👨‍🏫 About the Instructor
==========

Pragy
Senior Software Engineer + Instructor @ Scaler

https://linktr.ee/agarwal.pragy

💎 Key Takeaways
==========

✅ In-depth understanding of SOLID principles
✅ Walk-throughs with examples
✅ Practice quizzes & assignment

❓ FAQ
======

▶ Will the recording be available?
To Scaler students only

✏ Will these notes be available?
Yes. Published in the discord/telegram groups (link pinned in chat)

⏱ Timings for this session?
7.30pm - 10.30pm (3 hours) [15 min break midway]

🎧 Audio/Video issues
Disable Ad Blockers & VPN. Check your internet. Rejoin the session.

❔ Will Design Patterns, topic x/y/z be covered?
In upcoming masterclasses. Not in today's session.
Enroll for upcoming Masterclasses @ [scaler.com/events]
(https://www.scaler.com/events)

🖥 What programming language will be used?
The session will be language agnostic. I will write code in Java.
However, the concepts discussed will be applicable across languages

💡 Prerequisites?
Basics of Object Oriented Programming

Important Points
==========

💬 Communicate using the chat box
🙋 Post questions in the "Questions" tab
💙 Upvote others' question to increase visibility
👍 Use the thumbs-up/down buttons for continous feedback
⏱ Bonus content at the end

>
> ❓ What % of your work time is spend writing new code?
>
> • 10–15% • 15–40% • 40–80% • > 80%
>

< 15% of any devs work time is spent writing actual code!
Counterintuitive

⏱️ Where does the rest of the time go?

- planning
  - designing
  - reading other people's code
  - meetings
  - scrum
  - architecture design
  - researching – docs, articles, ..
  - requirements gathering
- debugging
- testing
- documentation
- performance reports
- code reviews

- breaks – tea, table tennis

Ensure that whatever code I write is "good" from the start.

✅ **Goals**
========

We'd like to make our code

1. readable
2. testable
3. maintainable
4. extensible

performance scalability – Data structures & Algorithms
infrastructure scalability – High level design

Robert C. Martin 👴 Uncle Bob

==========
💎 **SOLID Principles**
==========

- Single Responsibility
- Open/Close
- Liskov's Substitution
- Interface Segregation
- Dependency Inversion

Interface Segregation / Inversion of Control / Isolation

Dependency Injection / Dependency Inversion / Durability

💭 Context
==========

- Zoo Game 🦊
- characters - animals, zoo staff, visitors
- structures - cages, park, ponds, ..

Work with Pseudo code for a toy problem
Syntax highlighting - Java-esque - the language will be pseudo code

These concepts will be applicable to any modern language that supports OOP
Python/Java/JS/Typescript/C++/Ruby/C#/Php ..

🎨 Design a Character
==========

```java
class ZooEntity {
   // a character inside our zoo game - concept in our mind

   // properties (attributes)
      // animal
      Integer id;
      String name;
      Integer age;
      Integer weight;
      Boolean isNonVegetarian;
      Color color;
      String species;
      Gender gender;

      // zoo staff
      Integer employeeID;
      String name;
      Integer age;
      Gender gender;
      String department;

      // visitor
      Integer ticketID;
      String name;
      String phone;
      String address;
      Boolean isVIP;
      Gender gender;


   // methods (behavior)
      // animal
      void eat();
      void poop();
```

```
    void sleep();
    void attack();
    void speak();
    void run();
    void fly();

    // zoo staff
    void eat();
    void poop();
    void speak();
    void feedAnimals();
    void checkIn();

    // visitor
    void eat();
    void poop();
    void getEatenByALion();
    void takePhoto();
    void speak();

}
```

Major Issues
1. name collisions for attributes / methods - easy fix - just rename

🚨 Problems with the above code?

❓ Readable
Can I read it and understand it? Yes, certainly.
Consider a project with 100,000+ lines of code, 100 devs working on it, 5 year long project.

