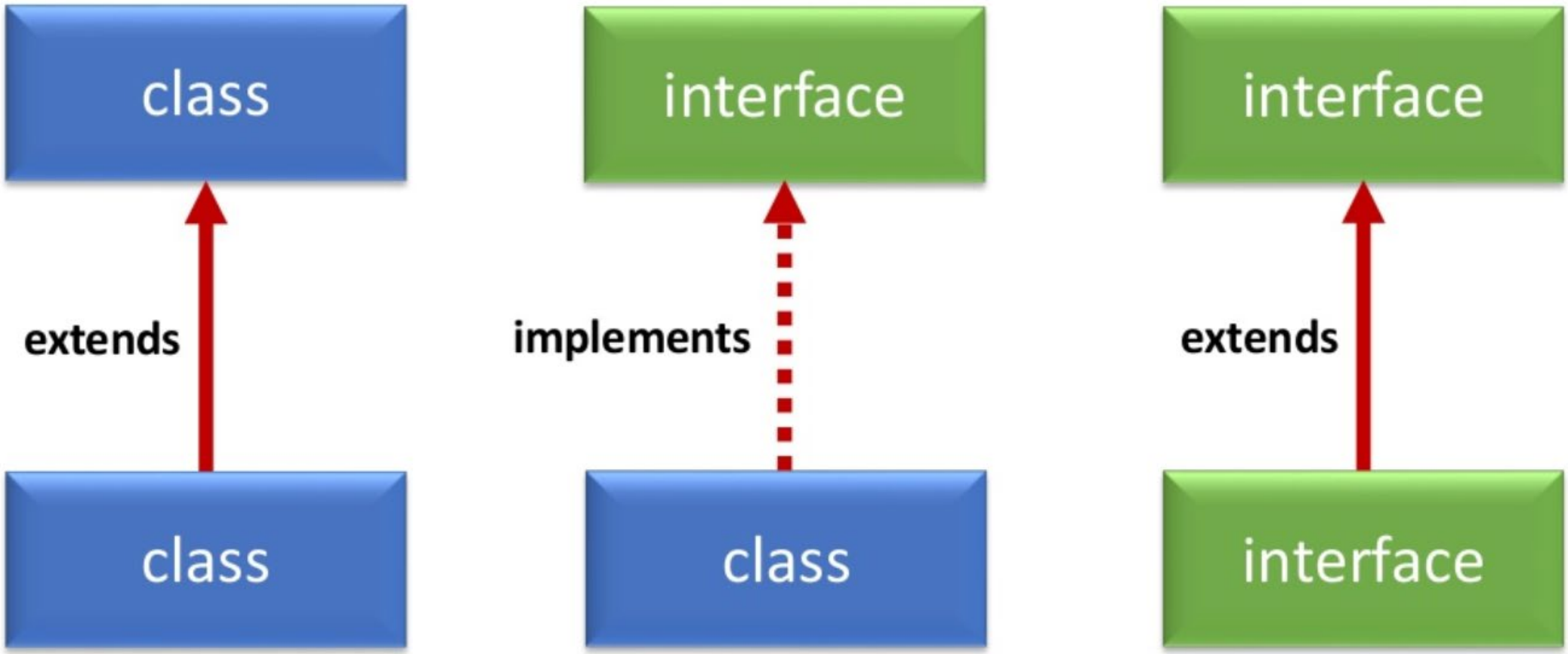
```
interface Interface_Name {
  // Any number of final, static fields
  // Any number of abstract method declarations
}
```

- **Example**

```
interface Moveable {
 int AVERAGE-SPEED=40; //fields public static final by default
 void move();  //methods are public abstract by default
}
```

Understanding Relationship between Classes and Interfaces

- As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



Implementing Interfaces

- When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.
- A class uses the **implements** keyword to implement an interface.
- Class provide the body of all the methods that are declared in interface.
- The implements keyword appears in the class declaration following the extends portion of the declaration.

