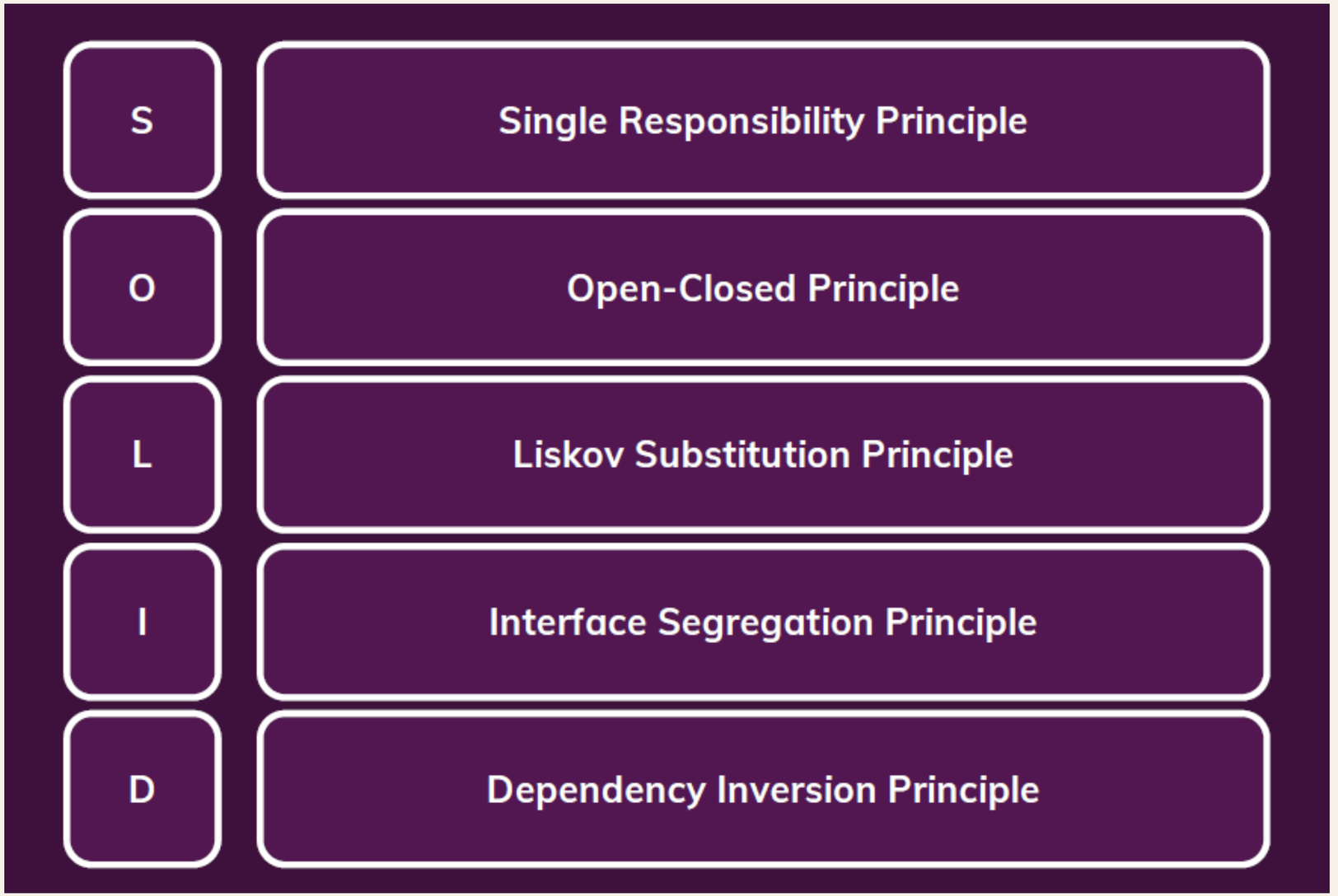
SOLID PRINCIPLES

SOLID PRINCIPLES



Single Responsibility Principle

Classes should have a single responsibility – a class shouldn't change for more than one reason.

Single Responsibility Principal

```java
package com.ilp.entity;
import com.ilp.interfaces.FeedbackInterface;

public class Feedback implements FeedbackInterface {
private String feedbackId;
private String feedback;
private int rating;

public String getFeedbackId() {
    return feedbackId;
}
public void setFeedbackId(String feedbackId) {
    this.feedbackId = feedbackId;
}
public String getFeedback() {
    return feedback;
}
public void setFeedback(String feedback) {
    this.feedback = feedback;
}
public int getRating() {
    return rating;
}
public void setRating(int rating) {
    this.rating = rating;
}
@Override
public void addfeedback(Feedback feedback) {
    System.out.println("Feedback is added");
}
@Override
public void submitfeedback(Feedback feedback) {
    // TODO Auto-generated method stub
    System.out.println("Feedback is subitted");
}
}
```

Open Closed Principle

A class should be open for extension but closed for modification.

