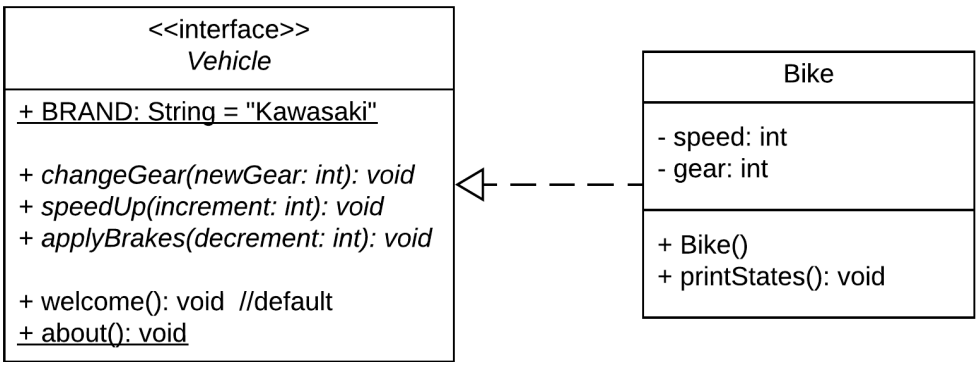
Objective:

- To understand the concept of Abstraction in Java
- To specify common behavior for objects using interfaces
- To define interfaces and define classes that implement interfaces

Interface: An *interface* is a class-like construct that contains only constants and abstract methods. It is used to provide total abstraction. Some key points about interfaces:

- All variables must be constant means the variable will be *public static final.*
- Interface doesn't contain any constructor, and an interface can't be instantiated using the *new* operator.
- All methods must be public abstract, default, or static methods.
- By default, all the fields are public, static, and final, and all the methods are public abstract.
- Multiple inheritances can be achieved through the interface.
- An interface can inherit other interfaces (more than one interface) using the *extends* keyword.

The following UML diagram shows the parent-child relationship between a class and an interface, where a Bike is a class and a Vehicle is an interface.



The implementation of the above UML diagram with the main class is shown below.

Vehicle.java

```
public interface Vehicle {
	String BRAND = "Kawasaki";

	void changeGear(int newGear);
	void speedUp(int increment);
	void applyBrakes(int decrement);
```

```
	default void welcome() {
		System.out.println("Welcome to the Kawasaki brand!");
	}

	static void about() {
		System.out.println("Kawasaki Heavy Industries Ltd is a " +
				" Japanese public multinational corporation");
	}
}
```

Bike.java

```
public class Bike implements Vehicle {
	private int speed, gear;

	public Bike() {
		speed = 0;
		gear = 0;
	}

	@Override
	public void changeGear(int newGear) {
		gear = newGear;
	}

	@Override
	public void speedUp(int increment) {
		speed = speed + increment;
	}

	@Override
	public void applyBrakes(int decrement) {
		speed = speed - decrement;
	}

	@Override
	public void welcome() {
		Vehicle.super.welcome();
		System.out.println("Let the good times roll!");
	}

	public void printStates() {
		System.out.println("speed: " + speed + ", gear: " + gear);
	}
}
```

Main.java

```
public class Main {

	public static void main(String[] args) {
		Bike b = new Bike();

		b.printStates();
		b.changeGear(2);
```

```
		b.speedUp(50);
		b.printStates();
		b.applyBrakes(20);
		b.printStates();

		b.welcome();

		System.out.println(Vehicle.BRAND);
		Vehicle.about();
	}
}
```

Multiple Inheritance: Multiple inheritance is not supported by Java using classes, but it can be achieved through the interface. Example of a multiple inheritance is given below.

Father.java

```
public interface Father {
	void honesty();
}
```

Mother.java

```
public interface Mother {
	void love();
}
```

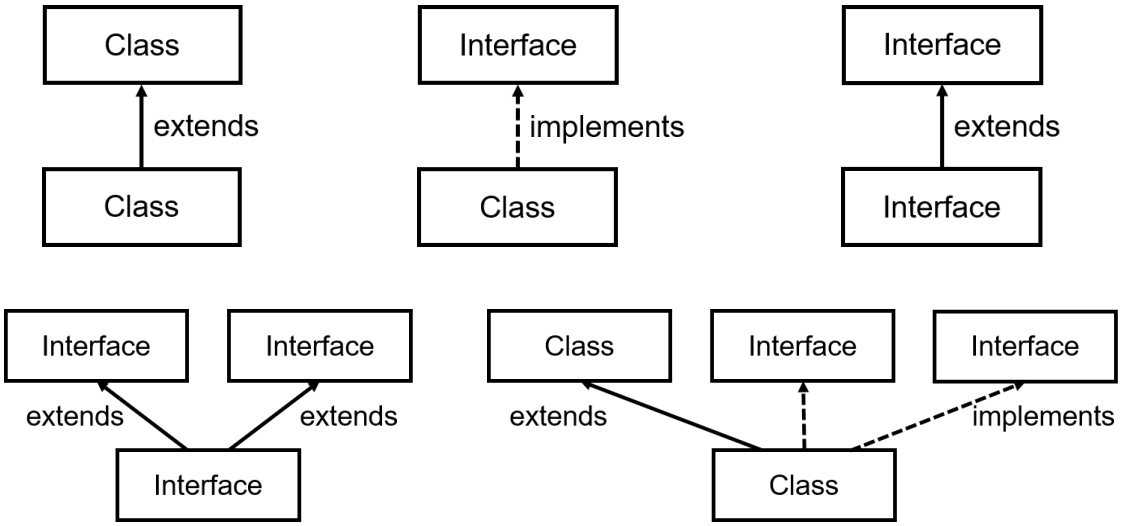
Children.java

```
public class Children implements Father, Mother {

	@Override
	public void love() {
		System.out.println("Love!");
	}

	@Override
	public void honesty() {
		System.out.println("Honesty!");
	}
}
```

