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

Pragy
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 their work time will be spent writing actual new code!

SOLID principles - is this really useful in our day to day job?

⏱ Where does the rest of the time go?

- Reading & understanding code
    - documentation
    - reviewing PRs
    - discussing with others
    - visiting stackoverflow
    - using chatGPT
- debugging, maintaining, ...
- Meetings
    - scrum
- Breaks - play TT, play snooker, chai, ..

✅ **Goals**
========

Minimize my work time and maximize my play time (while ensuring high salary)

Whatever work I do - it is very high quality - strive toward perfection!

"You can't improve what you can't measure"

We'd like to make our code

1. Readability
2. Maintainability
3. Extensibility
4. Testability

Robert C. Martin 👴 Uncle Bob

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

- Single Responsibility

- Open Closed
- Liskov's Substitution
- Interface Segregation
- Dependency Inversion

Single Use / Single Responsibility / Scalability / Simple
Open Close / Optimization

💭 Context

During a 3 hour class, can we write all the code for a large company like Amazon?
Any real problem will have a lot of nuances - and coding it will take time.

It's always better to start from curated examples - to show the gist of something

- Zoo Game 🦊
- Characters - staff, animals, visitors
- Structures - feeding place, cages, doors

Programming language:
- we will be writing pseudo-code
- I will use the syntax highlighting of java
- whatever concepts that we study today - will apply to any modern programming language that support Object Oriented Programming
    - Python
    - C++
    - Javascript (Typescript)
    - Ruby
    - Php
    - C#
    - Java / Kotlin
    - Dart

Golang / Rust / C / Delphi / Pascal - these languages don't support OOP
So the SOLID priciples might not apply to these languages (at least in the traditional manner)

Master topics like Object Oriented Programming
- resources
- career related questions
- resume building

🎨 Design a Character

Multiple characters in the zo
- Zoo Staff
- Animals

   - Visitors

All of these are "concepts" / "ideas" in our mind
Translate ideas => code - which feature of OOP do we use?
   We use a `class`
(Google: method vs function)

```java
class ZooCharacter {
   // attributes - properties (data members)
      // Staff
         String staffName;
         Gender staffGender;
         Integer age;
         Double salary;
         String designation;
         String department;
         // ...

      // Animal
         String name;
         Gender gender;
         Integer age;
         Boolean eatsMeat;
         Boolean canFly;

      // Visitor
         String name;
         Gender gender;
         Ticket ticket;
         Boolean isVIP;
         DateTime timeOfArrival;

   // methods - behavior
      // Staff
         void sleep();
         void cleanPremises();
         void eat();
         void poop();
         void feedAnimals();

      // Animal
         void sleep();
         void eat();
         void poop();
         void fly();
         void fight();
         void eatTheVisitor();

      // Visitor
         void roamAround();
         void eat();
         void petAnimals();
         void getEatenByAnimals();

}

class ZooCharacterTester {
   void testAnimalEat() {
      ZooCharacter animal = new ZooCharacter(...);
      animal.eat();

      // assert that this outputs a certain value
   }
}
```

Major Issues
Name collisions for staff/visitor/animal
- that is easy to fix - not a major issue
- we can just rename
    - bad solution
    - but okay, for now this will do

🐞 Problems with the above code?

❓ Readable
Yes, I can read and understand this code.
However, as the complexity increases
- multiple types of staff
    - gatekeeper
    - ticket issuer
    - cleaning staff
    - feeding staff
    - doctors
- categories of visitors
    - free ticket
    - paid ticket
    - premium ticket
    - underage - they must be accompanied by an adult
    - group of visitor - school visit

So as the complexity grows, this code becomes extremely difficult to read and understand.
   BEcause I have to look at so many different behaviors and things

❓ Testable
Yes, I can write testcases for this!
However if someone modifies the bhevaior of staff/visitor, that could (by mistake) effect the behaviour of animal
   because all the attributes are shared - within the same class we have all these attributes and methods

❓ Extensible
Will focus on this later

❓ Maintainable
If we have multiple devs working on different things
- Siddharth - Animal
- Vishnu - Staff
- Naved - Visitors

When they push their code - merge conflict!
because all these devs are working on the same class and the same file

🤔 What is the main reason for all these issues?

This class is doing too much work - it has wayy too many responsibilities

🛠 How to fix this?

