Abstraction in Java

Abstraction in Java

1. The process of hiding complex internal implementation details from users and providing only necessary functionality is called **Abstraction**.

2. In other words, abstraction in Java is a technique by which we can hide the data that is not required to a user. It hides all unwanted data so that users can work only with the required data.

3. The main purpose of abstraction is to represent the essential features without including the background details.

Examples of Abstraction

How to Achieve Abstraction in Java?

There are two ways to achieve or implement abstraction in Java program.

They are as follows:

1) **Abstract class (0 to 100%)**
2) **Interface (100%)**

- Java allows us to implement multiple levels of abstraction for a Java project.
- Abstract class and interface are the most common ways to achieve abstraction in Java.

Abstract Class in Java

An abstract class in Java is a class, which is declared with an abstract keyword.

It is just like a normal class but has two differences.

1. We cannot create an object of this class. Only objects of its non-abstract (or concrete) sub-classes can be created.
2. It can have zero or more abstract methods which are not allowed in a non-abstract class (concrete class).

ClassLoader class is a good example of an abstract class that does not have any abstract methods.

Key points to remember

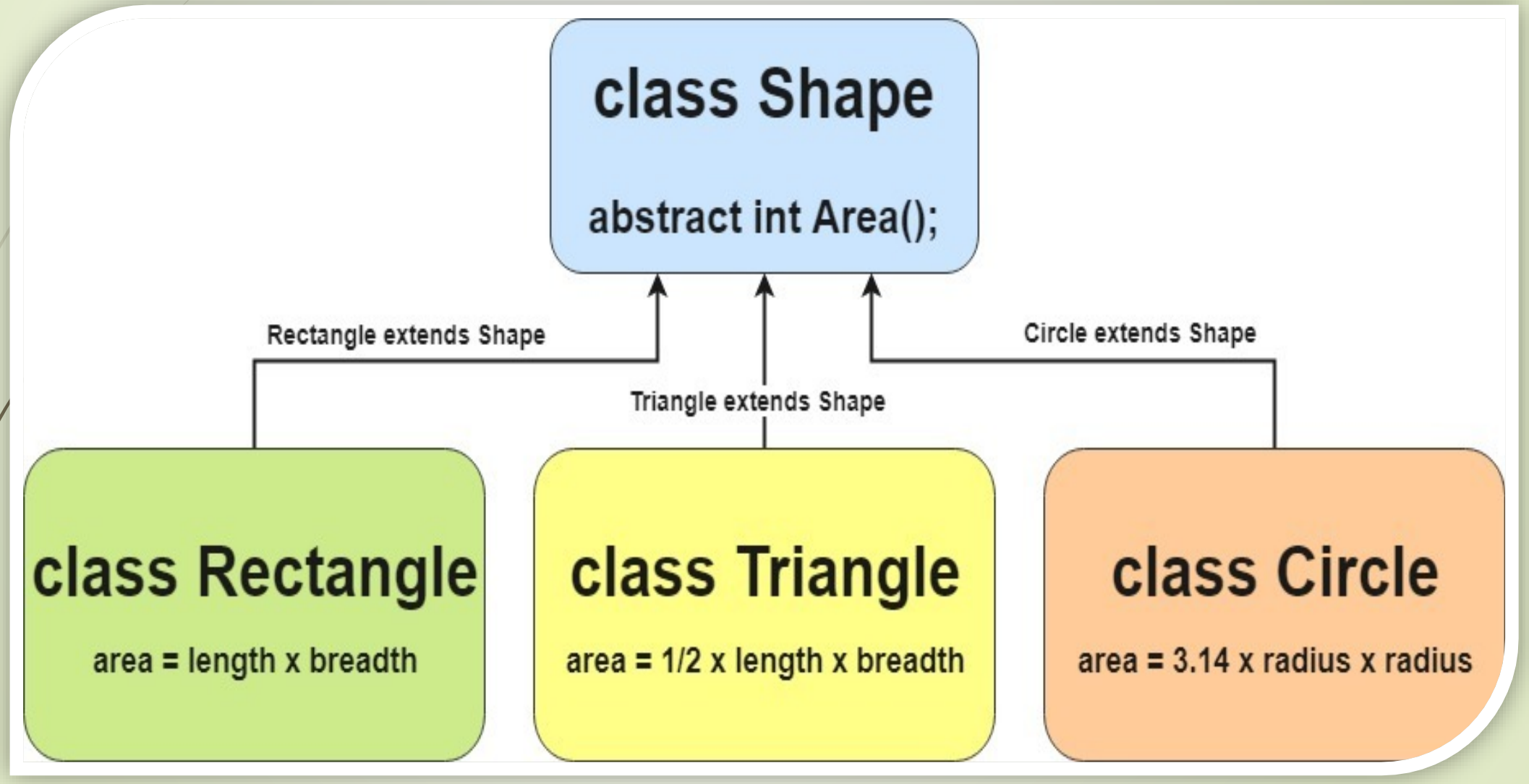
1. abstract is a non-access modifier in Java which is applicable for classes, interfaces, methods, and inner classes.
2. It represents an incomplete class that depends on subclasses for its implementation.
3. Creating subclass is compulsory for abstract class.
4. A non-abstract class is sometimes called a **concrete class**.
5. An abstract concept is not applicable to variables.

