💎 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?
   5pm - 8pm (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%

⏱️ Where does the rest of the time go?

- debugging
- reading code
  - lokup docs
  - explore stackoverflow
  - asking chatGPT for help
- test
- refactor - maintaining

- participate in meetings
- play TT
- plan the sprints

I want to write "perfect" code in the first go

✅ Goals

We'd like to make our code

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

Robert C. Martin 👴🏻 Uncle Bob - book "Clean Code"

💎 SOLID Principles

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

Interface Segregation / Inversion of Control
Dependency Inversion vs Injection

💭 Context

- Building a simple Zoo Game 🦊
- Various entities - animals, cages, food, staff, visitors, health, paths

---

🎨 Design the Entity

```java

class ZooEntity {
    // attributes - properties

    // animal
    String species; Integer age; FoodType diet; Color color; Gender gender; // ...
    // cage
    Material material; Integer squareFoot; Boolean isOpen;
    // staff
    String staffId; String role; Integer age; // ...

    Boolean isStaffEntity;

    // methods - behavior

    void takeCareOfAnimal();
    void getPaid();
    void cleanPremises();

    void eat();
    void sleep();
    void attackVisitors();
}

```

🐠 Let's implement the behavior of the various entities

```java

class ZooEntity {
    // attributes - properties
    void getPaid() {
        // for the staff
        if(!isStaffEntity) throw new InvalidObjectError();

        // calculate my salary based on position, hours worked, experience, ..

    }
}

```

🐞 Problems with the above code?

❓ Readable

Yes, I can definitely read it and understand it - fresher

1. This file will be HUGE
2. I have to myself figure out which attribute/behavior belongs to which entity

❓ Testable
It seems that I can write a test case for each method
Because all the different entities are all in the same class, making change for Animal could (by mistake) effect the behavior of another entity - Staff

❓ Extensible - we will come to this later

❓ Maintainable
There will be a lot of code conflicts - merge conflicts

🛠️ How to fix this?

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

- Every function/class/module (unit-of-code) should have one, and only one, *well-defined* responsibility

- Any piece of code has only 1 reason to change

- If a unit of code has multiple responsibilities - break it down into multiple units

```java

class Entity {
   String id; // primary key
   Date createdAt, updatedAt; // auditing
}


class Animal extends Entity {
   String species;
   Integer age;
   FoodType diet;
   Color color;
   Gender gender; // ...

   void eat();
   void sleep();
   void attackVisitors();
}

class Staff extends Entity {
   // attributes & behavior of the staff
}


class Visitor extends Entity {
}


class VisitorPass extends Entity {
}

```

- Readable
1. each class is now small
2. every class is now much more readable & easy to understand
3. Isn't there wayy too many classes now?
   - yes, kinda
   - but that's not an issue
   - at any given time, you will be working on 1 (or few) classes
      - each of them are perfectly readable

- Testable

If we modify some attribute/method inside Animal class, will it have a side-effect on the VisitorPass class? No.
Code is now "de-coupled" - side effects are less. Testcases are robust!

- Maintainable

Now different devs are working on different files - signficantly less conflicts

---

🐦 **Design Animal**

```java

class Animal {
   String species;
   Integer age;
   String color;

   void run();
   void fly();
   void eat();
}

```

🕊 Implement flying - different birds fly differently

```java

class Animal {
   String species;
   Integer age;
   String color;

   void fly() {
      if(!SpeciesDetector.isBird(species)) return;

      if (species == "Sparrow")
         print("fly low")
      else if (species == "Pigeon")
         print("Shit on people while flying")
      else if (species == "Eagle")
         print("Spread wings, and glide elegantly")

      // add a new else-if condition to add a new bird
   }
}


```

🐞 Problems with the above code?

- Readable
- Testable
- Maintainable

- Extensible - FOCUS!

Suppose that you don't have write access to this code.
How do we add a new animal type?

🛠 How to fix this?

```
========================
⭐ Open/Close Principle
========================
```

```
- Code should be closed for modification, yet, it should be open for extension!
                 ^^^^^^^^^^^^^^^^^^^^^^^
                 not allowed to edit
```

❓ Why should we not modify code?