Example of Interface implementation

```
interface Person{
  void msg();
  }
class Employee implements Person{
  //implementation of msg method in class
  public void msg(){
    System.out.println("i am Person");
  }
public static void main(String args[]){
   Employee obj = new Employee();
   obj.msg();
 }
}
```

Example of Interface implementation

```
interface Person{
  void msg();
  }
class Employee implements Person{
  //implementation of msg method in class
  public void msg(){
    System.out.println("i am Person");
  }
public static void main(String args[]){
   Employee obj = new Employee();
   obj.msg();
 }
}
```

Output is:

i am Person

Another Example of Interface

```
interface Moveable {
    int AVG_SPEED = 60;
    void move();
}
class Vehicle implements Moveable  {
   public void move() {
   System.out.println("Average speed is: "+AVG_SPEED);
  }
 public static void main (String[] arg)  {
   Vehicle v = new Vehicle();
   v.move();
 }
}
```

Another Example of Interface

```
interface Moveable {
    int AVG_SPEED = 60;
    void move();
}
class Vehicle implements Moveable  {
   public void move() {
   System.out.println("Average speed is: "+AVG_SPEED);
  }
 public static void main (String[] arg)  {
   Vehicle v = new Vehicle();
   v.move();
 }
}
```

By default this variable is public static final

Output is:

Average speed is:60

Interface in Java

- **When we use Abstract and when Interface**
- If you have a lot of methods and want default implementation for some of them, then go with abstract class
- If we do not know about any things about implementation just we have requirement specification then we should be go for **Interface**
- If we are talking about implementation but not completely (partially implemented) then we should be go for **abstract.**
- If your base contract keeps on changing, then you should use abstract class, as if you keep changing your base contract and use interface, then you have to change all the classes which implements that interface.

Rules for implementation interface

- **When overriding methods defined in interfaces, there are several rules to be followed:**
- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so, interface methods need not be implemented.

Rules for implementation interface

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.
- Interface cannot implement a class.
- Interface can be nested inside another interface.

Extending Interfaces

- An interface can extend another interface in the same way that a class can extend another class.
- The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.
- **Syntax :**

```
Child_Interface_Name extends Parent_Interface_Name
```

- **Example :**

```
public interface Football extends Sports
```

Extending Interfaces

```
public interface Vehicles{
   public void hasWheels();
   public void hasEngine();
}
public interface Car extends Vehicles{
   public void hasDoors();
   public void hasAirbags();
   public void hasRoof();
}
public interface Motorcycle extends Vehicles{
   public void hasPedal();
   public void hasHandlebars();
   public void hasStand();
}
```

Extending Interfaces

```
public interface Vehicles{
   public void hasWheels();
   public void hasEngine();
}
public interface Car extends Vehicles{
   public void hasDoors();
   public void hasAirbags();
   public void hasRoof();
}
public interface Motorcycle extends Vehicles{
   public void hasPedal();
   public void hasHandlebars();
   public void hasStand();
}
```

Parent interface

Child interfaces

Extending Interfaces

```
public interface Vehicles{
   public void hasWheels();
   public void hasEngine();
}
public interface Car extends Vehicles{
   public void hasDoors();
   public void hasAirbags();
   public void hasRoof();
}
public interface Motorcycle extends Vehicles{
   public void hasPedal();
   public void hasHandlebars();
   public void hasStand();
}
```

In this example, we can see the way the Car and Motorcycle interfaces extend the Vehicles interface. The Vehicles interface contains two methods: **hasWheels()** and **hasEngine()**. Any class that implements interface Vehicles must meet these two requirements.

Extending Interfaces

```
public interface Vehicles{
   public void hasWheels();
   public void hasEngine();
}
public interface Car extends Vehicles{
   public void hasDoors();
   public void hasAirbags();
   public void hasRoof();
}
public interface Motorcycle extends Vehicles{
   public void hasPedal();
   public void hasHandlebars();
   public void hasStand();
}
```

Simply we can say If a class implements interface Vehicles, then it needs to implement two methods.