When to Use Abstract class in Java ?

An abstract class can be used when we need to share the same method to all non-abstract subclasses with their own specific implementations.

Moreover, the common member of the abstract class can also be shared by the subclasses. Thus, abstract class is useful to make the program more flexible and understandable.

Abstract Method in Java



Abstract Method in Java

A method that is declared with abstract modifier in an abstract class and has no implementation (means no body) is called abstract method in Java. It does not contain any body.

Abstract method has simply a signature declaration followed by a semicolon. It has the following general form as given below:

Syntax:

```
abstract type methodName(arguments);    // No body
```

For example:

```
abstract void msg();     // No body.
```

Important Points about Abstract Classes & Abstract Methods in Java

1. In Java, an instance of an abstract class cannot be created, we can have references of abstract class type.

2. An abstract class can contain constructors in Java. And a constructor of abstract class is called when an instance of a inherited class is created.

3. An abstract class can contain constructors in Java. And a constructor of abstract class is called when an instance of a inherited class is created.

4. Abstract classes can also have final methods (methods that cannot be overridden).

Important Points about Abstract Classes & Abstract Methods in Java

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

6. If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

7. Abstract class allows to define **private, final, static and concrete methods.** Everything is possible to define in an abstract class as per application requirements.

8. Abstract class does not support multiple inheritance in Java but allows in interfaces.

9. It can implement one or more interfaces in Java.

Important Points about Abstract Classes & Abstract Methods in Java

10. Method name and signature must be the same as in the abstract class.

11. The visibility of the method in the subclass cannot be reduced while overriding abstract method.

12. Abstract method cannot be **static** or **final**.

13. It cannot be private because the abstract method must be implemented in the subclass. If we declare it private, we cannot implement it from outside the class.

FAQ

Question 1: Why abstract class has constructor even though we cannot create object?

Answer:

We cannot create an object of abstract class but we can create an object of subclass of abstract class. When we create an object of subclass of an abstract class, it calls the constructor of subclass.

This subclass constructor has super in the first line that calls constructor of an abstract class. Thus, the constructors of an abstract class are used from constructor of its subclass.

FAQ

Question 2 : Why should we create reference to superclass (abstract class reference)?

Answer:

We should create a reference of the superclass to access subclass features because superclass reference allows only to access those features of subclass which have already declared in superclass.

If you create an individual method in subclass, the superclass reference cannot access that method. Thus, any programmer cannot add their own additional features in subclasses other than whatever is given in superclass.

Advantage of Abstract class in Java

1. Abstract class makes programming better and more flexible by giving the scope of implementing abstract methods.
2. Programmers can implement an abstract method to perform different tasks depending on the need.
3. We can easily manage code.

Interface in Java

Interface in Java

1. An interface in Java is syntactically similar to a class but can have only abstract methods declaration and constants as members.