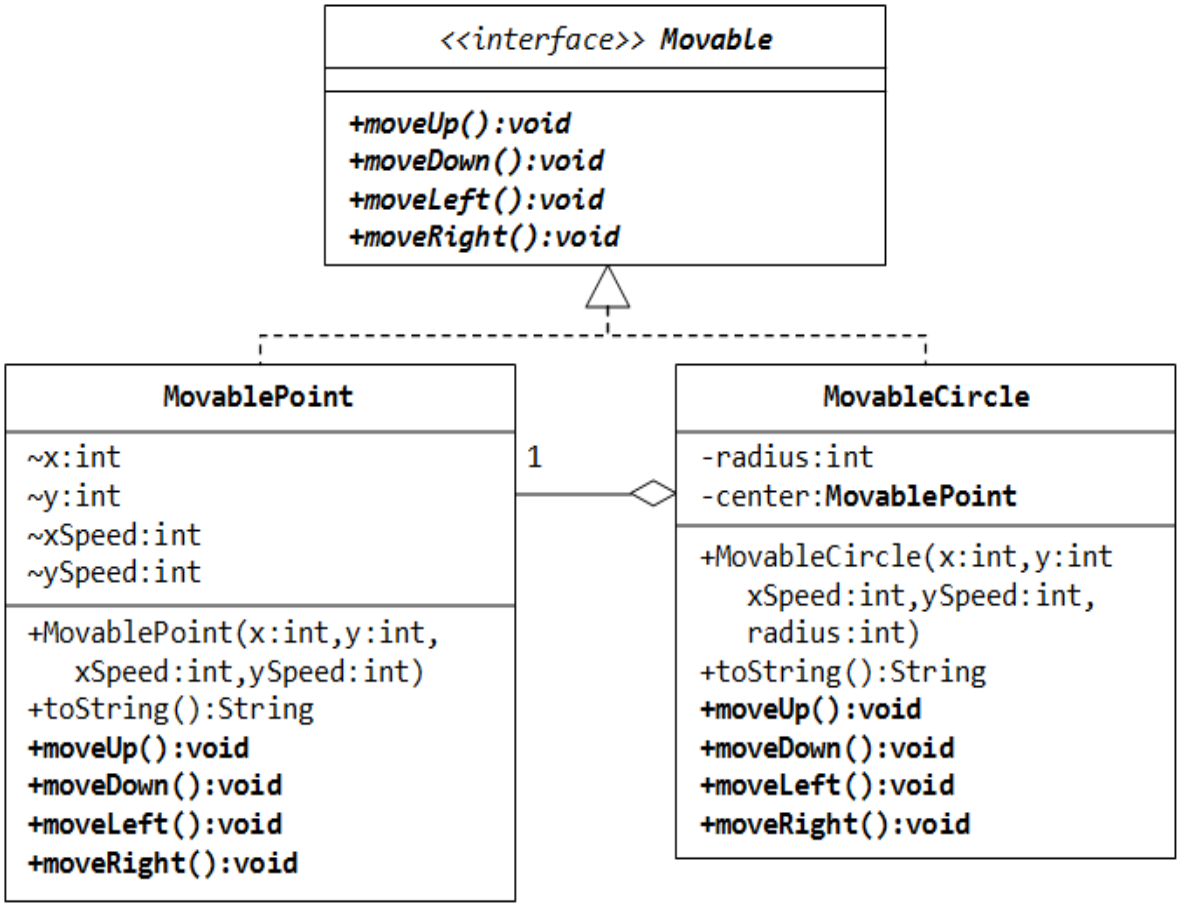
Main.java

```
public class Main {

	public static void main(String[] args) {
		Children c = new Children();
		c.love();
		c.honesty();
	}
}
```

The relationship between classes and interfaces: Several block diagrams are shown below which indicates how classes and interfaces are related to each other.



Task-01:
Implement the following UML class diagrams and test the methods.



Task-02:
Implement the following UML class diagrams and test the methods.