💎 **Key Takeaways**
================

✅ In-depth understanding of SOLID principles

✅ Walk-throughs with examples

✅ Understand concepts like Dependency Injection, Runtime Polymorphism, ..

✅ 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

👨‍💻 **About the Instructor**
=======================

Pragy
[linkedin.com/in/AgarwalPragy](https://www.linkedin.com/in/AgarwalPragy/)

Senior Software Engineer + Instructor @ Scaler

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 their work time writing fresh code!

⏱ Where does the rest of the time go?

- Reading other peoples' code
- debugging
- testing
- mentoring junior developers
- research - learning new things
- reviewing and merging PRs

- meetings
- breaks - TT, chai, ...

1. Code is written ONCE but it is read HUNDREDS of times
2. Minimize the amount of time we spend writing code
   - make sure that we do it right the very first time!
   - following a set of rules/guidelines/principles

✅ Goals

We'd like to make our code

1. Readable
2. Maintainble
3. Extensible
4. Testable

Robert C. Martin 👴 Uncle Bob

💎 SOLID Principles

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

Interface Segregation / Inversion of Control
Dependency Injection / Dependency Inversion

"Toy" problems - simple problems that clearly demonstrate a concept
At the end it is also important to practice with realistic problems
   - assignment at the end

💭 Context

- Simple Zoo Game 🦊

- Characters - Animals / Staff / Visitors
  Structures - Cages that contain the animals

SOLID Principle / Design Patterns - they apply to languages that support Object Oriented Programming (OOP)

I will use pseudo code - it might look like Java

🎨 Design a Character

```java

// in OOP, a class models a concept
// in classes we have - properties (attributes) and behavior (methods)

class ZooEntity {

    String entityType; // animal / staff / visitor

    String name;
    Color color;
    Float weight;
    String species;
    Boolean canBreatherUnderWater;

    String staffName;
    Integer age;
    String department;
    Float salary;

    String visitorType;
    String visitorName;
    // ...

    void eat() { ... }
    void fly() { ... }
    DooDoo poop() { ... }
    void run() { ... }

    void staffSleep() { ... }
    void checkInForJob() { ... }

    void walk() { ... }
    void poop() { ... }
    void clickPics() { ... }

}
```

Active vs Passive Learning
If you watch this like a movie - you will learn, and by the end of the week, you will forget

Participation is key!

🐞 Problems with the above code?

❓ Readable
I can totally read and understand this code!
But there is no separation of concerns - everything is mixed up
And as the code grows, it will quickly become very difficult to read!

❓ Testable
I can write testcases for each invidiual function!
Because Animal and Visitor functionality is in the same class, it could happen that changing one mistakenly changes other things as well
This makes testing very difficult

❓ Extensible
We will come back to this

❓ Maintainable
Tons of tons of merge conflicts because all devs are working on the same file

🛠️ How to fix this?

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

- Any class/function/module (unit-of-code) should have only 1, well-defined responsibility

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

- What should you do if you identify a piece of code which has more than 1 responsibility? Refactor it - break it down into multiple pieces of code, each with 1 well defined responsibility

```java

class ZooCharacter {
    String name;
    Integer age;
    void eat() { ... }
}

class Animal extends ZooCharacter {

    Color color;
    Float weight;
    String species;
    Boolean canBreatherUnderWater;

    void fly() { ... }
    DooDoo poop() { ... }
    void run() { ... }

}


class Staff extends ZooCharacter {

    String department;
    Float salary;

    void sleep() { ... }
    void checkInForJob() { ... }

}

class Visitor extends ZooCharacter {

    String visitorType;

    void walk() { ... }
    void poop() { ... }
    void clickPics() { ... }
}

```

- Readable
Yes, it definitely more readable now.
Isn't there too much code now? There are 4 classes now!

If you're CTO - you will work on 1 feature at a time
If you're an engineer - you will still work with 1 feature at a time

Yes, there are more files now - but you will only work with a handful of them at any given time - so overall less code for you.

- Testable
If I make a change in Animal class, will that effect the testcases of the Visitor class? Definitely no!
This is a win!

- Maintainable
If 1 dev is working on Animals, and another is working on Staff - merge conflicts?
Significantly lesser!