2. In other words, an interface is a collection of abstract methods and constants (i.e. static and final fields).

3. It is used to achieve complete abstraction.

4. Every interface in Java is abstract by default.

5. So, it is not compulsory to write abstract keyword with an interface.

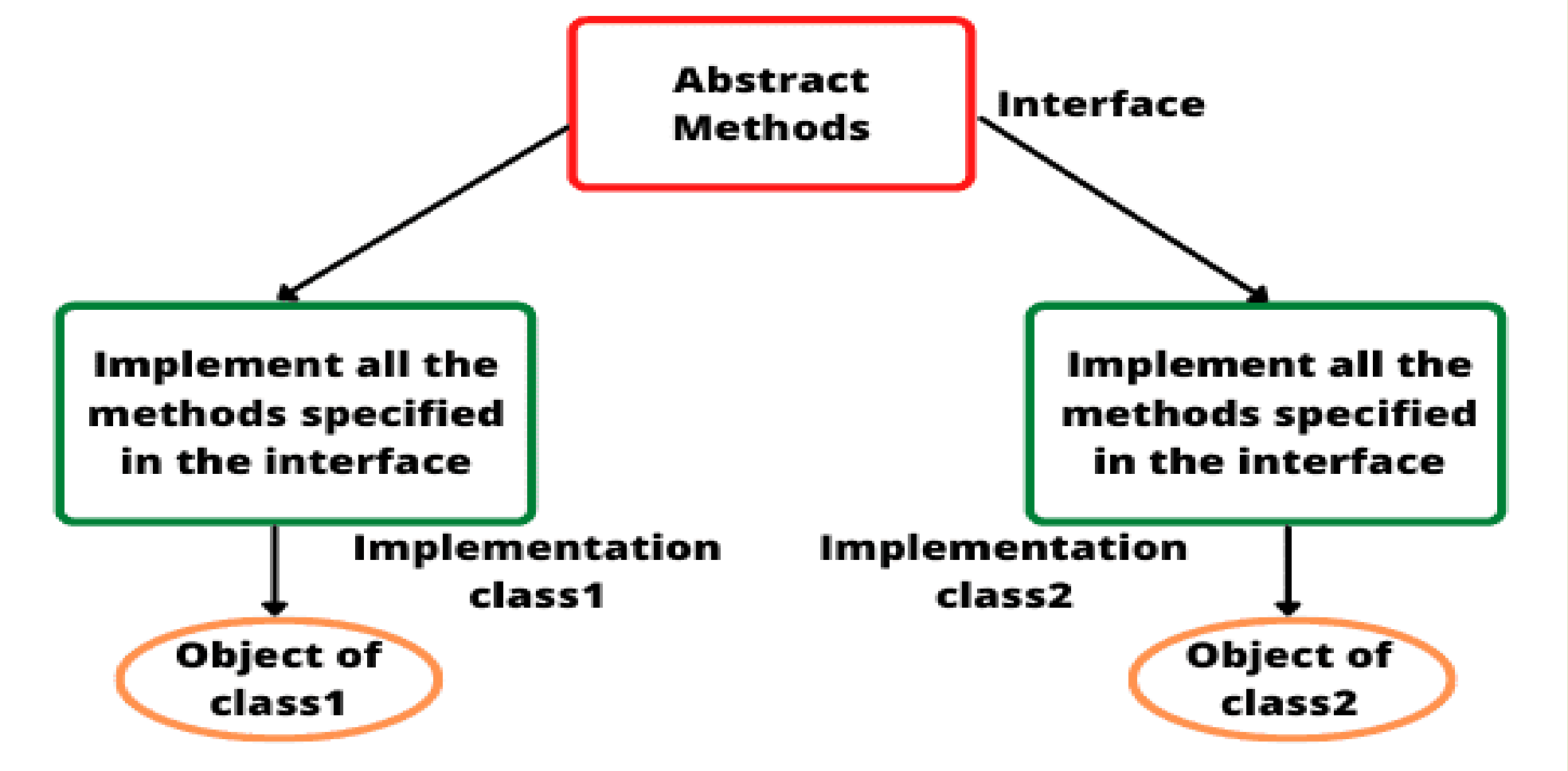
Interface in Java

6. A class that implements an interface is called implementation class.

7. A class can implement any number of interfaces in Java.

8. Every implementation class can have its own implementation for abstract methods specified in the interface

implements an interface and Implementation Classes



Why do we need Interface?