Coding workflow

```
- dev: write code, test on local machine, commit, and raise Pull Request
- team: review the request, give suggestions, iterative process .. PR gets merged!
- QA team: write new testcases for it. Integration testing
- Deployment
    + staging servers
        + check if there are any issues - was the deployment successful - preview the changes
    + production
        * A/B testing
            - deploy to 5% of the userbase
            - monitor the health
            - number of exceptions / errors are not increasing
            - monitor the user reviews - make sure that they're not decreasing
        * final deployment to 100% of the userbase
```

Oracle - almost an entire month for a feature to go from commit to prod

```java

[library] ZooLibrary {
   // jar, .com, .o, .pyc, exe, dll, ...

   abstract class Animal {
      String species;
      Integer age;
      String color;
   }

   abstract class Bird extends Animal {
      abstract void fly();
   }

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

   class Eagle extends Bird {
      void fly() {
            print("Spread wings, and glide elegantly")
        }
    }
    // ...
}

[our code]
// install the library first - install some specific version
import ZooLibrary.Bird;
import ZooLibrary.Sparrow;

// want to add a new Bird type, can we do it?

class BionicBird extends Bird {
    Float oilCapacity;
    Float oilAmount;
}

class Peacock extends Bird {
    // implement your custom behavior
}

class PetSparrow extends Sparrow {
    // override whatever we want here
}

class Main {
    void main() {
        Sparrow tweety = new Sparrow();
        Peacock proudy = new Peacock();
    }
}
```

- Extension

Did we improve on the extension?

All we have to do is create a new class - and even without modifying existing code, we can still extend it!

Design Patterns - Strategy Pattern

❓ Isn't this the exact same solution (breaking a class into multiple) as the Single Responsibility?
Yes, definitely!

❓ Is SRP == O/C principle?
No. Solution was same, but the intent was different

🔗 All the SOLID principles are tightly linked!

🐓 Can all birds fly?

```java

// existing code
```

```java
abstract class Animal {
    String species;
    Integer age;
    String color;
}

abstract class Bird extends Animal {
    abstract void fly();
}

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

// we want to add a new bird type - Kiwi

class Kiwi extends Bird {
    void fly() {
        // what do we put here??
    }
}

```

There are certain birds that cannot fly - Kiwi, Penguin, Ostrich, Emu, 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 the problem - don't implement the `void fly()`

```java

class Kiwi extends Bird {
    // no void fly()

    void eat() { ... }
}

```

🐞 Compiler Error!
Either you should implement `void fly()` inside `class Kiwi extends Bird`, or you should mark the `Kiwi` class abstract as well!

⚠ Throw a proper exception

```java

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

```

🐞 Violates Expectations!

```java

abstract class Bird {
   abstract void fly();
}

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

class ZooApp {
   Bird getBirdFromUserChoice() {
      // get the list of user choosable birds from some configuration
      // use reflection to dynamically create an object of the correct class
      if(user.choice == "sparrow")
         return new Sparrow();
      else if(user.choice == "eagle")
         return new Eagle();
      // prompt the user to select a species
      // create the appropriate object
      // return that object
   }

   void main() {
      Bird b = getBirdFromUserChoice(); // Sparrow object, Eagle object
      b.fly();
   }
}

```

✅ Before extension

The code is working, it is well tested, and everyone is happy!

❌ After extension

Without modifying existing code, it now magically breaks!

```java

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

   void main() {
      Bird b = getBirdFromUserChoice(); // Sparrow object, Eagle object, Kiwi object
      b.fly(); // exception!!
   }

```

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

- Any child class `class Child extends Parent` should not change the interface that it inherits from the parent `class Parent`

- A type should be replacable by its subtype — wherever you're using an object of `class

`Parent`, if you replace that object with an object of `class Child extends Parent`, the code should continue working

- child class shouldn't break expectations

🎨 Redesign the system - use interfaces or class hierarchy

```java

interface ICanFly {
    void fly();
}

abstract class Bird {
    abstract void eat();
    abstract void poop();

    // no void fly here - because we know that not all birds can fly
}

class Sparrow extends Bird implements ICanFly {
    void fly() { print("fly low"); }
}
class Eagle extends Bird implements ICanFly {
    void fly() { print("fly high"); }
}
class Kiwi extends Bird {
    // because I don't implement ICanFly, I don't have to provide the void fly()
}




class ZooApp {
    ICanFly getBirdFromUserChoice() {
        // get the list of user choosable birds from some configuration
        // use reflection to dynamically create an object of the correct class
        if(user.choice == "sparrow")
            return new Sparrow();
        else if(user.choice == "eagle")
            return new Eagle();
        // prompt the user to select a species
        // create the appropriate object
        // return that object

        // there is no way this can return a Kiwi, because Kiwi doesn't implement ICanFly
    }

    void main() {
        ICanFly b = getBirdFromUserChoice(); // Sparrow object, Eagle object
        b.fly();
    }
}


```

I'm using Python/C++/C#/Javascript - what do I do?

- How do learn Object Oriented Programming in my language of choice
- How do I learn about all these "advanced" things like Reflection, metaprogramming, ...

- Runtime Polymorphism / Design Patterns - Strategy Pattern
- you're going too fast

- what are some resources?

