**Coupling** refers to the degree to which one class knows about another class. If one class uses another class, that is *coupling*.

**2. Intent/Definition**

*Coupling* refers to the degree to which one class knows about another class. If one class uses another class, that is coupling. Low dependencies between “artifacts” (classes, modules, components). There shouldn’t be too much dependency between the modules, even if there is a dependency it should be via the interfaces and should be minimal.

**Key Points**

* While creating a complex application in java, the logic of one class will call the logic of another class to provide the same service to the clients.
* If one class calling another class logic then it is called collaboration.
* When one class is collaborating with another class then there exists a tight coupling between the two classes.
* If one class wants to call the logic of a second class then the first-class need an object of second class it means the first class creates an object of the second class.

**3. Two types of Coupling**

1. Tight-coupling
2. Loose-coupling

**3.1 Tight Coupling**

If one class is tightly coupled with another class then it is tight-coupling.

**Implementation Example**

This is an example of tight coupling. Here to start the journey, the *Traveler* class is creating *Car* object to interact with it using the *move()* method.

In the above example the traveler object is tightly coupled with the car object because in place car object if you want to use bike object then, we need to make changes in Traveller class. Hence tight coupling should be avoided.

**Step 1:** Create a *Car* class.

public class Car {

@Override

public void move() {

System.out.println("Car is moving");

}

}

**Step 2:** Create *Traveler* class which holds the reference of *Car* class. *Traveler* class is tightly coupled with *Car* class.

class Traveler {

Car c = new Car();

public void startJourney() {

c.move();

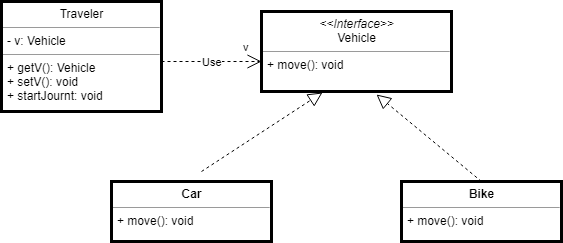
}

}

**3.2 Loose Coupling**

1. Low dependencies between “artifacts” (classes, modules, components).
2. There shouldn’t be too much dependency between the modules, even if there is a dependency it should be via the interfaces and should be minimal.
3. Avoid tight-coupling for collaboration between two classes (if one class wants to call the logic of a second class, then the first-class needs an object of second class it means the first class creates an object of the second class).
4. Strive for loosely coupled design between objects that interact.
5. Inversion Of Control (IoC) / Dependency Injection (DI) - With DI objects are given their dependencies at creation time by some third party (i.e. Java EE CDI, Spring DI…) that coordinates each object in the system. Objects aren’t expected to create or obtain their dependencies—dependencies are injected into the objects that need them. The key benefit of DI—loose coupling.

**Implementation Example**

****

This is an example of loose coupling. In this class, *Traveler* class is not tightly coupled with *Car or Bike* implementation. Instead of applying a dependency injection mechanism, the loose coupling implementation is achieved to allow the start journey with any class which has implemented the *Vehicle* interface.

**Step 1:** *Vehicle* interface to allow loose coupling implementation.

interface Vehicle {

public void move();

}

**Step 2:** The *Car* class implements the *Vehicle* interface.

class Car implements Vehicle {

@Override

public void move() {

System.out.println("Car is moving");

}

}

**Step 3:** *Bike* class implements *Vehicle* interface.

class Bike implements Vehicle {

@Override

public void move() {

System.out.println("Bike is moving");

}

}

**Step 4:**Now create a *Traveler* class that holds the reference to the *Vehicle* interface.

class Traveler {

private Vehicle v;

public Vehicle getV() {

return v;

}

public void setV(Vehicle v) {

this.v = v;

}

public void startJourney() {

v.move();

}

}

**Step 5:**Test class for loose coupling example - *Traveler* is an example of loose coupling.

public static void main(String[] args) {

Traveler traveler = new Traveler();

traveler.setV(new Car()); // Inject Car dependency

traveler.startJourney(); // start journey by Car

traveler.setV(new Bike()); // Inject Bike dependency

traveler.startJourney(); // Start journey by Bike

}