There are mainly five reasons or purposes of using an interface in Java.

1. In industry, architect-level people create interfaces, and then they are given to developers for writing classes by implementing interfaces provided.

2. Using interfaces is the best way to expose our project's API to some other projects.

In other words, we can provide interface methods to the third-party vendors for their implementation.

For Example, HDFC bank can expose methods or interfaces to various shopping carts.

Why do we need Interface?

3. Programmers use interface to customize features of software differently for different objects.

4. It is used to achieve full abstraction in java.

5. By using interfaces, we can achieve the functionality of **multiple inheritance.**

Syntax for Interface in Java

Syntax:

```
accessModifier  interface  interfaceName
{
  // declare constant fields.
  // declare methods that abstract by default.
}
```

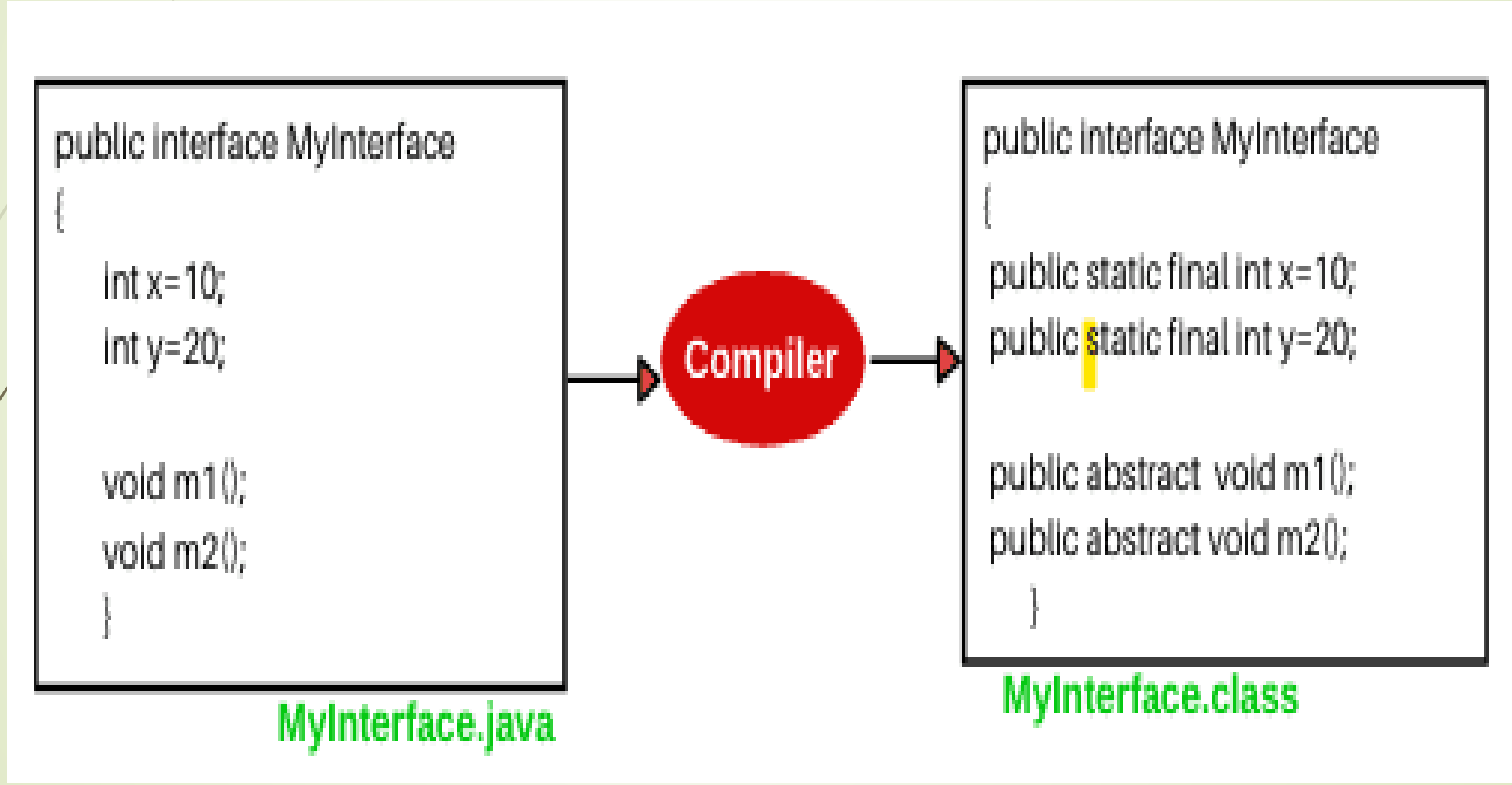
Example:

```
public abstract interface MyInterface
{
  int x = 10; // public static final keyword added by the compiler automatically.
  void m1(); // public abstract keywords added by the compiler automatically.
  void m2();
}
```