Readable right now, but as complexity grows, it will quickly become unreadable.

❓ Testable
I can totally write testcases for each method.
However, because all the code is in a single class, changing the behavior of animal can (by mistake) end up changing the behavior of the Visitor

Code is tightly coupled - is it difficult to test!

❓ Extensible
(we will come back to this later)

❓ Maintainable
dev 1 - features for animals - editing class Zooentity
dev 2 - features for visitors - editing class Zooentity
submit code changes - merge conflicts

Main issue: this code is doing way too much

🛠️ How to fix this?

==========

⭐ Single Responsibility Principle
==========

- any class/function/module (unit-of-code) should have a single, well-defined respobsility
    - single responsibility - not more than 1
    - well defined - should not be vague

- any piece of code should have only 1 reason to change

- if you identify that a piece of code has multiple responsibilities - split the code into multiple units

```java
// OOP - class Inheritance


class ZooEntity {
   // common   - age, gender, id, eating, pooping

   Integer id;
   String name;
   Integer age;
   Gender gender;

   void eat();
   void poop();
   void speak();
}

class Animal extends ZooEntity {   // every animal is also a zoo character
   // animal specific
   String species;
   Boolean isNonVegetarian;
   Boolean canFly;

   void attack();
   void run();
   void eatAVisitor();
}

class Staff extends ZooEntity {
   // staff specific
   String department;
   Ingteger salary;
   Boolean isFullTime;

   void feedAnimals();
   void checkIn();
   void cleanPremises();
}

class Visitor extends ZooEntity {
   // visitor specific
   Boolean isVIP;
   Integer ticketID;

   /// ...
}

```

- Readable
aren't there wayy too many classes/files now?

yes! earlier we had 1 class. Now we have 4 classes
but that's not an issue!
- because you will never work on multiple features at once.
- at any given time you have to read only 1 or a handful of classes/files

yes we have multiple classes now, but each class is now small and very easy to understand!

- Testable
Can a change in Animal class effect the behavior of the Staff class?
No!
Responsibilites are now decoupled!

- Maintainable
if multiple devs are working on different files, will it cause merge conflicts?
No / or at least minimized

🦤 Design a Bird

```java
class Animal extends ZooEntity {
   String species;
}

class Bird extends Animal {

   // inherits the attrib species from the parent class

   // void chirp();
   // void poopOnPeopleBelow();

   void fly();
}
```

🕊️ different birds fly differently!

```java
class Bird {
   // String species; // inherited from parent class

   void fly() {
      // how can we implement this?

      if (species == "Sparrow")
         print("fly low")
      else if (species == "eagle")
         print("glide elegantly high up above")
      else if (species == "pigeon")
         print("poop attack anyone below you and fly")

   }
```

```
}
```

🐞 Problems with the above code?

this handles multiple responsibilities - Sparrow / Eagle / Piegon

- Readable
- Testable
- Maintainable

- Extensible - FOCUS!

Imagine that I need to add a new Bird type - what code changes will I need to make?

```java

[PublicZooLibary]
{
   class Bird {
      // String species; // inherited from parent class

      void fly() {
         // how can we implement this?

         if (species == "Sparrow")
            print("fly low")
         else if (species == "eagle")
            print("glide elegantly high up above")
         else if (species == "pigeon")
            print("poop attack anyone below you and fly")

         /**

         else if (species == "DiscoChicken")
            print("females (pehens) can fly, but males cannot")

         */
      }
   }
}

[MyCustomGame] {

   import PublicZooLibary.Bird;

   // add a new type of bird - Peacock - what do I need to do?

   class ZooGame {
      void main() {
         Bird sparrow = new Bird(...);
         sparrow.fly();

         Bird eagle = new Bird(...);
         eagle.fly();

         // ... use the peacock here
      }
   }
}

```

To add a new type of Bird, I will have to modify the Bird class

- Is it always the case that you have modification access to the source code of everything that you use?
  - Do you always write everything from scratch?
    No - you use external libraries
  - A lot of libraries are shipped in compiled formats (.dll .com .exe .so .jar .pyc)

🛠️ How to fix this?