=====================================
⭐ Single Responsibility Principle   (SRP)
=====================================

- Any class/function/module (unit of code) should have only 1, well-defined responsibility
    - one responsibility: not too many
    - well-defined: it should not be vague
        - anyone reading the code should be able to understand what the responsibility is

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

If you encounter a piece of code that doesn't adhere to the SRP
- you should break it down into multiple pieces

What feature of OOP can we use to break a large class into multiple classes?
   Inheritance!

```java

class ZooCharacter {
   String name;
   Integer age;
   Gender gender;

   void eat();
   void poop();
   void sleep();
}

class Staff extends ZooCharacter {
   String designation;
   Double salary;

   void cleanPremises();
}

class Visitor extends ZooCharacter {
   String ticketID;

   void roamAround();
}

class Animal extends ZooCharacter {
   Boolean canFly;
```

```java
    String species;

    void eatTheVisitor();
}
```

OOP - used when the developer time is more expensive than the CPU time
    - 99% of the cases, the dev is more valuable
    - 1% cases (high freq trading / building games / rendering video / machine learning) the CPU time is more valuable - do NOT use SOLID principles here

Did we improve on any of the metrics? Did we solve any of the issues?

❓ Readable

   ! But we have too many classes/files now !

   - No matter what position you're at (junior/senior) you will be working with only 1 or a few features at any given time
   - at any given time you're interacting with only a few files

   So yes, there's a lot of files now, but each of those is small, and easy to read and understand

❓ Testable

   If I make a change in the Animal class, can these changes effect the behavior of the Staff class?

   No! This means that the testcases are more robust now! Less coupling of the code!

❓ Extensible
(Will come back to this later)

❓ Maintainable

   We will have reduced the merge conflicts because diff devs are working on different files

So is the code perfect now?
   No. Far from it

Did we take a step in the right direction?
   Yes. We improved on all metrics by making a small change!

Requirements => develop code => ship it => everyone is happy :)

NEVER.

Requirements => develop code => Requirements change => .... => it's a never ending process!

We want to expand on the animals - add the behvior for a Bird

🐦 Design a Bird

```java

// don't clutter the Animal class - follow SRP
// break the animal class into subclasses - Bird / Reptile / Mammal / ..

class Bird extends Animal {
    // inherits the attributes from parent classes
    // String species;

    void fly() {

    }
}

```

🕊 different birds fly differently!

```java
class Bird extends Animal {
    // inherits the attributes from parent classes
    // String species;

    void fly() {

        if (species == "sparrow")
            print("flap wings and fly low")
        else if (species == "pigeon")
            print("fly on top of people and poop on their heads")
        else if (species == "eagle")
            print("glide elegantly high above")

    }
}
```

🐞 Problems with the above code?

- Readable
- Testable
- Maintainable

- Extensible - FOCUS!

❓ Do we always write all code ourselves from scratch?

  - you copy paste from stackoverflow / chatGPT

- you use other people's code by importing external libraries

```java

[PublicZooLibary] // this is a library that we found on github
{
    class Bird extends Animal {
        // inherits the attributes from parent classes
        // String species;

        void fly() {

            if (species == "sparrow")
                print("flap wings and fly low")
            else if (species == "pigeon")
                print("fly on top of people and poop on their heads")
            else if (species == "eagle")
                print("glide elegantly high above")

        }
    }
}

[MyCustomGame] {

    import PublicZooLibary.Animal;
    import PublicZooLibary.Bird;
    // .. import other things from external Libraries

    // If I wish to add a new type of Bird
    // can I modify the Bird class?

    class MySimpleZooGame {

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

    }

}

```

? Do we always have modification access?

No we don't. Most of the time we don't even have the source code
- libraries can be shipped in compiled form
  (.so .dll .com .jar .war .pyc ...)
- even if you have the source code, you might not have modification access to it

? How can we add a new type of Bird?

If we don't have modification access to the original codebase, how can we extend to add a new type of Bird?

🛠️ How to fix this?

⭐ Open Close Principle

- Any piece of code should be
  - open for extension, but
  - closed for modification (don't modify existing code)

❓ Why should code be "closed" for modification

❓ Why is it bad to modify existing code?