Points to remember:

- ✓ Java compiler automatically adds public and abstract keywords before to all interface methods.
- ✓ Moreover, it also adds **public**, **static**, and **final** keywords before interface variables.
- ✓ Therefore, all the variables declared in an interface are considered as **public**, **static**, and **final** by default and acts like **constant**.
- ✓ We cannot change their value once they initialized.

Example


public interface MyInterface
{
int x=10;
int y=20;
void m1();
void m2();
}
MyInterface.java
Compiler
public interface MyInterface
{
public static final int x=10;
public static final int y=20;
public abstract void m1();
public abstract void m2();
}
MyInterface.class

Points to be noted:

1. Earlier to Java 8, an interface could not define any implementation whatsoever. An interface can only declare abstract methods.

2. Java 8 changed this rule. From Java 8 onwards, it is also possible to add a default implementation to an interface method.

3. To support lambda functions, Java 8 has added a new feature to interface. We can also declare default methods and static methods inside interfaces.

4. From Java 9 onwards, an interface can also declare private methods.

Features of Interface

There are the following features of an interface in Java. They are as follows:

1. Interface provides pure abstraction in Java. It also represents the Is-A relationship.

2. It can contain three types of methods: abstract, default, and static methods.

3. All the (non-default) methods declared in the interface are by default abstract and public. So, there is no need to write abstract or public modifiers before them.

4. The fields (data members) declared in an interface are by default public, static, and final. Therefore, they are just public constants. So, we cannot change their value by implementing class once they are initialized.

Features of Interface

5. Interface cannot have constructors.

6. The interface is the only mechanism that allows achieving multiple inheritance in java.

7. A Java class can implement any number of interfaces by using keyword implements.

8. Interface can extend an interface and can also extend multiple interfaces.

Rules of Interface in Java

Here are some key points for defining an interface in Java that must be kept in mind. The rules are as follows:

1. An interface cannot be instantiated directly. But we can create a reference to an interface that can point to an object of any of its derived types implementing it.

2. An interface may not be declared with **final** keyword.

3. It cannot have instance variables. If we declare a variable in an interface, it must be initialized at the time of declaration.

4. A class that implements an interface, must provide its own implementations of all the methods defined in the interface.

Rules of Interface in Java

5. We cannot reduce the visibility of an interface method while overriding. That is, when we implement an interface method, it must be declared as public.

6. It can also be declared with an empty body (i.e. without any members). For example, **java.util** package defines **EventListener** interface without a body.

7. An interface can be declared within another interface or class. Such interfaces are called **nested interfaces** in Java.

8. A top-level interface can be **public** or **default** with the abstract modifier in its definition. Therefore, an interface declared with **private**, **protected**, or **final** will generate a compile-time error.