==========
⭐ **Open/Close Principle**
==========

- Code should be closed for modification, yet, it should remain open for extension!
  - modification: changing existing code
    - you might have to change it for bug fixes
    - but you should not change existing code for changing requirements
  - extension: adding new functionality in the face of changing requirements

Seems impossible! How can we add new functionality without modifying existing code?

❓ Why is it bad to modify existing code?

Code life cycle (development to deployment)

  - dev writes the code in their local machine (write/test/commit)
  - Pull Request - review
    - other people in the team they review and suggest improvements / clarifications
    - you might go and modify the code again
    - iterative
  - PR gets merged
  - Quality Assurance team - other tests (unit/integrations/manual/...)
    - iterative
  - Deployment
    + Staging servers
      - ensure that there's nothing wrong
      - monitor the health status
    + Production servers
      * A/B test
        - deploy to only 5% of the userbase
          - observability
          - monitor exceptions
          - monitor user reviews
          - revenue changes
          - system health
          - throughput
          - CPU load
      * Deploy to the entire userbase

It is really expensive to have the code go through all these steps
So if a piece of code has already gone through all of this, we don't want to repeat it.

Only new features should go through this process

```java

[PublicZooLibary]
{
   abstract class Bird {
      // String species; // inherited from parent class

      abstract void fly(); // this function hasn't been defined yet
      // we don't know how to implement it
      // anyone who claims to be a Bird should supply this function
   }

   class Sparrow extends Bird {
      void fly() { print("fly low") }
   }
   class Eagle extends Bird {
      void fly() { print("glide elegantly high up above") }
   }
   class Pigeon extends Bird {
      void fly() { print("poop attack anyone below you and fly") }
   }
}

[MyCustomGame] {

   import PublicZooLibary.Bird;
   import PublicZooLibary.Sparrow;
   import PublicZooLibary.Eagle;
   import PublicZooLibary.Pigeon;

   // add a new type of bird - Peacock - what do I need to do?
   // this time it's easy!

   class Peacock extends Bird {
      void fly() {
         print("pehens can fly, peacocks cannot")
      }
   }

   class ZooGame {
      void main() {
         Sparrow sparrow = new Sparrow(...);
         sparrow.fly();

         Eagle eagle = new Eagle(...);
         eagle.fly();

         Peacock discoChicken = new Peacock(...);
         peacock.fly();
      }
   }
}


```

- Extension
We were able to extend the Bird functionality (added a new type - Peacock) without modifying existing code

? Isn't this the same thing that we did for Single Responsibility as well?
yes - for SRP we split the class by using inhertiance
      for OCP we split the class by using inheritance

❓ Does that mean that OCP == SRP?
No - the solution was the same (OOP) but the intent was different
intent for SRP - was split the responsibilities - readable, testabile, maintainable
intent for OCP - to make the code extensible despite it not being modifiable

🔗 All the SOLID principles are tightly linked together - adhereing to 1 will automatically make your code good for some others.

Everyone focusses on DSA
But this is not enough!

How does the computer work?
- Operating systems
  - threading
  - memory management
  - disk access
- Concurrency & Synchronization
  - threads
  - threadpool
  - IO bound vs CPU bound
  - locks / semaphores
  - race conditions
  - dining philosophers
- Networks
  - protocols
  - which protocol to use when
  - how are networks connected (IP address/MAC addresses/Subnetting)

How do Databases work (SQL)
- design the DB schema
  - normalization (1NF/2NF/BCNF)
  - why normalization (read/write/delete anomalies)
- how nulls work
- advanced queries (SQL queries)
- index
- primary keys
- ACID transactions (Atomicity, Consistency, Isolation, Durability)
- Rollbacks
- optimize the query performance