And If a class implements interface Car, then it needs to implement five methods.(two inherit from parent vehicles)

And If a class implements interface Motorcycle, then it needs to implement five methods.(two inherit from parent vehicles)

Example of Extending Interfaces-1

```
interface Interface1 {
    public void f1();
}
 //Interface2 extending Interface1
interface Interface2 extends Interface1 {
   public void f2();
}
class A implements Interface2 {
    public void f1() {
      System.out.println("Contents of Interface1");   }
   public void f2() {
      System.out.println("Contents of Interface2");   }
    public void f3() {
      System.out.println("Contents of Class X");    }
}
```

Example of Extending Interfaces-2

```
class ExtendingInterface
{
    public static void main(String[] args)
    {
     Interface2 v2; //Reference variable of Interface2
     v2 = new A();  //assign object of class A
     v2.f1();
     v2.f2();
     A x1=new A();
     x1.f3();
    }
}
```

Output is:

Contents of Interface1
Contents of Interface2
Contents of Class X

Example of Extending Interfaces

```
interface RollNoDetails {
   void rollNo(); }
interface PersonDetails extends RollNoDetails {
   void name();  }
public class Details implements PersonDetails{
   public void name(){
      System.out.println("Adil");   }
   public void rollNo(){
      System.out.println("1122");  }
   public static void main(String[] args) {
      Details details = new Details();
      System.out.print("Name: ");
      details.name();
      System.out.print("Roll No: ");
      details.rollNo();  }
}
```

Output is:

Name: Adil

Roll No: 1122

Valid use of Interfaces

```
interface InterfaceA {
  void msg();
}
class ClassB implements InterfaceA {
   public void msg(){
    System.out.println("Hello Java");
   }
}
class ClassA extends ClassB {
    public static void main(String[]args)
    {
        ClassA a = new ClassA();
        a.msg();
    }
}
```

interface is implemented and then other Class is Extended

Output is:

Hello Java

Another Valid use of Interfaces

```
interface InterfaceA {
  void msg();
}
abstract class ClassB implements InterfaceA {
}
class ClassA extends ClassB {
    public void msg(){
    System.out.println("Hello Java");
   }
    public static void main(String[]args)
    {
        ClassA a = new ClassA();
        a.msg();
    }
}
```

In this way we must use abstract keyword Here

Output is:

Hello Java

Interface and Method Overriding

```
interface Vehicle {
    public void showSpeed();
}
 class Car implements Vehicle {
    public long MAX_SPEED = 200;
    public void showSpeed() {
        System.out.println("Speed implementation in Car class");    }
}
public class Audi extends Car {
    public long MAX_SPEED = 250;
    public void showSpeed() {
        System.out.println("Speed implementation in Audi class");  }
    public static void main(String[] args) {
        Audi c1 = new Audi();
        c1.showSpeed();
        System.out.println("Audi's maximum speed is: " + c1.MAX_SPEED);   }
}
```

Interface and Method Overriding

```
interface Vehicle {
    public void showSpeed();
}
 class Car implements Vehicle {
    public long MAX_SPEED = 200;
    public void showSpeed() {
        System.out.println("Speed implementation in Car class");    }
}
public class Audi extends Car {
    public long MAX_SPEED = 250;
    public void showSpeed() {
        System.out.println("Speed implementation in Audi class");  }
    public static void main(String[] args) {
        Audi c1 = new Audi();
        c1.showSpeed();
        System.out.println("Audi's maximum speed is: " + c1.MAX_SPEED);   }
}
```

Interface and Method Overriding

```
interface Vehicle {
    public void showSpeed();
}
 class Car implements Vehicle {
    public long MAX_SPEED = 200;
    public void showSpeed() {
        System.out.println("Speed implementation in Car class");    }
}
public class Audi extends Car {
    public long MAX_SPEED = 250;
    public void showSpeed() {
        System.out.println("Speed implementation in Audi class");  }
    public static void main(String[] args) {
        Audi c1 = new Audi();
        c1.showSpeed();
        System.out.println("Audi's maximum speed is: " + c1.MAX_SPEED);   }
}
```

Output is:

Speed implementation in Audi class
Audi's maximum speed is: 250

Extending Multiple Interfaces

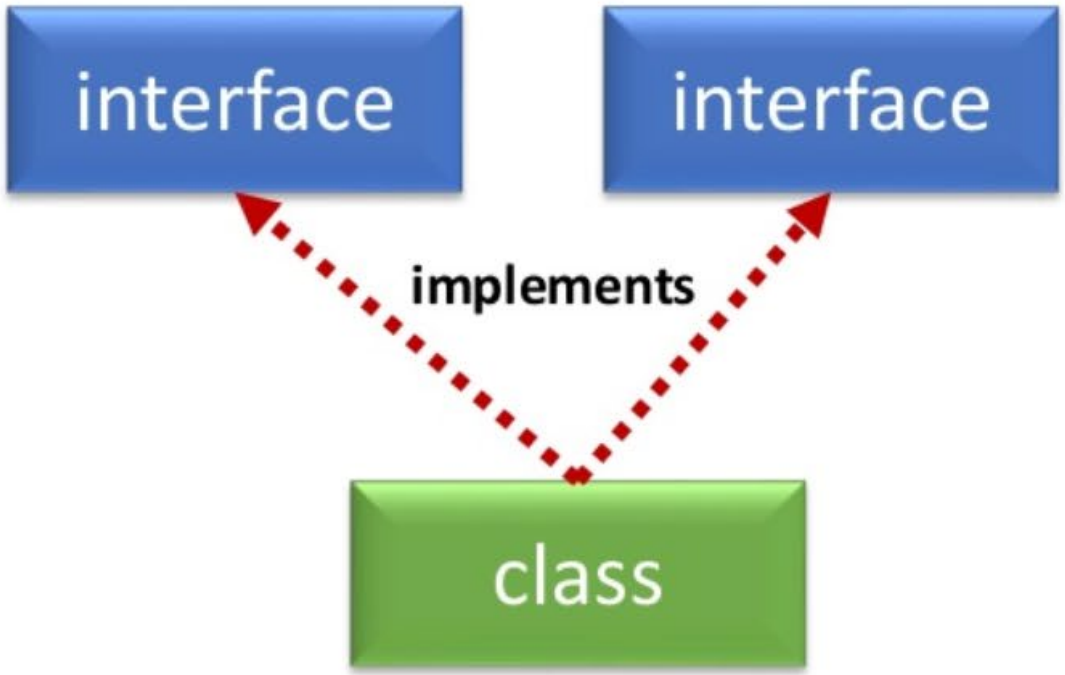
- A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.
- The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

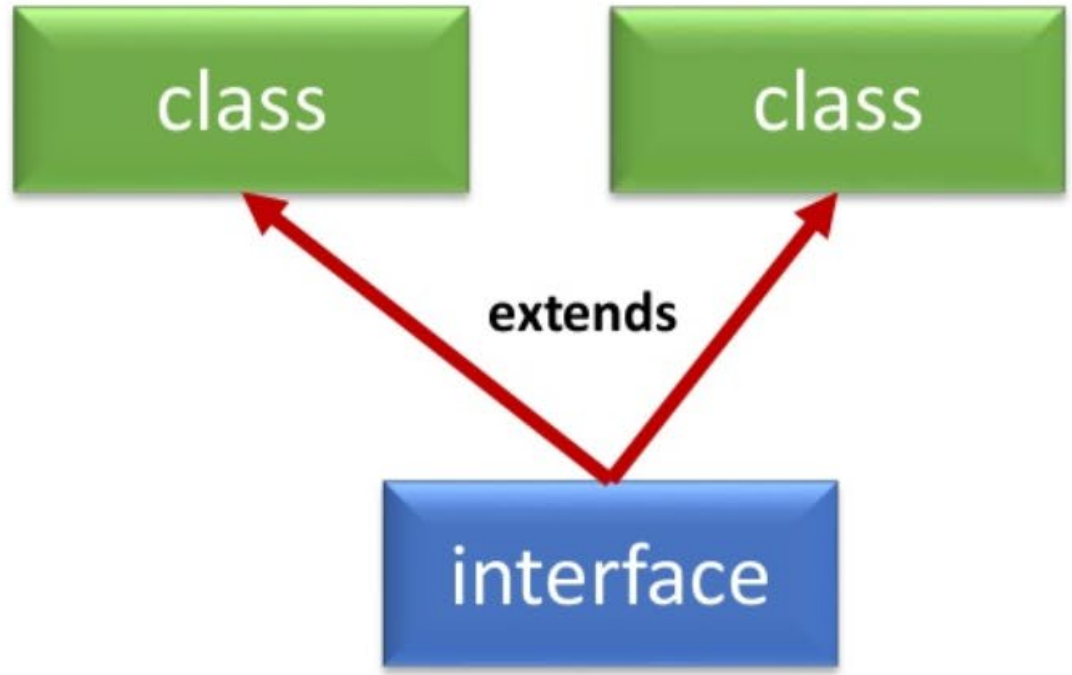
```
public interface Car extends Vehicles, Belongings, Transit
```