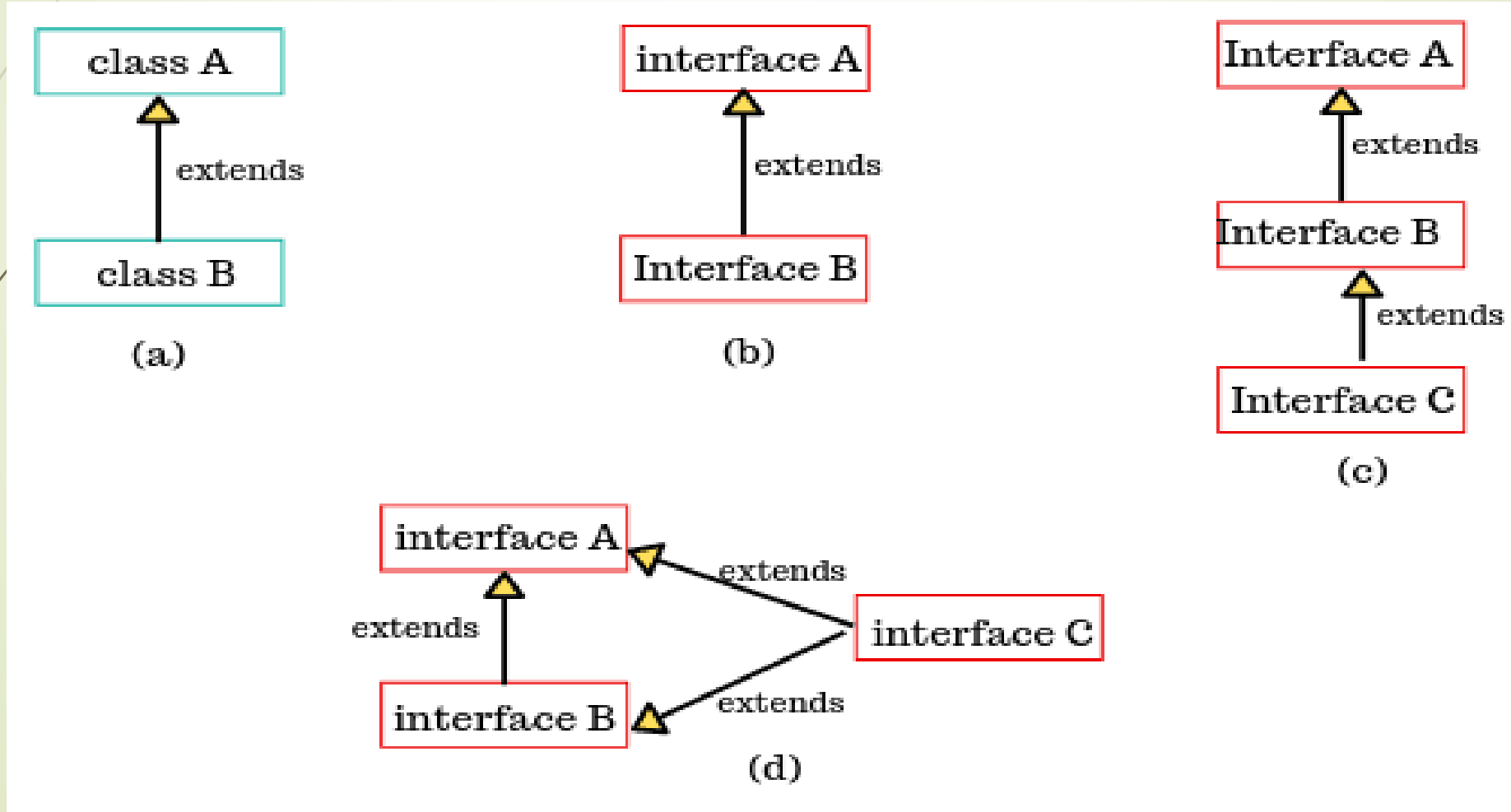
Rules of Interface in Java

9. All non-default methods defined in an interface are abstract and public by default. Therefore, a method defined with **private**, **protected**, or **final** in an interface will generate **compile-time error**.

10. If you add any new method in interface, all concrete classes which implement that interface must provide implementations for newly added method because all methods in interface are by default abstract.

Extending Interface in Java

Like classes, an interface can also extend another interface. This means that an interface can be sub interfaces from other interfaces.



Key Points to remember:

1. An interface cannot extend classes because it would violate rules that an interface can have only abstract methods and constants.

2. An interface can extend Interface1, Interface2.

3. All methods of interfaces when implementing in a class must be declared as public otherwise, you will get a compile-time error if any other modifier is specified.