Low Level Design (how to write good code)
- Object Oriented Programming (OOP)
- SOLID Principles (today's masterclass)
- Deisgn Patterns
  - when to apply what
  - what patterns are language specific
  - internet is full of bullshit
- Lots and Lots of case studies (toy problems)
  - Library management
  - TicTacToe/SnakeLadder/Chess
  - Splitwise
  - Parking lot
- Machine Coding (note: not machine learning)
  - actual working end to end code with tests & deployment

High Level Design
- Horizontal vs Vertical Scaling, Load Balancing, Consistent Hashing, Stateless vs Stateful servers, Caching architectures (local, global, distributed), CDNs, Cache Invalidation (write around, write through, ..), Eviction (LRU, FIFO, ..),

Typeahead, CAP, PACELC, ..
   1.5 month long HLD curriculum

---

quick 15 mins break:
resuming at 9.15 PM sharp

- Single Responsibility
- Open/Closed

🐓 Can all the birds fly?

```java
abstract class Bird {
   abstract void fly();
}
class Sparrow extends Bird {
   void fly() { print("fly low") }
}
class Pigeon extends Bird {
   void fly() { print("poop attack anyone below you and fly") }
}


class Kiwi extends Bird {
   void fly() {
      // kiwi cannot fly!
      // what should we do here?
   }
}
```

Kiwi, Penguin, Ostrich, Emu, Pecocks (male), Dodo, ...

>
> ❓ How do we solve this?
>
> • Throw exception with a proper message
> • Don't implement the `fly()` method
> • Return `null`
> • Redesign the system
>

🏃 Run away from your problems - simply not implement the `void fly()`

```java
abstract class Bird {
   abstract void fly();
}

class Kiwi extends Bird {
   // note: there's no void fly here!
}
```

🐞 Compiler Error: either class Kiwi must implement void fly, or it must itself be marked as an abstract class

⚠️ Throw an exception with a proper error message

```java
abstract class Bird {
   abstract void fly();
}

class Kiwi extends Bird {
   void fly() {
      throw new FlightlessBirdException("Bro, kiwi's don't fly!")
   }
}
```

🐞 This voilated expectations!

```java
abstract class Bird {
   abstract void fly();
}
class Sparrow extends Bird {
   void fly() { print("fly low") }
}
class Pigeon extends Bird {
   void fly() { print("poop attack anyone below you and fly") }
}


class ZooGame {
   Bird getBirdObjectFromUserChoice() {
      // it shows multiple bird types to the user
      // it lets the user select a species
      // creates a Bird object of that species
      // returns that object

      // this function can return a Bird object of type Sparrow, or a Bird object of type Pigeon
      // Runtime polymorphism:
         // Sparrow s = new Sparrow();
         // return s; // this is allowed, because every sparrow is also a Bird
   }

   void main() {
      Bird b = getBirdObjectFromUserChoice();
      b.fly();
   }
}
```

✅ Before extension
Code works, easy to understand, passes testcases, everyone is happy
(dev happy, user happy, QA happy)

❌ After extension

some intern gets tasked with adding Kiwis to the game

```java
class Kiwi extends Bird {
   void fly() {
      throw new FlightlessBirdException("kiwi's don't fly")
   }
}
```

Q1: did they change existing code?
   No
Q2: was the code working before this extension?
   Yes
Q3: is the code working now?
   No!
   Because now, `getBirdObjectFromUserChoice` can return a Kiwi object
   there can be an exception from the Kiwi class
   What really breaks: main method

Very very bad! This violates expectations!