It takes time to learn, and "digest" these topics

Topic List

Scaler - Low Level Design (1.5 months)

1. Object Oriented Programming
   1. Classes vs Interfaces
   2. Inheritance
   3. Composition over Inheritance
   4. Polymorphism
   5. In python/C++ you can have multiple inheritance
      But in Java you can't - use interfaces
   ...
2. SOLID Patterns
3. Design Patterns
   1. Singleton
   2. Builder - always use this in Java, but NEVER in python!
   3. Strategy
   4. Factory
   ...
4. How to build a system
   1. Requirements
   2. Class / Entity-Relationship diagram
   3. Design database schema (well-normalized)
   4. Choose which design patterns to use and why & apply them
5. Lots of practice - Case studies
   1. Build Snake Ladder / Chess / Tic Tac Toe
   2. Parking Lot / Library Management
   3. Splitwise
   ...
6. REST APIs / MVC Pattern

6.40 -> 6.55 (quick 15 mins break)

✈ What else can fly?

```java

interface ICanFly {
    void fly();
    void flapWings();
    void kickToTakeOff();
}

abstract class Bird {
    abstract void eat();
    abstract void poop();

    // no void fly here - because we know that not all birds can fly
}

class Sparrow extends Bird implements ICanFly {
    void fly() { print("fly low"); }
}
class Eagle extends Bird implements ICanFly {
    void fly() { print("fly high"); }
}
class Kiwi extends Bird {
    // because I don't implement ICanFly, I don't have to provide the void fly()
}

// What else can fly?
// Bats / Insects / Jetpack / Shaktiman / Aeroplanes / Mom's Chappal / Patang (kite)

class Shaktiman implements ICanFly {
    void fly() { print("Sping around really fast"); }
    void flapWings() {
        // Sorry Shaktiman!
    }
}
```

>
> ❓ 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]
>

They shouldn't be in the ICanFly interface, if our codebase have entities apart from birds that can fly

⭐ Interface Segregation Principle

- Clients of an interface should not be forced to implement methods they don't need

- keep our interfaces minimal

How will you fix `ICanFly`?

Break the interface down into multiple interfaces

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

interface IHasWings {
    void flapWings();
    void kickToTakeOff();
}



class Sparrow extends Bird implements ICanFly, IHasWings {
    void fly() { print("fly low"); }
}
class Eagle extends Bird implements ICanFly, IHasWings {
    void fly() { print("fly high"); }
}
class Kiwi extends Bird {
    // because I don't implement ICanFly, I don't have to provide the void fly()
}

// What else can fly?
// Bats / Insects / Jetpack / Shaktiman / Aeroplanes / Mom's Chappal / Patang (kite)

class Shaktiman implements ICanFly {
    void fly() { print("Sping around really fast"); }
    // it doesn't implement IHasWings
}
```

🔗 Isn't this just the Single Responsibility Principle applied to interfaces!?
🔗 All SOLID principles are tightly linked

Won't this make the codebase too lengthy?
- depends on what your usecase is
- SOLID principles are guidelines, not rules
    - it is important to know when to follow the guidelines, and when to avoid them
    - you're in a hackathon / startup - you might violate some SOLID principles
    - if you're working @ Google, on a large codebase, in a team of 20 devs - follow ALL SOLID principle very carefully

Max dev salary @ Google (for senior positions), in India (Bengaluru / Hyderabad) - 3 Cr
Why do these companies pay so much?
- senior devs have to anticipate what might happen tomorrow
- how can I write code today, so that tomorrow, I don't have to re-write it

---

🗑️ Design a Cage

```java


/*
High level code and low level code

High level code is something that is "abstract" - it tell you what to do, but not how to do it
   interfaces / abstract classes

Low level code tells you exactly how to perform something
   classes / functions

Delegation - assign the task to someone else
Manager takes a list of tasks, and assigns them to individual workers
Workers actually perform the task

Manager - high level role (more abstract role)
Worker - low level role (more concrete role)

*/

interface IBowl {                                  // high level abstraction
   void refill();
   void feed(Animal animal);
}

class GrainBowl implements IBowl { ... }           //
class MeatBowl implements IBowl {                  //  low level
   void refill() {
      // get a meat bag from the refrigerated storage
      // unpack it
      // put it in a plate
      // add enzymes
      // add supplements
   }

   void feed(Animal animal) {
      // ensure that this is the correct diet for this animal
      // let the animal eat it
   }
}
class MilkBowl implements IBowl { ... }            //

interface IDoor {                                  // high level abstraction
   void resistAttack();
   void lock();
   void unlock();
}
class WoodenDoor implements IDoor { ... }          //
class IronDoor implements IDoor { ... }            // low level
class AdamantiumDoor implements IDoor { ... }      //

class Cage1 {                                      // is this high level
   // this cage is for small birds                 // or ~low~level~?
   WoodenDoor door = new WoodenDoor();
   GrainBowl bowl = new GrainBowl();