The typical code lifecycle in a large company (Google)

- Developer
  - write code on their laptop
  - test it locally
  - commit & push & generate a Pull Request (PR)
- PR will go for review
  - other devs in the team will suggest improvements
  - the dev will go and make changes to make those improvements
  - they will re-raise the PR
  - re-review
  - (iterative process)
  - PR gets merged
- Quality Assurance
  - write extra unit tests
  - write integration tests
  - UI: manual testing
  - update the docs
- Deployment Pipeline
  + Staging servers
    - will ensure that the code doesn't break
    - there will be a lot of testing (unit/integration) & stress-testing
  + Production servers
    * A/B test
      - deploy to only 5% of the users
      - we will monitor a lot of metrics
        - number of exception
        - customer satisfaction
        - number of purchases
        - ...
    * Deploy to 100% of the userbase

In large companies it can take upto a month for a piece of code to go through this pipeline

We DO NOT want the same piece of code to go through all this again
Only new code should go though this

```java

[PublicZooLibary] // this is a library that we found on github
```

```
{
    abstract class Bird extends Animal {
        // inherits the attributes from parent classes
        // String species;

        abstract void fly();
    }

    class Sparrow extends Bird {
        void fly()  {
            print("flap wings and fly low")
        }
    }
    class Pigeon extends Bird {
        void fly()  {
            print("fly on top of people and poop on their heads")
        }
    }
    class Eagle extends Bird {
        void fly()  {
                print("glide elegantly high above")
        }
    }

}

[MyCustomGame] {

    import PublicZooLibary.Animal;
    import PublicZooLibary.Bird;
    import PublicZooLibary.Sparrow;
    // .. import other things from external Libraries

    // If I wish to add a new type of Bird
    // can I modify the Bird class?

    class Peacock extends Bird {
        void fly() {
            print("female (pe-hens) can fly, but the males can't")
        }
    }

    class MySimpleZooGame {

        void main() {
            Bird sparrow = new Sparrow(...);
            sparrow.fly();
        }

    }

}
```

- Was I able to add a new Bird type?
  Yes
- Did I have to change existing code?
  - Yes? Didn't we modify the original Bird class
  - Not really

      - OCP says that the original code (Bird class) should've been written in an extensible manner
      - and it were written in a manner that followed the OCP
      - then any external dev (who doesn't have modification access) would still have been able to "extend" the code without modifying it

❓ Isn't this the same thing that we did for Single Responsibility as well?

ZooCharacter had too many responsibilities
   - to fix that we broke it down into multiple subclasses
   - to follow the Single Responsbility Principle (SRP)

Bird class was not extensible
   - to fix that we broke it down into multiple subclasses
   - to follow the Open Close Principle (OCP)

Absolutely - we made the same "change"

❓ Does that mean that OCP == SRP?

No. The solution was the same, but the intent/benefit was different

🔗 All the SOLID principles are tightly linked together. If you adhere to one, you might get others for free!

Salary for Staff Engineer / Principle Architect @ Google in Bengaluru/Hyderabad

SDE => SDE 2 => SDE 3 => Staff Enginner (10+ YOE)

3 Crores (1+ cr is the base package + bonus + stocks)

Why would a company pay this much to 1 developer?

Because "good" developers are able to anticipate future changes and write code today that doesn't need to be modified for those future changes!

"System Design" - Low Level Design / High Level Design

SOLID Principle is a part of Low Level Design

Scaler LLD Cucciculum - 2 months

Topics to learn to master Low Level Design

- Object Oriented Programming (inheritance, abstraction, encapsulation, generalization, polymorphism, composition over inheritance, MRO, interfaces, ...)
- SOLID Principle
- Design Patterns

    - Structural / Behavioral / Creational
    - builder / singleton / factory / strategy / adapter / proxy / ...
- Design the database schema
- Entities & Relationships (ER Diagram / Class Diagram)
- Case Studies
    - Tic Tac Toe / Chess / Snake Ladder
    - Parking Lot / Splitwise / Library Management
- Testing & Test Driven Developement
- REST API
    - idempotency
    - naming conventions

(if you don't understand a topic, will you be able to realize when someone is teaching you incorrectly? No)
Can you implement Builder Pattern in Python
- You should NEVER implement the builder pattern in python
- why?
    - because the builder pattern provides the following
        - named arguments
        - positional arguments
        - optional arguments
        - validation of arguments
- builder pattern makes sense in Java
- C++ / Python / Ruby / Kotlin / JS - all these features are provided out of the box

Always have a mentor who know what the correct thing is and can guide you on the correct thing

Where to learn this?
1. Free Masterclasses with certifications: https://www.scaler.com/events
2. Free courses with certifications: https://www.scaler.com/topics/
3. Comprehensive Program with Masters degree: https://www.scaler.com/academy/