==========
⭐ **Liskov's Substitution Principle**
==========

- Child classes should not violate expectations set by the parent class

- Any object of `class Parent` must be replacable (without issues) by a object of `class Child extends Parent`

- Child classes should not be forced to implement behavior that they can't exhibit (don't force all Birds to fly, because some of them can't)

🎨 Redesign the system!

```java

class Bird {
   // usual stuff
   void poop();  // all birds do poop
   void eat();   // all birds do eat

   // Note: because some birds cannot fly, it will be wrong to expect this from the Bird class
}

interface ICanFly {
   void fly();
}

// flying birds
class Sparrow extends Bird implements ICanFly {
   void fly() { print("fly low") }
}
class Pigeon extends Bird implements ICanFly {
   void fly() { print("poop attack anyone below you and fly") }
}


// flightless birds
```

```java
class Kiwi extends Bird {
   // No need to implement ICanFly Interface
}



class ZooGame {
   ICanFly getFlyingObjectFromUserChoice() {
      // it shows multiple bird types to the user
      // it lets the user select a species
      // creates a Bird object of that species
      // returns that object

      // this function can return a Bird object of type Sparrow, or a Bird object of type Pigeon
      // Runtime polymorphism:
         // Sparrow s = new Sparrow();
         // return s; // this is allowed, because every sparrow is also a ICanFly
   }

   void main() {
      ICanFly b = getFlyingObjectFromUserChoice();
      b.fly(); // this is perfectly fine!
   }
}
```


Q: but Pragy, didn't we modify existing code for this change to happen?
Q: aren't we violating the OCP?

Yes, it would be wrong to start with a bad design and then refactor it into good design - because then you would be modifying existing code

You want to design the system in a good way right from the very start!

Top salaries of Soft devs in India (Bangalore/Hyderabad) in tier-1 companies
   upto 3 Cr (base salary) 10+ years of experience

Why would a company pay this much to a dev?
   because a good dev can anticipate the future requirement changes and design code today that adheres to those changes!

✈ What else can fly?

```java
class Bird {
   // Note: because some birds cannot fly, it will be wrong to expect this from the Bird class
}

interface ICanFly {
   void fly();
```

```
   void flapWings();
}


class Shaktiman implements ICanFly {
   void fly() { print("spin fast") }

   void flapWings() {
      // SORRY Shaktiman!
   }
}
```

What does a bird to to begin a flight?
- sitting on a branch
- make a small jump with my tiny legs
- spread my wings
- flap them once
- get airborne
- start flying

Are there things apart from birds that can fly?
- aeroplanes
- bats / snakes
- insects
- kite (patang)
- shaktiman
- papa ki pari
- mom's chappal (towards me)
- udta Punjab

>
> ❓ Should these additional methods be part of the ICanFly interface?
>
> • Yes, obviously. All things methods are related to flying
> • Nope. [send your reason in the chat]
>

==========
⭐ **Interface Segregation Principle**
==========

- Keep your interfaces minimal
- clients of your interfaces should not be forced to implement behavior that they don't need

note: client =/= user. client == any code that uses your interface

❓ Isn't this similar to LSP? Isn't this just SRP applied to interfaces?
Yes & yes.
But intent is different
Liskov's Substition - type hierarchy (mathematics of the type system in code)
Interface Segregation - writing good code

LSP =/= ISP =/= SRP

🔗 All the SOLID principles are tightly linked together

How will you fix `ICanFly`?

Split it into multiple interfaces - `ICanFly` `IHasWings`, ...