4. Class extends class implements interface.

5. Class extends class implements Interface1, Interface2…

The implementation of interfaces can have the following general forms

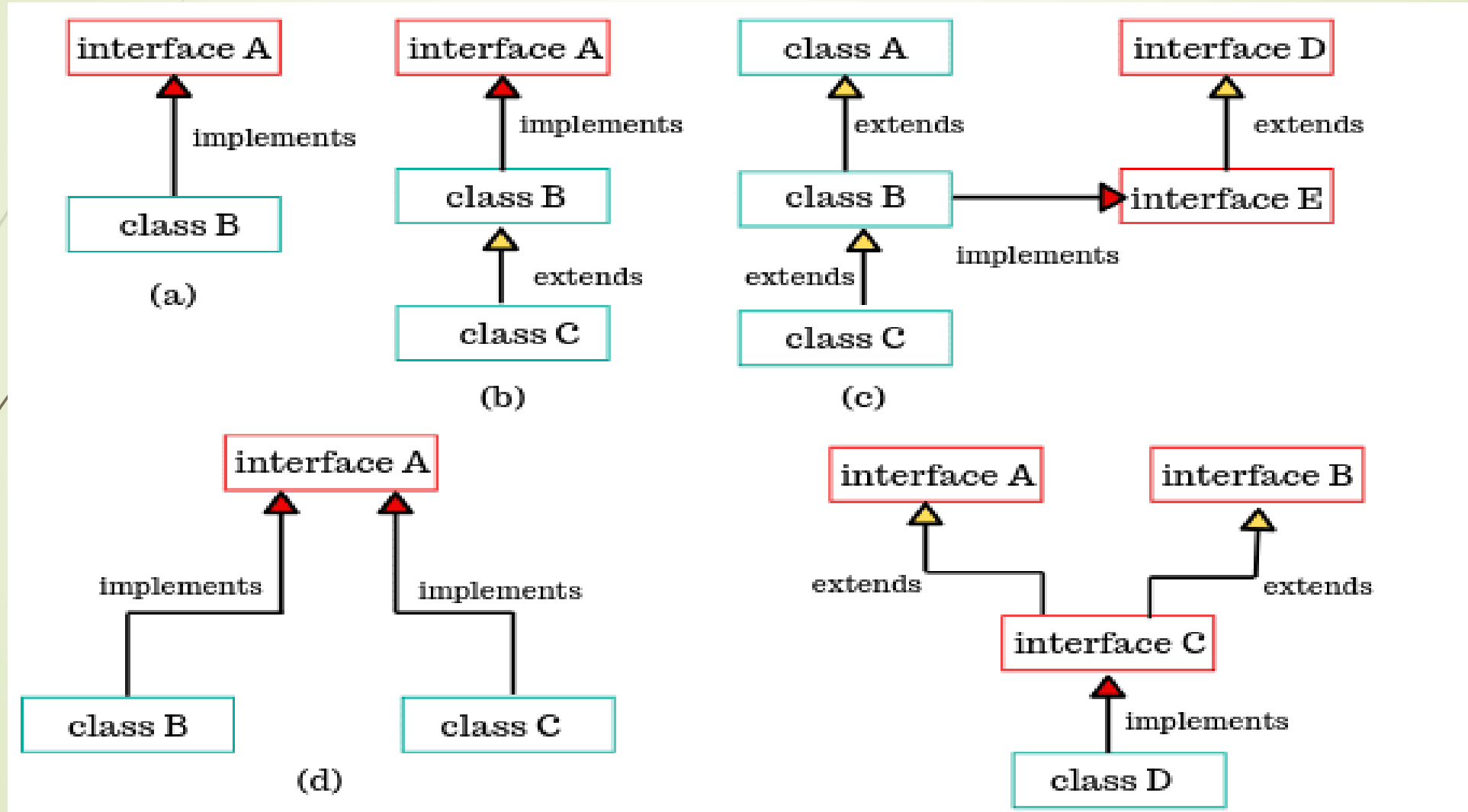


Fig: Various forms of interface implementation

Polymorphism in Java Interface

When two or more classes implement the same interface with different implementations, then through the object of each class, we can achieve polymorphic behavior for a given interface.

This is called **polymorphism** in interface.