Links: [[02 Polymorphism]], [[01 Inheritance]] ____ # Abstraction It is a concept which is used for **hiding implementation complexity** and showing only the essential features or functionality to the user.

(Note: **Encapsulation** is used for *data hiding* (protecting variables). **Abstraction** is used for *implementation hiding* (hiding *how* a method works)).

Abstraction focuses on *what* an object does, not *how* it does it.

Abstraction can be achieved using an abstract class or by an interface.

### Abstract Class

We use `abstract` keyword to declare a class as abstract. An abstract class provides a template for other classes, forcing them to implement certain methods.

You **cannot** create an object (instantiate) of an abstract class.

- We can declare abstract as well as non-abstract (concrete) methods.

- It can have constructors, static methods, and `final` methods.
- It works together with [[02 Polymorphism#Run-Time Polymorphism (Method Overriding)]].

Abstract methods are declared with the `abstract` keyword. These methods don't have a definition or body in the class in which they are declared. They **must** be overridden by a subclass to provide the definition.

```java
// We cannot create an object of 'Shape'
abstract class Shape {
    int color;

    // Concrete (non-abstract) method with a body
    void setColor(int c) {
        this.color = c;
    }

    // Abstract method - no body
    // Forces subclasses to provide their own version
    abstract void draw();
}

class Circle extends Shape {
    // We MUST implement the abstract 'draw' method
    @Override
    void draw() {
        System.out.println("Drawing a circle");
    }
}
```

**Interface**

An interface is a blueprint of a class. It specifies what a class must do, but not how.

It is used to achieve 100% abstraction (before Java 8).

- We use the `interface` keyword.
- All methods in an interface are `public` and `abstract` by default.
- All variables (fields) are `public`, `static`, and `final` (constants) by default.
- A class uses the `implements` keyword to use an interface.
- A class can implement **multiple** interfaces. This is how Java achieves multiple inheritance.

```java
interface Drawable {
    void draw(); // This is public and abstract by default
}

interface Loggable {
    void log(String message);
}

// A class can implement multiple interfaces
class Circle implements Drawable, Loggable {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }

    @Override
    public void log(String message) {
        System.out.println("LOG: " + message);
    }
}
```

**`default` and `static` methods** Java 8 allowed interfaces to have methods with implementation:

- **`default` methods:** A class can override them, but doesn't have to. It allows adding new functionality to an interface without breaking existing classes.
- **`static` methods:** Utility methods that are part of the interface, not a specific object.

**Default Methods** Allows adding new methods to interfaces with a default implementation, without breaking existing classes that implement the interface.

We use the default keyword.

```java
interface MyInterface {
    void existingMethod();

    default void newDefaultMethod() {
        System.out.println("This is a default implementation.");
    }
}

class MyClass implements MyInterface {
    public void existingMethod() {
        // ...
    }
    // No need to implement newDefaultMethod(), it's optional.
}
```

**Static Method** Interfaces can now have static methods. These are utility methods that are part of the interface, not the implementing class.

They are called on the interface itself, not on an instance.

```java
interface MyInterface {
    static void utilityMethod() {
        System.out.println("This is a static utility method.");
    }
}

// How to call it:
MyInterface.utilityMethod();
```

### Abstract Class vs. Interface

- **Methods:** An abstract class can have both abstract and concrete methods. An interface (pre-Java 8) can only have abstract methods.
- **Variables:** An abstract class can have any type of variable (instance, static, final). An interface can only have `public static final` constants.
- **Inheritance:** A class can `extends` only **one** abstract class. A class can `implements` **many** interfaces.
- **Constructor:** An abstract class can have a constructor (which is called by the subclass constructor). An interface **cannot** have a constructor.
- **Purpose:**
  - Use an **abstract class** for an “IS-A” relationship, when you want to provide common, shared code for all subclasses (e.g., `Dog` IS-A `Animal`, and all animals `breathe()`).

- Use an **interface** for a “CAN-DO” relationship, when you want to define a capability or contract that unrelated classes can share (e.g., `Circle` CAN-DO `Drawable`, `Car` CAN-DO `Drawable`).