---

we've designed lots of characters
now let's try to design structures

🗑️ Design a Cage
==========

```java

interface IDoor {
   void resistAttack(Attack attack);      // High Level Abstraction
}
class IronDoor implements IDoor {         // Low Level Implementation Detail
   void resistAttack(Attack attack) {
      if(attack.power <= IRON_MAX_RESISTANCE)
         return;
      print("Door Broken down - all animals are now dead/escaped")
   }
}
class WoodenDoor implements IDoor { ... } // Low Level Implementation Detail
class AdamantiumDoor implements IDoor { ... } // Low Level ...


interface IBowl {                         // High Level Abstraction
   void feed(Animal a);
}
class MeatBowl implements IBowl { ... }   // Low Level Implementation Detail
class GrainBowl implements IBowl { ... }  // Low Level Implementation Detail
class FruitBowl implements IBowl { ... }  // Low Level Implementation Detail


class Cage1 {                             // High Level Controller
   // building a cage for tigers

   // dependencies - Bowls, Animals, Doors, ...

   MeatBowl bowl = new MeatBowl();
   IronDoor door = new IronDoor();
   List<Tiger> kitties;

   void feed() {
      for(Tiger t: kitties) {
         this.bowl.feed(t);   // delegating the feeding task to a dependency
      }
   }

   void resistAttack(Attack attack) {
      this.door.resistAttack(attack); // delegating the task to a dependency
   }
}

class Cage2 {
   // building a cage for chickens
```

```
    // dependencies - Bowls, Animals, Doors, ...

    GrainBowl bowl = new GrainBowl();
    WoodenDoor door = new WoodenDoor();
    List<Chicken> chicks;

    void feed() {
        for(Chicken c: chicks) {
            this.bowl.feed(c);   // delegating the feeding task to a dependency
        }
    }

    void resistAttack(Attack attack) {
        this.door.resistAttack(attack); // delegating the task to a dependency
    }
}

// similarly 100s of other CageX classes here


// adding a new cage requires adding a new class
class CageXMen {
    // building a cage for XMen

    // dependencies - Bowls, Animals, Doors, ...

    MeatBowl bowl = new MeatBowl();
    AdamantiumDoor door = new AdamantiumDoor();
    List<XMen> xmen;

    // ...
}


class MyAwesomeZooGame {
    void main() {
        Cage1 forTigers = new Cage1();
        forTigers.feed();

        Cage2 forChickens = new Cage2();
        forChickens.resistAttack(new Attack(10));

        CageXMen ...
    }
}
```

🐞 Lot of code repetition
I literally copy pasted the Cage1 class to make the Cage2 class!

High Level vs Low Level code

High Level
- abstractions: a piece of code that tells you what to do, but not how to do it (interfaces / abstract classes)
- controllers: manegerial code that delegates tasks to dependencies

Low Level
- implementation details: tell you exactly how something is being done!

```
 interface     abstract class      interface
  -------         ---------          -------
```

```
  IBowl          Animal          IDoor          high level
 -------        ---------       -------         abstractions
    ||              ||              ||
 MeatBowl        Tiger         IronDoor         low level
                                                implementation details
     |_____________|_____________|
                   |
                 Cage1                          high level
                                                controller
```

In the above code, the High Level `Cage1` class depends on Low level implementation details `MeatBowl`, `Tiger`, `IronDoor`

==========
⭐ **Dependency Inversion Principle** **– what to do**
==========

- Code should NEVER depend on Low level implementation details
- Code should dependend ONLY on high level abstractions

```
   -------        ---------       -------
    IBowl           Animal         IDoor          high level abstractions
   -------        ---------       -------
     |_____________|_____________|
                   |
                 Cage                           code
```

But how?