Focus on Animal

```java
[library] SimpleZooLibrary {

    class Animal {
        String species;

        Boolean canFly;
        Boolean canBreatherUnderWater;
        Boolean hasBeak;
        Boolean hasTail;

        Integer numberOfWings;

        void fly() {
            if(species == "Eagle") {
                print("Fly high")
            } else if (species == "Sparrow") {
                print("Fly low")
            } else if (species == "Pigeon") {
                print("Fly over people's head and poop on them")
            }
            // else if (species == "Peacock") { ... }
        }

    }

}


[our code] MyZooGame {

    import SimpleZooLibrary.Animal;

    // I want to add a new type of Animal - I wanna add peacock

    class Game {
        void main() {
            Animal sparrow = new Animal();
            sparrow.fly();
        }
    }

}
```

Do you always write all code from scratch? No.
We import lots of code from external libraries.

```java
import java.lang.String; // built in standard library
// do we have the permission to go and modify this library code? No.
```

🐞 Problems with the above code?

- Readable
- Testable
- Maintainable

- Extensible - FOCUS!

If someone else has written the library that we're importing, we still want to be able to "extend" the functionality - currently it is not possible.

🛠 How to fix this?

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

- Your code should be closed for modification - yet, open for extension!

Seems impossible!

❓ Why is modification bad?

- Write code on your local system, test it, and submit a Pull Request (PR)
- Other devs will review the PR - they will decide if something needs to be improved.
  - Iterate.
  - Finally code is merged
- Quality Assurance team (QA) - write more tests (unit/integration)
- Deployment Phase
    + Staging servers - if everything is fine
    + Production Servers
        * A/B testing
            - deploy to only 2% of the userbase
            - monitor
                - are the number of exceptions increasing
                - is the user rating decreasing
                - are the sales decreasing
                - are there more bug reports
                - is the performance still good
        * Finaly finally - deployed to the entire userbase

Only want to do this heavy process for "new" functionality - not for existing functionality

ChatGPT / Slack / Zerodha - developer APIs - expose some of the codebase as libraries to other developers
With the org - it could be that one team creates an API/library for another team

You as the API/library author have to ensure that your end users (other devs) are able to extend functionality even though they're not allowed to modify your code.

TA Help Request
  - AI solution that "assists" the TAs and students to quickly answer common queries
  - Separate API - input: query, output: resolution - Python
  - Ruby + React - rest of the codebase

```java
```

```
[library] SimpleZooLibrary {

    class Animal {
        String species;

        Boolean canFly;
        Boolean canBreatherUnderWater;
        Boolean hasTail;

    }

    abstract class Bird extends Animal {
        Integer numberOfWings;
        Integer beakSize;

        abstract void fly(); // abstract function - incomplete function
    }

    class Sparrow extends Bird {
        void fly() {
            print("fly low")
        }
    }

    class Eagle extends Bird {
        void fly() {
            print("Fly high")
        }
    }

}



[our code] MyZooGame {

    import SimpleZooLibrary.Animal;

    // I want to add a new type of Animal - I wanna add peacock

    class Peacock extends Bird {
        @override
        void fly() {
            print("pe-hens can fly, pecocks can't")
        }
    }

    class Game {
        void main() {
            // Animal sparrow = new Animal();
            // sparrow.fly();

            Peacock pea = new Peacock();
        }
    }

}

```

- Extension
I want to be able to extend code that I do not have modifications permissions for!

? Isn't this the same thing that we did for Single Responsibility Principle too?
Yes!

? So does this mean that SRP == O/C Principle?
No - the solution was same, but the intent was different

🔗 All these SOLID principles are linked together - they go hand in hand

If you implement one, a lot of times, you will get the others for free

----

Staff Engineer / Principle Engineer (v. senior positions) in Tier-1 companies (Google) - salary - in Bengaluru / Hyderabad (10+ year)
Upto 3 Cr/annum (excluding stocks)

Why would a company pay so much?

Because expert devs are expected to "anticipate" requirement changes
You're supposed to write code today that requires minimal changes in the future even if the requirements change!

Object Oriented Programming - if you're weak in something - you have to improve youself

Topics that you "need" to master (Low Level Design - LLD)

1. Fundamentals (Operating systems - how memory work, how caches work / Networking concepts)
2. Concurrency - threading, locks, semaphores, race conditions, thrashing
3. Object Oriented Programming
4. SOLID Principles
5. Design Patterns
   - tons of online resources - they don't talk about how the patterns change based on the language
   - Builder pattern
     - very heavily used in Java
     - NEVER used in Python/Ruby
6. Case Studies
   - Design Tic Tac Toe / Snake Ladder / Chess
   - Design a Parking Lot / Library Management System
   - Design Payment REST API
7. Common backend design patterns
   - MVC
   - Repository Pattern
8. How to design the database schema
   - normalize it
   - optimize the queries
   - manage your indexes
   - levels on db concurrency
   - ACID properties

9.06 - back by 9.20 sharp!