9.23 -> 12 mins break -> 9.35

class resumes at 9.35 sharp :)

🐓 Can all Birds fly?

```java

abstract class Bird {
    abstract void fly();
}

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

class Pigeon extends Bird {
```

~~~
    void fly()  {
        print("fly on top of people and poop on their heads")
    }
}

class Eagle extends Bird {
    void fly()  {
        print("glide elegantly high above")
    }
}

class Kiwi extends Bird {
    void fly() {
        // Kiwi's can't fly
    }
}

```


No. There are many species of birds that cannot fly
Kiwi, Ostrich, Penguin, 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 - Simply don't implement the `Kiwi.fly()` method

```java

abstract class Bird {
    abstract void fly();
}

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

    void poop() {
        print(...)
    }
}
```
    🐞 Compiler Error!
       Because the term "abstract" enforces a "contract"
       "abstract" = incomplete (missing details)

   I cannot instantiate (create an object of) the Bird class directly -
because it is incomplete

   First, some class must "extend" bird and supply the missing functionality
`abstract void fly()`
   Only then, I can instantiate from that child class
~~~

    Either the class Kiwi should implement the abstract method `void fly()`
    or, the class Kiwi itself should be marked as abstract (incomplete)

⚠ Raise a proper exception

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

class Kiwi extends Bird {

    void fly() {
        throw new FlightlessBirdException("Bro, I'm a kiwi, I can't fly")
    }
}
```

    🐞 Violates Expectations!

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

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

class Pigeon extends Bird {
    void fly()  {
        print("fly on top of people and poop on their heads")
    }
}


class ZooGame {

    Bird getBirdObjectFromUserSelection() {
        // go through all the bird classes
        // show the various birds to the user in a nice UI
        // the user will select one bird type
        // we will create an object of that bird type
        // return that object

        // this function can return Sparrow(..) or Pigeon(..)
    }

    void main() {
        Bird b = new getBirdObjectFromUserSelection()
        b.fly()
    }
}
```

Runtime polymorphism: if I have a variable of datatype of `class Parent` then I can store an object of `class Child extends Parent` in that variable

✅ Before extension

- Does the code work?
  yes
  dev is happy, QA is happy, CEO is happy, customer is happy

An intern comes to the company - and implements the class Kiwi

```java
class Kiwi extends Bird {
    void fly() {
        throw new FlightlessBirdException("Bro, I'm a kiwi, I can't fly")
    }
}
```

❌ After extension

- Did you modify your existing code?
  No
- Are you aware of the fact that someone has made this change?
  No (most likely)
- Was the code working before this addition was made?
  Yes
- Is the code working now?
  No! Why?
  because now, the `Bird getBirdObjectFromUserSelection()` can potentially return a `Kiwi` object, and then the `void main()` method will fail!
- Is the intern's code failing?
  No
- Whose code is failing?
  My code!
  W.T.F!!??

Violates Expectations!

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

- mathematical definition: Any object of `class Parent` should be replacable with any object of `class Child extends Parent` without causing any issues

- Indian intuition:
    - parents - set expectations
    - child
        - fullfill the expectations
        - child not NOT violate the expectations set by the parent

- A child class should NOT violate the expectations set by the parent class

(solution: If you're the parent, don't have unrealistic expectations!)

🎨 Redesign the system!

```java

abstract class Bird {
   // we will NOT have the fly method here because not all birds can fly
   // abstract void fly() // this is bad,

   abstract void eat() // all birds can eat
   abstract void poop()
}

interface ICanFly {
   // allows me to "standardize" the API
   void fly();
}


class Sparrow extends Bird implements ICanFly {
   void eat() { ... }
   void poop() { ... }


   void fly() // because it implements ICanFly
}

class Kiwi extends Bird {
   void eat() { ... }
   void poop() { ... }

   // there's no need to implement void fly()
   // because we're not implementing the ICanFly interface
   // no compiler error here
}


```

```py

from abc import ABC, abstractmethod

class Bird(ABC):
   @abstractmethod
   def eat(self):
      ...
   @abstractmethod
   def poop(self):
      ...

class FlightMixin(ABC):
   @abstractmethod
   def fly(self):
      ...
```

```python
class Sparrow(Bird, FlightMixin):
    def eat(self):
        print(...)
    def poop(self):
        print(...)
    def fly(self):
        print(...)


class Kiwi(Bird):
    def eat(self):
        print(...)
    def poop(self):
        print(...)
```

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

- Earlier the `abstract class Bird` had `abstract void fly()`, but now we moved it to the interface
- This is not modification
- we expect the developer to follow the LSP from the very start

We're not modifying code, we're asking the dev to write better code from the get go.

How will the code & main method look now?

