**Chapter # 7: Interfaces**

**7.1 Interface:**

In Java, an interface is a collection of abstract methods (methods without implementation) and constants (public static final variables). An interface defines a contract that a class can choose to implement. When a class implements an interface, it agrees to provide an implementation for all the abstract methods defined in the interface.

An interface can be thought of as a blueprint or a set of guidelines for how classes should behave. It allows for polymorphism, which means that objects of different classes that implement the same interface can be treated in a uniform way.

Interfaces are declared using the interface keyword, and can be implemented by a class using the implements keyword. An interface can extend another interface using the extends keyword, allowing for the creation of a hierarchy of interfaces.

Interfaces offer a way to obtain some of the advantages of multiple inheritance, such as code sharing between classes, without some of the disadvantages, such as the potential for conflicts between inherited methods, since Java does not permit multiple inheritance of classes.

Here is an Example of Interface in Java

|  |
| --- |
| public interface Animal {     public void makeSound();     public void move();  } |

We also need to implement the interface so that we can create an instance.

|  |
| --- |
| public class Dog implements Animal {      public void makeSound() {         System.out.println("Bark");      }        public void move() {         System.out.println("Running");      }   } |

Now any instance of Dog can be treated as an Animal.

|  |
| --- |
| Animal myPet = new Dog();  myPet.makeSound(); // prints "Bark"  myPet.move(); // prints "Running" |

**7.2 Abstract Class and Interfaces:**

Both interfaces and abstract classes are used in Java to achieve abstraction and create a blueprint for other classes to follow. However, there are some key differences between them:

1. Implementation: Whereas an interface can only contain abstract methods, an abstract class can contain both abstract and non-abstract methods. Also, whereas an interface cannot give any implementation for its methods, an abstract class can supply implementations for some of its methods.
2. Inheritance: A class can extend only one abstract class, but can implement multiple interfaces. This is because Java does not support multiple inheritance of classes, but allows multiple inheritance of interfaces.
3. Constructor: An abstract class can have a constructor, whereas an interface cannot have a constructor.
4. Access modifiers: In contrast to interface methods and variables, which are always public, abstract class methods and variables might have access modifiers of public, protected, or package-private.
5. Usage: Abstract classes are useful when you want to create a base class that provides some default implementations that can be inherited by its subclasses, whereas interfaces are useful when you want to define a set of methods that must be implemented by any class that wants to adhere to the interface.

In conclusion, abstract classes are useful when you want to create a base class that offers some default implementations that can be inherited by its subclasses, as opposed to interfaces, which are useful when you want to define a set of methods that must be implemented by any class that wants to adhere to the interface. Interfaces can be inherited several times, whereas abstract classes can only be extended by a single subclass, making interfaces more flexible than abstract classes.

**7.3 Purpose of an Interface:**

In Java, an interface serves as a contract or a set of guidelines for how a class should behave. It defines a set of abstract methods that must be implemented by any class that wants to adhere to the interface.

The purpose of an interface is to provide a way to achieve abstraction and polymorphism. Abstraction means hiding implementation details and providing a high-level view of functionality. Polymorphism means the ability to treat objects of different classes as if they are objects of the same class.

The definition of a class can be distinguished from the specifics of its implementation by creating an interface. You may therefore develop code that isn't dependent on any one particular implementation, which makes it simpler to maintain and alter. By establishing a set of methods that may be implemented by many classes in various ways while still being handled uniformly, interfaces also enable polymorphism.

Moreover, interfaces aid in enforcing a standard set of behaviors among many classes that implement the same interface. Without having to worry about the unique implementation details of each object, this makes it simpler to design code that can function with any object that complies with a particular interface.

Overall, the purpose of an interface in Java is to provide a contract or a set of guidelines for how a class should behave, to separate the definition of a class from its implementation details, and to achieve abstraction and polymorphism.

**7.4 When Choose Interface:**

Whenever you wish to specify a set of methods that any class that wants to abide to the interface must implement without providing any implementation details, you would use an interface in Java. When you wish to provide certain of a class's methods a default implementation while leaving some of its other methods as abstract so that their subclasses might implement them, you would use an abstract class.

Here are some specific scenarios where you might choose to use an interface instead of an abstract class:

* When you want to define a common set of behaviors across multiple unrelated classes: An interface allows you to define a common set of methods that must be implemented by any class that wants to adhere to the interface. This is useful when you have multiple unrelated classes that need to adhere to a common set of behaviors.
* When you want to achieve multiple inheritance: Since Java does not allow multiple inheritance of classes, you can achieve multiple inheritance of behaviors by implementing multiple interfaces. This is useful when you want to share behaviors across multiple classes that do not share a common ancestor.
* When you want to define a lightweight contract: Interfaces provide a lightweight way to define a contract for a class without providing any implementation details. This makes interfaces easy to read, understand, and modify.

**7.5 Restrictions on Interfaces:**

In Java, there are several restrictions on interfaces:

* **An interface cannot be instantiated**

An interface is a purely abstract concept and cannot be instantiated on its own. Instead, you must create a class that implements the interface and instantiate the class.

* **An interface cannot have constructors:**

An interface cannot have a constructor because it cannot be instantiated. Therefore, you cannot define any constructor-related logic inside an interface.

* **All methods in an interface are abstract:**

All methods in an interface are implicitly abstract and must be implemented by any class that implements the interface. They cannot have any implementation details.

* **All variables in an interface are implicitly public, static, and final:**

All variables defined inside an interface are implicitly public, static, and final. They cannot be changed or modified by any class that implements the interface.

* **An interface cannot have static or instance blocks:**

An interface cannot have static or instance blocks because it cannot be instantiated. Therefore, you cannot define any block-related logic inside an interface.

* **An interface cannot extend a class:**

An interface can only extend other interfaces, but not classes. This is because Java does not allow multiple inheritance of classes, but allows multiple inheritance of interfaces.

* **An interface cannot implement another interface:**

An interface can extend multiple interfaces, but it cannot implement another interface. This is because implementing an interface means providing concrete implementations for its methods, which is not allowed in an interface.

These restrictions are in place to ensure that interfaces remain a purely abstract concept that defines a set of methods and variables that must be implemented by any class that wants to adhere to the interface. This makes interfaces a useful tool for achieving abstraction and polymorphism in Java.

Top of Form