- The above example indicates that the child interface Car is an extension of the Vehicles interface we declared previously, as well as an interface called Belongings, and an interface called Transit. Any methods declared in the Belongings and Transit interfaces will be inherited by the child interface Car.

Multiple inheritance in Java by interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.





Multiple inheritance in Java by interface

```
interface Printable{
  void print();
}
interface Showable{
  void show();
}
class A implements Printable, Showable{
   public void print(){System.out.println("Hello");}
   public void show(){System.out.println("Java");}
public static void main(String args[]){
  A obj = new A();
  obj.print();
  obj.show();
 }
}
```

Multiple inheritance in Java by interface

```
interface Printable{
  void print();
}
interface Showable{
  void show();
}
class A implements Printable, Showable{
   public void print(){System.out.println("Hello");}
   public void show(){System.out.println("Java");}
public static void main(String args[]){
  A obj = new A();
  obj.print();
  obj.show();
 }
}
```

Here class implements more than two interfaces

Multiple Inheritance Example with interface-1

```
interface Father{
    int prop1 = 5000;
    float ht1 = 6.0f;
}
interface Mother{
    int prop2 = 5000;
    float ht2 = 5.6f;
}
class Child implements Father , Mother{
    void property(){
        System.out.println("Child property is = "+(prop1+prop2));  }
    void height(){
        System.out.println("Child height is = "+(ht1+ht2)/2); }
}
```

Multiple Inheritance Example with interface-2

```
public class Test {
    public static void main(String[] args) {
        Child ch = new Child();
        ch.property();
        ch.height();
    }
}
```

Output is:

Child property is = 10000

Child height is = 5.8

interface in Java

- Multiple inheritance is not supported through class in java but it is possible by interface, why?
- As we have explained in the inheritance topic, multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.
- Let see an example in next slide.

Multiple inheritance in Java by interface

```
interface Printable{
  void print();
}
interface Showable{
  void print();
}
class Test implements Printable , Showable{
   public void print(){System.out.println("Hello Java");}
public static void main(String args[]){
  Test obj = new Test();
  obj.print();
 }
}
```

Multiple inheritance in Java by interface

```
interface Printable{
  void print();
}
interface Showable{
  void print();
}
class Test implements Printable , Showable{
    public void print(){System.out.println("Hello Java");}
public static void main(String args[]){
   Test obj = new Test();
   obj.print();
 }
}
```

Multiple inheritance in Java by interface

```
interface Printable{
  void print();
}
interface Showable{
  void print();
}
class Test implements Printable , Showable{
   public void print(){System.out.println("Hello Java");}
public static void main(String args[]){
  Test obj = new Test();
  obj.print();
 }
}
```

One class implements two interfaces

Multiple inheritance in Java by interface

```
interface Printable{
  void print();
}
interface Showable{
  void print();
}
class Test implements Printable , Showable{
   public void print(){System.out.println("Hello Java");}
public static void main(String args[]){
  Test obj = new Test();
  obj.print();
 }
}
```

Output is:
Hello Java

Class Extends Another Class and Implements Another Interface

```
interface Moveable {
    public void move();  }
class Car {
    protected void start() {
        System.out.println("Starting...");     }
    public void stop() {
        System.out.println("Stopping...");    }
}
class Truck extends Car implements Moveable {
    public void move() {
        System.out.println("Moving...");   }
 public static void main(String[] args) {
   Truck m = new Truck();
   m.start();
   m.stop();
   m.move();  }
}
```

Output is:

Starting...
Stopping...
Moving...