```java

abstract class Bird {
    // we will NOT have the fly method here because not all birds can fly
    // abstract void fly() // this is bad,

    abstract void eat() // all birds can eat
    abstract void poop()
}

interface ICanFly {
    // allows me to "standardize" the API
    void fly();
}


class Sparrow extends Bird implements ICanFly {
    void eat() { ... }
    void poop() { ... }


    void fly() // because it implements ICanFly
}

class Kiwi extends Bird {
```

~~~
    void eat() { ... }
    void poop() { ... }

    // there's no need to implement void fly()
    // because we're not implementing the ICanFly interface
    // no compiler error here
}


class ZooGame {

    Bird getBirdObjectFromUserSelection() {
        // go through all the bird classes
        // show the various birds to the user in a nice UI
        // the user will select one bird type
        // we will create an object of that bird type
        // return that object

        // this function can return Sparrow(..) or Pigeon(..) or Kiwi(..)
    }

    void main() {
        Bird b = new getBirdObjectFromUserSelection()
        // b.fly() // compiler error

        if(b instanceof ICanFly) {
            // explicitly cast it
            ICanFly flyingB = (ICanFly) b;
            flyingB.fly();
            // or the compiler might be able to detect the enclosing type 
constraint and not require explicit casting
        }

    }
}
```
~~~

✈ What else can fly?
=====================

~~~
```java

abstract class Bird {
    ...
}

interface ICanFly {
    void fly();

    void flapWings();
    void kickToTakeOff();
}
~~~

```
class Sparrow extends Bird implements ICanFly {
    void fly()
}

class Kiwi extends Bird {
    // there's no need to implement void fly()
}

class Shaktiman implements ICanFly {
    void fly() {
        print("raises finger, and spin, and make weird sounds")
    }

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

Shaktiman / Airplane / Papa ki Pari / Bat / Insects / Kite (patang) / mummy ki chappal / Drone

How does a Bird fly?
    - sit on a branch
    - make a small kick from my tiny legs to jump in air
    - spread wings
    - flap my wings
    - take flight

>
> ❓ 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]
>

Not really!

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

- Keep your interfaces minimal
- Your client should not be forced to implement methods that it doesn't need!

```java

interface DatabaseCursor {
    List<Row> find(String query)
    List<Row> insert(String query)
    List<Row> delete(String query)
    List<Row> join(String query, String table1, String table2)
}
```

```java
class SQLDatabaseCursor {
    List<Row> find(String query)
    List<Row> insert(String query)
    List<Row> delete(String query)

    List<Row> join(String query, String table1, String table2)
}

class MongoDatabaseCursor {
    List<Row> find(String query)
    List<Row> insert(String query)
    List<Row> delete(String query)

    // but MongoDB doesn't support joins!
}
```

❓ Isn't this similar to LSP? Isn't this just SRP applied to interfaces?

- LSP is about type theory (designing compilers and what is mathematically allowed)
- SRP is about code responsibilities and readability
- ISP is about thinking about your clients and how they might use your interfaces

  intents differ, even though the end result/solution might not

🔗 All the SOLID principles are tightly linked

How will you fix `ICanFly`?

---

multiple inheritance
multi-level inheritance
abstract class vs interface (java)
abstract class vs mixin (python)

why don't some programming languages support multiple inheritance
- Diamond problem

How does python deal with the diamond problem
- Method Resolution Order (MRO)