==========
🗡️ **Dependency Injection** **– how to achieve it**
==========

- Don't create the dependencies yourself – let your clients supply (inject) the dependencies into you (via constructor/function params)

```java
interface IDoor { ... }                    // High Level Abstraction
class IronDoor implements IDoor { ... }    // Low Level Implementation Detail
class WoodenDoor implements IDoor { ... } // Low Level Implementation Detail
class AdamantiumDoor implements IDoor { ... } // Low Level ...

interface IBowl { ... }                    // High Level Abstraction
```

```
class MeatBowl implements IBowl { ... }   // Low Level Implementation Detail
class GrainBowl implements IBowl { ... }  // Low Level Implementation Detail
class FruitBowl implements IBowl { ... }  // Low Level Implementation Detail


class Cage {                             // High Level Controller
   // generic cage

   IBowl bowl;
   IDoor door;
   List<Animal> animals;

   // inject the dependencies via the constructor
   //                      dependency injection
   //          vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
   public Cage(IBowl bowl, IDoor door, List<Animal> animals) {
      this.bowl = bowl;
      this.door = door;
      this.animals.addAll(animals);
   }

   void feed() {
      for(Animal a: animals) {
         this.bowl.feed(a);   // delegating the feeding task to a dependency
      }
   }

   void resistAttack(Attack attack) {
      this.door.resistAttack(attack); // delegating the task to a dependency
   }
}



class MyAwesomeZooGame {
   void main() {
      Cage forTigers = new Cage(
         new MeatBowl(),
         new IronDoor(),
         Arrays.asList(new Tiger("simba"), new Tiger("mufasa")),
      );
      forTigers.feed();

      Cage forChickens = new Cage(
         new GrainBowl(),
         new WoodenDoor(),
         ...
      );
      forChickens.resistAttack(new Attack(10));



      Cage forXMen = new Cage(
         new MeatBowl(),
         new AdamantiumDoor(),
         ...
      );
   }
}
```

Enterprise Code

When you go to large companies like Google

- you will find "overengineered code"

That's okay
- because large companies have projects that have
  - 100,000+ LOC
  - 100s of devs
  - 10+ years in pipeline
  - devs join & leave all the time
  - requirements change frequently
  - huge userbases, any mistakes can effect revenue or customer loyalty

Always predict any and all future requirements, and design code from day 1 so that you don't have to modify it later!

If you're a dev who is not good at LLD
- find everything too confusing
- see names like `class PaymentGatewayStrategyAbstractFactory`
- very difficult time contributing to the codebase & reading other people's code

If you're good at LLD
- most of the time you don't even have to read the code!
- just the filename will tell you the design pattern
- and knowing the design pattern will tell you exactly what the file does!

==========

🎁 Bonus Content

==========

> We all need people who will give us feedback.
> That's how we improve. 💬 Bill Gates

🧩 Assignment

---

https://github.com/kshitijmishra23/low-level-design-concepts/tree/master/src/oops/SOLID/

⭐ Interview Questions

> ❓ Which of the following is an example of breaking
> Dependency Inversion Principle?
>
> A) A high-level module that depends on a low-level module
> through an interface

```
>
> B) A high-level module that depends on a low-level module directly
>
> C) A low-level module that depends on a high-level module
>    through an interface
>
> D) A low-level module that depends on a high-level module directly
>
```

```
> ?  What is the main goal of the Interface Segregation Principle?
>
> A) To ensure that a class only needs to implement methods that are
>    actually required by its client
>
> B) To ensure that a class can be reused without any issues
>
> C) To ensure that a class can be extended without modifying its source code
>
> D) To ensure that a class can be tested without any issues
```

```
>
> ?  Which of the following is an example of breaking
>    Liskov Substitution Principle?
>
> A) A subclass that overrides a method of its superclass and changes
>    its signature
>
> B) A subclass that adds new methods
>
> C) A subclass that can be used in place of its superclass without
>    any issues
>
> D) A subclass that can be reused without any issues
>
```

```
> ?  How can we achieve the Interface Segregation Principle in our classes?
>
> A) By creating multiple interfaces for different groups of clients
> B) By creating one large interface for all clients
> C) By creating one small interface for all clients
> D) By creating one interface for each class
```

```
> ?  Which SOLID principle states that a subclass should be able to replace
> its superclass without altering the correctness of the program?
>
> A) Single Responsibility Principle
> B) Open-Close Principle
> C) Liskov Substitution Principle
> D) Interface Segregation Principle
>
```

```
>
> ?  How can we achieve the Open-Close Principle in our classes?
>
> A) By using inheritance
> B) By using composition
> C) By using polymorphism
> D) All of the above
>
```

```
# ============================ That's all, folks! ============================
```