🐓 Can all Birds fly?

```java

abstract class Bird {
    String species;

    abstract void fly();
}


class Sparrow extends Bird { void fly() { print("fly low") }}
class Eagle extends Bird { void fly() { print("fly high") }}


class Kiwi extends Bird {
    void fly() {
        // ...??
    }
}
```

Penguins, Kiwis, Ostrict, Emu, Dodo, .. examples of Birds which can't fly!

>
> ❓ 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 the problem - pretend it doesn't exist!
Don't implement the kiwi.fly() function

```java

abstract class Bird {
    String species;

    abstract void fly();
}

class Kiwi extends Bird {
    // no void fly
}

```

🐞 Compiler Error! Either class Kiwi must implement void fly, or it must be an abstract class itself!

⚠

```java
abstract class Bird {
    String species;

    abstract void fly();
}

class Kiwi extends Bird {
    void fly() {
        throw new FlightlessBirdException("Kiwi's don't fly bro!")
        return null;
    }
}
```

🐞 Violates Expectations, and it causes "magical" bugs!

```java

abstract class Bird {
    String species;

    abstract void fly();
}

class Sparrow extends Bird { void fly() { print("fly low") }}
class Eagle extends Bird { void fly() { print("fly high") }}

class Game {

    Bird getBirdFromUserChoice() {
        // show an input to the user
        // the user will select a bird type
        // you will return an object of that bird type
    }

    void main() {
        Bird tweety = getBirdFromUserChoice();
        tweety.fly(); // make it fly!
    }
}
```

✅ Before extension
This code is well tested, it works correctly, devs are happy, users are happy

❌ After extension

```java
class Kiwi extends Bird {
    void fly() {
        return null;
        // or
        throw new FlightlessBirdException();
    }
}
```

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

- Any object of child `class Child extends Parent` should be able to replace an object of parent `class Parent` without any issues!

- meaning: Don't violate expectations!

- If you're expecting some functionality from a parent class - the child class must ALSO support that functionality - it shouldn't throw an exception for that functionality

- Category theory - type theory
    - mathematical principle
    - a value of a subtype should be able to replace a value of the parent type

🎨

```java

interface ICanFly {
    // Java has interfaces because it doesn't support multiple inheritance
    // Python - Abstract Base Class (ABC)
    // C++ - Virtual methods
    void fly();
}

abstract class Bird {
    String species;
    // it is not the case that all birds can fly
    // so fly should not be in the bird class
}


class Sparrow extends Bird implements ICanFly { // sparrow is a flying bird
    void fly() {
        print("fly low")
    }
}
class Eagle extends Bird implements ICanFly {
    void fly() { print("fly high") }
}


class Kiwi extends Bird {  // this does NOT implement ICanFly
    // we won't have to implement void fly
    // and the compiler won't complain!
}


class Game {

    ICanFly getBirdFromUserChoice() { // this can only return flying things
        // show an input to the user
        // the user will select a bird type
        // you will return an object of that bird type
    }

    void main() {
        ICanFly tweety = getBirdFromUserChoice();
        tweety.fly(); // make it fly!
    }
}
```

✈ What else can fly?

```java
interface ICanFly {
    void fly();

    void smallJump();

    void spreadWings();
}

abstract class Bird {
    String species;

    // .. other code
    abstract void poop();
}


class Sparrow extends Bird implements ICanFly { // sparrow is a flying bird
    void fly() {
        print("fly low")
    }
}

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

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

Helicopter / insects / Planes / Shaktiman / Doremon / Punjab / Mom's Chappal

>
> ❓ 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
==================================

- Any client of an interface should not be forced to implement a method it doen't need

- meaning: Keep your interfaces minimal

How will you fix `ICanFly`?

Split it into multiple interfaces
ICanFly / IHaveWings

Sparrow extends Bird implements ICanFly, IHaveWings
Shaktiman implements ICanFly

🔗 Isn't this just the Single Responsibility applied to interfaces?

🔗 Yes! All the SOLID principles are linked!

🗑️ Design a Cage
================

```java


/*
   High Level vs Low Level code

   High Level
      - Abstractions
         pieces of code that only tell you what to do, not how to do it
         examples:
            interfaces / apis / List / DatabaseConnection

      - Controllers
         pieces of code that "manage" other code
         delegate tasks
         "delegate": entrust (a task or responsibility) to another person, typically one who is less 
senior than oneself.
         examples:
            request handler

   Low Level
      - Concrete Implementation classes
         tells you exactly how to do something
      examples:
         ArrayList / SQLConnection / MongoDBConnection

