

# Liskov Substitution Principle



# LSP Overview (Self-Study)

- Requires the objects of your subclasses to behave in the same way as the objects of your superclass.
- Methods which use a superclass type must be able to work with the subclass without any issues.
- If the two classes are almost the exact same thing, why bother use this?  
Because you can add new functionality without breaking anything in base class.
- Extends Open-Closed Principle

## References:

- Tim Buchalka's Academy Course - <https://www.udemy.com/course/java-design-patterns-course/>



# LSP Real-life

- Birds such as parrot and penguins
- **The problem:** some Birds cannot fly which means they will violate LSP principle because they cannot replace their superclass
- **The Solution:** split Bird category into two, one for flying birds that can fly and one for walking birds

## References:

- Medium - <https://tusharghosh09006.medium.com/liskov-substitution-principle-lsp-744eceb29e8>



# LSP in Action - Before

```
public interface Bird{
    public void fly();
    public void walk();
}

public class Parrot implements Bird{
    public void fly(){ // to do}
    public void walk(){ // to do }
} // ok

public class Penguin implements Bird{
    public void fly(){ // to do }
    public void walk(){ // to do }
} // it's break the principle of LSP. Penguin can not fly.
```

## References:

- Medium - <https://tusharghosh09006.medium.com/liskov-substitution-principle-lsp-744eceb29e8>



# LSP in Action - After

```
public interface Bird{
    // to do;
}

public interface FlyingBird extends Bird{
    public void fly(){}
}

public interface WalkingBird extends Bird{
    public void work(){}
}

public class Parrot implements FlyingBird, WalkingBird {
    public void fly(){ // to do}
    public void walk(){ // to do }
}

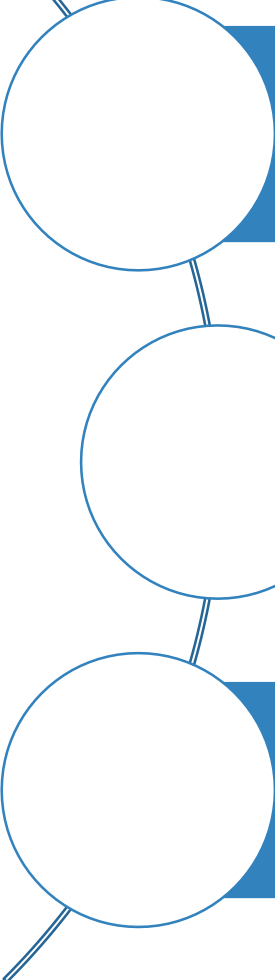
public class Penguin implements WalkingBird{
    public void walk(){ // to do }
}
```

## References:

- Medium - <https://tusharghosh09006.medium.com/liskov-substitution-principle-lsp-744eceb29e8>

# LSP summary

**HOMEWORK**



object of a superclass can be replaceable with the objects of its subclasses without breaking the application.

**Code reusability:** When LSP is followed, the code can be reused easily because it allows for the generic interfaces or base classes that can be implemented or extended by different subclasses.

**Improved code quality:** By following LSP, the code becomes more understandable and easier to maintain.