OPEN-CLOSED PRINCIPLE

```
interface MovieFeedback
 {
void displayfeedback();
 }
class ConvertToRatings
 {
   double converttoratings()
   { }
 }
class Ratings extends ConverTtoRatings implements MovieFeedback
{

 double converttoratings()
 {  }

void displayfeedback()
      {
       System.out.println("Feedback is diplayed");
      }
}
```

Liskov Substitution Principle

Objects should be replaceable with instances of their subclasses without altering the behavior.

Liskov Substitution Principle

```java
package com.ilp.entity;
import com.ilp.interfaces.FeedbackInterface;

public class Feedback implements FeedbackInterface {
private String feedbackId;
private String feedback;
private int rating;

@Override
public void addfeedback(Feedback feedback) {
    System.out.println("Feedback is added");
}
@Override
public void submitfeedback(Feedback feedback) {
    // TODO Auto-generated method stub
    System.out.println("Feedback is subitted");
}
```

```java
ckage com.ilp.entity;

blic class FeedbackType extends Feedback {
  private boolean isPremiumType;

  public boolean isPremiumType() {
      return isPremiumType;
  }

  public void setPremiumType(boolean isPremiumType) {
      this.isPremiumType = isPremiumType;
  }
```

Feedback class

Feedback – PARENT
FeedbackType -CHILD

Liskov Substitution Principle

```java
package com.ilp.entity;

import com.ilp.interfaces.FeedbackManagerInterface;

public class FeedbackManager implements FeedbackManagerInterface  {

    private String feedbackMAnagerId;
    private String fb;
    public Feedback feedback;

public FeedbackManager(Feedback feedback) {
        this.feedback = feedback;
        System.out.println("Feedback successful\n");
```

FeedbackManager class has a constructor that will get invoked only when an object of type Feedback is passed.

In FeedbackUtility class 2 objects of class FeedbackManager is created and the object of Feedback class and FeedbackType is passed and the constructor gets invoked in both the object creations.

```java
package com.ilp.utility;
import com.ilp.entity.Feedback;
import com.ilp.entity.FeedbackManager;
import com.ilp.entity.FeedbackType;

public class FeedbackUtility {

    public static void main(String[] args) {


     Feedback fb1 = new Feedback();
     FeedbackType fbt1 = new FeedbackType();
     FeedbackManager fbm1 = new FeedbackManager(fb1);   //Liskov substitution (child)
     FeedbackManager fbm2 = new FeedbackManager(fbt1);  //Liskov substitution (parent)

     fb1.addfeedback(fb1);
     fb1.submitfeedback(fb1);

  }
}
```

Interface Segregation Principle

Many client-specific interfaces are better than one general purpose interface.

Interface Segregation

INTERFACES

```java
package com.ilp.interfaces;

import com.ilp.entity.Feedback;

public interface FeedbackInterface {
    void addfeedback(Feedback feedback);
    void submitfeedback(Feedback feedback);
}
```

```java
package com.ilp.interfaces;

import com.ilp.entity.Feedback;
public interface FeedbackManagerInterface
{
    void displayfeedback(Feedback feedback);
    void replytofeedback(Feedback feedback);
    void deletefeedback(Feedback feedback);

}
```

Classes

```java
package com.ilp.entity;

import com.ilp.interfaces.FeedbackManagerInterface;

public class FeedbackManager implements FeedbackManagerInterface  {

    private String feedbackMAnagerId;
    private String fb;
    public Feedback feedback;

    public void replytofeedback(Feedback feedback) {
        // TODO Auto-generated method stub
        System.out.println("Admin replies to the feedback");

    }
    public void deletefeedback(Feedback feedback) {
        // TODO Auto-generated method stub
        System.out.println("Admin deletes the feedback");
    }
```

```java
package com.ilp.entity;
import com.ilp.interfaces.FeedbackInterface;

public class Feedback implements FeedbackInterface {
private String feedbackId;
private String feedback;
private int rating;

@Override
public void addfeedback(Feedback feedback) {
    System.out.println("Feedback is added");
}
@Override
public void submitfeedback(Feedback feedback) {
    // TODO Auto-generated method stub
    System.out.println("Feedback is subitted");
}
```

Dependency Inversion Principle

You should depend upon abstractions, not concretions.

Dependency Inversion Principle

```java
package com.ilp.interfaces;

public interface FeedbackDatabase {

    public void store();
}
```

```java
package com.ilp.entity;

import com.ilp.interfaces.FeedbackRemoteDatabase;

public class FeedbackMainDatabase {
    private FeedbackDatabase feedbackdatabase;

    public FeedbackMainDatabase(FeedbackDatabase feedbackdatabase) {
        this.feedbackdatabase = feedbackdatabase;
    }

void saveSettings()
{
    this.feedbackdatabase.store();
}
}
```