*/


interface IDoor {                                    // high level abstraction
   void resistAttacks(Attack attack);
}
class IronDoor implements IDoor {                    // low level code
   void resistAttacks(Attack attack) {
      if(attack.power <= 100) {
         print("successfully resisted")
         attack.attacker.decreaseStrength(10);
      }
   }
}
class WoodenDoor implements IDoor { ... }           // low level

interface IBowl {                                   // high level
```

```
   void feed(Animal animal);
}
class MeatBowl implements IBowl { ... }                // low level
class FruitBowl implements IBowl { ... }               // low level

abstract class Animal {
   String species;
   String name;
}
class Tiger extends Animal { ... }
abstract class Bird extends Animal { ... }
class Sparrow extends Bird { ... }

class Cage1 {                                     // high level or low level?
   // this cage is for Tigers                        ----------

   IronDoor door;   // dependencies
   MeatBowl bowl;
   List<Tiger> tigers;

   Cage1() {
      // build my dependencies
      this.door = new IronDoor();
      this.bowl = new MeatBowl();
      this.tigers = Arrays.asList(new Tiger("t1"), new Tiger("t2"));
   }

   void resistAttacks(Attack attack) {
      // delegate the task to the door
      this.door.resistAttacks(attack);
   }

   void feedAnimals() {
      for(Tiger t: this.tigers) {
         this.bowl.feed(t);   // delegate the task
      }
   }
}

class Cage2 {
   // this cage is for Birds

   WoodenDoor door;   // dependencies
   FruitBowl bowl;
   List<Bird> birds;

   Cage2() {
      // build my dependencies
      this.door = new WoodenDoor();
      this.bowl = new FruitBowl();
      this.birds = Arrays.asList(new Sparrow("s1"), new Peacock("p1"));
   }
}

// client
class Game {
   void main() {
      Cage1 forTigers = new Cage1();
      Cage2 forBirds = new Cage2();
   }
}
```

DRY - Don't Repeat Yourself

🐞 Lot of code repetition

```
  -------        ---------        -------
   IBowl           Animal           IDoor         high level abstractions
  -------        ---------        -------
     ║               ║                ║
┌──────────┐     ┌────────┐     ┌──────────┐
```

```
 | MeatBowl |    | Tiger |    | IronDoor |        low level implementations
      |              |              |
      ----------------------------------
                     |
                 | Cage1 |                        high level controller
```

High level `class Cage1` depends on low level implementation details `MeatBowl`, `Tiger`, ...

==================================
⭐ **Dependency Inversion Principle** **– what we wanna achieve**
==================================

- High level modules should only depend on high level abstractions
- they should not depend on low level implementation details
- invert the dependency tree

```
   -------       ---------       -------
    IBowl         IAnimal         IDoor          high level abstractions
   -------       ---------       -------
      |              |              |
      -------------------------------
                     |
                  | Cage |                       high level module
```

But how?

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

- instead of creating your dependencies yourself, you let your client "inject" them into you

```java
interface IDoor { ... }                                 // high level abstraction
class IronDoor implements IDoor { ... }                 // low level code
class WoodenDoor implements IDoor { ... }               // low level

interface IBowl { ... }                                 // high level
class MeatBowl implements IBowl { ... }                 // low level
class FruitBowl implements IBowl { ... }                // low level

abstract class Animal { ... }
class Tiger extends Animal { ... }
abstract class Bird extends Animal { ... }
class Sparrow extends Bird { ... }

class Cage {
   // generic cage

   IDoor door;    // dependencies
   IBowl bowl;
   List<Animal> inhabitants;

   // inject the dependencies
   Cage(IDoor door, IBowl bowl, List<Animal> inhabitants) {
      this.door = door;
```

```java
        this.bowl = bowl;
        this.inhabitants.addAll(inhabitants);
    }
}

// client

class Game {
    void main() {
        Cage forTigers = new Cage( // inject the dependencies
            new IronDoor(),
            new MeatBowl(),
            Arrays.asList(new Tiger("t1"), new Tiger("t2"))
        );

        Cage forBirds = new Cage(
            new WoodenDoor(),
            new FruitBowl(),
            Arrays.asList(new Sparrow("s1"), new Peacock("p1"))
        );
    }
}
```

Inversion of Control - any framework (React/Spring/Django/Laravel/...)
- there's a framework which is in control
- it will inject dependencies into various controllers/classes that you have

Enterprise Code

- find complex design patterns everywhere
    - AbstractBuilderFactory
    - Singleton
    - .. other complex and long names

- Google has over 50k engineers
- lots of codebases
- all of the engineers are working in teams
- people leave the company very often

If you're a junior dev who doesn't know Low Level Design
- you look at code - there's 50k files
- every filename is 100 letters long
- weird and complex names

If you're good at LLD
- you won't even have to read half of the code
- just looking at the file name tells you exactly what the code does!

- Data Structures & Algo   - 1-2 years (entry level)
- Low Level Design
- Concurrency
- High Level Design
- Projects

🎁 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! ============================