   List<Bird> birds;                               // manager/controller class

   public Cage1() {
      birds.add(new Sparrow(...))
      birds.add(new Pigeon(...))                   // high level class
   }

   void feed() {
      for(Bird b: birds)
         this.bowl.feed(b)
   }

   void lock() {
      this.door.lock()    // delegate the task
   }
}

class Cage2 {
   // this cage is for big cats - lions, tigers, etc

   IronDoor door = new IronDoor();
   MeatBowl bowl = new MeatBowl();

   List<BigCat> kitties;

   public Cage2() {
      birds.add(new Sparrow(...))
      birds.add(new Pigeon(...))                   // high level class
   }

   void feed() {
      for(BigCat b: kitties)
         this.bowl.feed(b)
   }

   void lock() {
      this.door.lock()    // delegate the task
   }
}


class ZooAPp {
   void main() {
      Cage1 forBirds = new Cage1();
      Cage2 forCats = new Cage2();
   }
}
```

🐞 Lot of code repetition

- Cage1 and Cage2 class don't follow DRY (Don't repeat yourself)
- if I want to store XMen in the Zoo, then I've to implemet a completely new class

```
  -------        ---------        -------
   IBowl          IAnimal          IDoor            high level abstractions
  -------        ---------        -------
```

```
    ||              ||              ||
 ┌──────────┐   ┌───────┐   ┌──────────┐
 │ MeatBowl │   │ Tiger │   │ IronDoor │          low level
 └──────────┘   └───────┘   └──────────┘
     |              |              |
     ------------------------------
                    |
                ┌───────┐
                │ Cage2 │                          high level
                └───────┘
```

High level class `Cage2` depends on low level implementation details `MeatBowl`, `Tiger`, etc.

==================================
⭐ **Dependency Inversion Principle**   **– what do we want**
==================================

- High level code should *not* depend on low-level implementation details
- High level code should only depend on high-level abstractions

```
   -------      ---------      -------
    IBowl        IAnimal        IDoor           high level
   -------      ---------      -------
     |              |              |
     ------------------------------
                    |
                ┌──────┐
                │ Cage │                         high level
                └──────┘
```

We want high level class `Cage` to directly depend on the high level abstractions `IBowl`, etc.

But how?

=======================
🖋 **Dependency Injection**   **– how do we achieve it?**
=======================

- Instead of creating the dependencies yourself, you let your clients create the dependencies for you, and you let them "inject" those dependencies into you

```java
interface IBowl { ... }                            // high level abstraction
class GrainBowl implements IBowl { ... }          //
class MeatBowl implements IBowl { ... }           // low level
class MilkBowl implements IBowl { ... }           //

interface IDoor { ... }                           // high level abstraction
class WoodenDoor implements IDoor { ... }         //
class IronDoor implements IDoor { ... }           // low level
class AdamantiumDoor implements IDoor { ... }     //


class Cage {
   IBowl bowl;                           // dependency-inversion
   IDoor door;                        // dependency-inversion
   List<Animal> inhabitants;          // dependency-inversion


   //               dependency-injection via the constructor
   //               vvvvv        vvvv              vvvvvvvvvvv
   public Cage(IBowl bowl, IDoor door, List<Animal> inhabitants) {
      this.bowl = bowl;
      this.door = door;
      this.inhabitants.addAll(inhabitants);
   }
}

class ZooApp {
   void main() {
      Cage birdCard = new Cage(
         new GrainBowl(),
         new WoodenDoor(),
         Arrays.asList(new Peacock(), new Sparrow())
      );

      Cage kittyCage = new Cage(
         new MeatBowl(),
         new IronDoor(),
         Arrays.asList(new Tiger(), new Lion())
      );

      Cage xmenCage = new Cage(
         new MeatBowl(),
         new AdamantiumDoor(),
         Arrays.asList(new Wolverine(), new Deadpool())
      );
   }
}
```

Enterprise Code
===============

When you go to large companies like Google
- crack the interview
- survive & thrive in the company

You will find very complex - you will look at code and you will think that it is overengineered.

If you don't know SOLID principles, and Design Patterns, and OOP - it will be impossible for you to make sense of it.

However, if you know Low Level Design (LLD) - then you won't even have to read the code! Just the filename/classname will tell you exactly what the code does!

================

🎁 Bonus Content
================

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

Resources

- Clean Code book: https://github.com/dev-marko/clean-code-book

- Design Patterns:
  - Python specific - if you're a pythonista, then please only follow this: https://python-patterns.guide/
  - for general langauges, follow this: https://refactoring.guru/design-patterns

🧩 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! ===========================