What is marker or tagged interface?

- An interface that have no member is known as marker or tagged interface. In simple words we can say that **empty interface in java** is called **marker interface**.
- For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
//Way of writing Serializable interface
public interface Serializable
{
}
```

What is marker or tagged interface?

- Basically Tag interfaces are meaningful to the JVM (Java virtual machine). we can also create your own tag interfaces to segregate and categorize your code. It would improve the readability of your code.
- **Some Examples of Java Marker Interface**
- java.lang.Cloneable
- java.util.EventListener
- java.io.Serializable
- java.rmi.Remote

interface in Java

- **Why Interface have no Constructor ?**
- Because, constructor are used for eliminate the default values by user defined values, but in case of interface all the data members are public static final that means all are constant so no need to eliminate these values.
- Other reason because constructor is like a method and it is concrete method and interface does not have concrete method it have only abstract methods that's why interface have no constructor.

interface in Java-1

- **Why Constructors are not Allowed in Interface?**
- As we know that all the methods in interface are public abstract by default which means the method implementation cannot be provided in the interface itself. It has to be provided by the implementing class.

```
interface SumInterface{
   public int sum(int num1, int num2);
}
public class ClassSum implements SumInterface{
   public int sum(int num1, int num2){
         return num1+num2;
   }
   public static void main(String args[]) {
         ClassSum obj= new ClassSum();
         System.out.println(obj.sum(2, 3));   }
}
```

Difference between an interface and an Abstract class?

Abstract class	Interface
It is collection of abstract method and concrete methods.	It is collection of abstract method.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
There properties can be reused commonly in a specific application.	There properties commonly usable in any application of java environment.
Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.

Difference between an interface and an Abstract class?

Abstract class	Interface
Abstract class can have static methods, main method and constructor.	Interface can't have static methods, main method or constructor.
The default access specifier of abstract class methods are default.	There default access specifier of interface method are public.
These class properties can be reused in other class using extend keyword.	These properties can be reused in any other class using implements keyword.
Inside abstract class we can take constructor.	Inside interface we can not take any constructor.

Difference between an interface and an Abstract class?

Abstract class	Interface
Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
There are no any restriction for abstract class variable.	For the interface variable can not declare variable as private, protected, transient, volatile.
There are no any restriction for abstract class method modifier that means we can use any modifiers.	For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Difference between an interface and an Abstract class?

Abstract class	Interface
A class can extend only one abstract class	A class can implement any number of interfaces
abstract class can extend from a class or from an abstract class	interface can extend only from an interface

Difference with Program Example

- **Difference No.1:**
- Abstract class can extend only one class or one abstract class at a time
- Interface can extend any number of interfaces at a time

- **Difference No.2:**
- Abstract class can be inherited by a class or an abstract class
- Interfaces can be extended only by interfaces. Classes has to implement them instead of extend

Difference with Program Example

- **Difference No.3**
- Abstract class can have both abstract and concrete methods
- Interface can only have abstract methods, they cannot have concrete methods

- **Difference No.4**
- A class can extend only one abstract class at a time
- A class can implement any number of interfaces at a time

Difference with Program Example

- **Difference No.5**
- In abstract class, the keyword ‘abstract’ is mandatory to declare a method as an abstract
- In interfaces, the keyword ‘abstract’ is optional to declare a method as an abstract because all the methods are abstract by default

- **Difference No.6**
- Abstract class can have protected , public and public abstract methods
- Interface can have only public abstract methods i.e. by default

Interface can extend any number of interfaces at a time-1

Difference No. 1

```
//first interface
interface Test1{
    public void msg1();
}
//second interface
interface Test2 {
    public void msg2();
}
//This interface is extending both the above interfaces
interface Test3 extends Test1,Test2{
}
```