🗑️ Design a Cage

```java

interface IBowl {                     // High level abstraction (what to do)
    void feed(Animal animal);
}
```

```
class MeatBowl implements IBowl {    // Low Level implementation detail
   void feed(Animal animal) {
      // measure the meat out
      // remove any tiny bones so animal doesn't choke
      // add enzymes for easy digestion
      // do temperature check
      // feed the animal
   }
}
class FruitBowl implements IBowl {   // Low Level implementation detail
   void feed(Animal animal) {
      ...
   }
}

interface IDoor {                    // High level abstraction (what to do)
   void resistAttack(Attack attack);
}
class IronDoor implements IDoor {
   void resistAttack(Attack attack) {// Low Level implementation detail
      if (attack.power <= 100)
         print("attack failed")
      else
         ...
   }
}
class WoodenDoor implements IDoor {  // Low Level implementation detail
   void resistAttack(Attack attack) {
      ...
   }
}
class AdamantiumDoor implements IDoor { ... } // Low Level implementation
detail



class Cage1 {              // High level (controller class)
   // cage for Tigers
   // dependencies - we will delegate our responbilities to these
dependencies

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

   void feedAnimals() {
      for(Tiger kitty: this.kitties)
         // delegate the feeding task to the dependency
         this.bowl.feed(kitty)
   }

   void resistAttack(Attack attack) {
      // delegate to our dependency
      this.door.resistAttack(attack);
   }
}

class Cage2 {
   // cage for Pigeons

   FruitBowl bowl = new FruitBowl();
```

```
    WoodenDoor door = new WoodenDoor();
    List<Pigeons> tweeties;

    void feedAnimals() {
        for(Pigeon tweety: this.tweeties)
            // delegate the feeding task to the dependency
            this.bowl.feed(tweety)
    }

    void resistAttack(Attack attack) {
        // delegate to our dependency
        this.door.resistAttack(attack);
    }
}


class ZooGame {
    void main() {
        // create a cage for tigers
        Cage1 tigerCage = new Cage1();

        // create a cage for pigeons
        Cage2 pigeonCage = new Cage2();


        // create a cage for XMen
        // I have to create a class
    }
}
```

🐞 Too much code repetition!
🐞 To add a new cage, I need to first implement a new class for that cage

High Level vs Low Level code

High Level (abstractions)
  - code that tells you what to do, but not how to do it

Low Level (implementation details)
  - code that tells you exactly how to do some task
  - step by step approach

```
 interface    abstract class    interface
  -------       ---------        -------
   IBowl          Animal          IDoor          High Level
  -------       ---------        -------         abstraction
     ‖              ‖               ‖
┌──────────┐    ┌───────┐    ┌──────────┐
│ MeatBowl │    │ Tiger │    │ IronDoor │        Low Level
└──────────┘    └───────┘    └──────────┘        implementation details
```

```
    │                               │
    └───────────────┬───────────────┘
                    │
               ╔═════════╗
               ║  Cage1  ║                         some piece of code
               ╚═════════╝
```

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

=================================
⭐ **Dependency Inversion Principle**   **– the guideline**
=================================

- Code should NOT depend on low level implementation details
- Code should ONLY depend on high level abstractions

```
   -------         ---------        -------
    IBowl            Animal           IDoor            High Level
   -------         ---------        -------            abstraction
      │                │                │
      └────────────────┼────────────────┘    (direct dependence)
                       │
                  ╔═════════╗
                  ║  Cage   ║                        some piece of code
                  ╚═════════╝
```

But how?

=======================
💉 **Dependency Injection**   **– how to achieve that guideline**
=======================

- Don't create/instantiate your dependencies yourself
- Let your client "inject" the dependencies into you

```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 Implementation Detail

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 {
   // this will be generic - it can model any type of code

   IDoor door;
   IBowl bowl;

   List<Animal> animals;

   // client can inject dependencies (via constructor / via methods)
   public Cage(IDoor door, IBowl bowl, List<Animal> animals) {
      this.door = door
      this.bowl = bowl
      this.animals.addAll(animals)
   }
   ...
}


class ZooGame {
   void main() {
      // create a cage for tigers
      Cage tigerCage = new Cage(
         new IronDoor(...),
         new MeatBowl(...),
         Arrays.asList(new Tiger("Simba"), new Tiger("Musafa"))
      );

      // create a cage for pigeons
      Cage pigeonCage = new Cage(
         new WoodenDoor(...),
         new FruitBowl(...),
         Arrays.asList(new Pigeon("Tweety"), new Pigeon("Shweety"))
      );


      // create a cage for XMen
      Cage xmenCage = new Cage(
         new AdamantiumDoor(...),
         new MeatBowl(...),
         Arrays.asList(new XMen("Wolverine"), new XMen("Deadpool"))
      );
   }
}
```

We achieved more by writing less code!

Enterprise Code

When you go to companies like Google
- you will "over engineered" code
    - design patterns everywhere
    - very long names everywhere
- even when it is not strictly needed

If you're a dev who is not good at LLD

    - but you will have a very hard time
    - you will not be able to understand the code
    - be put into a PIP (Performance improvement proposal)

If you're dev who is good at LLD
    - 90% of the time you won't even have to read the code
    - because the class name will tell you exactly what the class 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! ===========================
```