eBuilder TechTalk #4

Object Oriented Design Principles

Speaker: Wimal Perera
Date: 22/5 (Tuesday) from 9.30AM - 11.00AM
Venue: Moonstone, 5th floor

Targeted audience: Developers (However this event is OPEN for anyone who has an interest in the topic. Expect to see lots of code.)

Agenda

UML

OO Design Principles

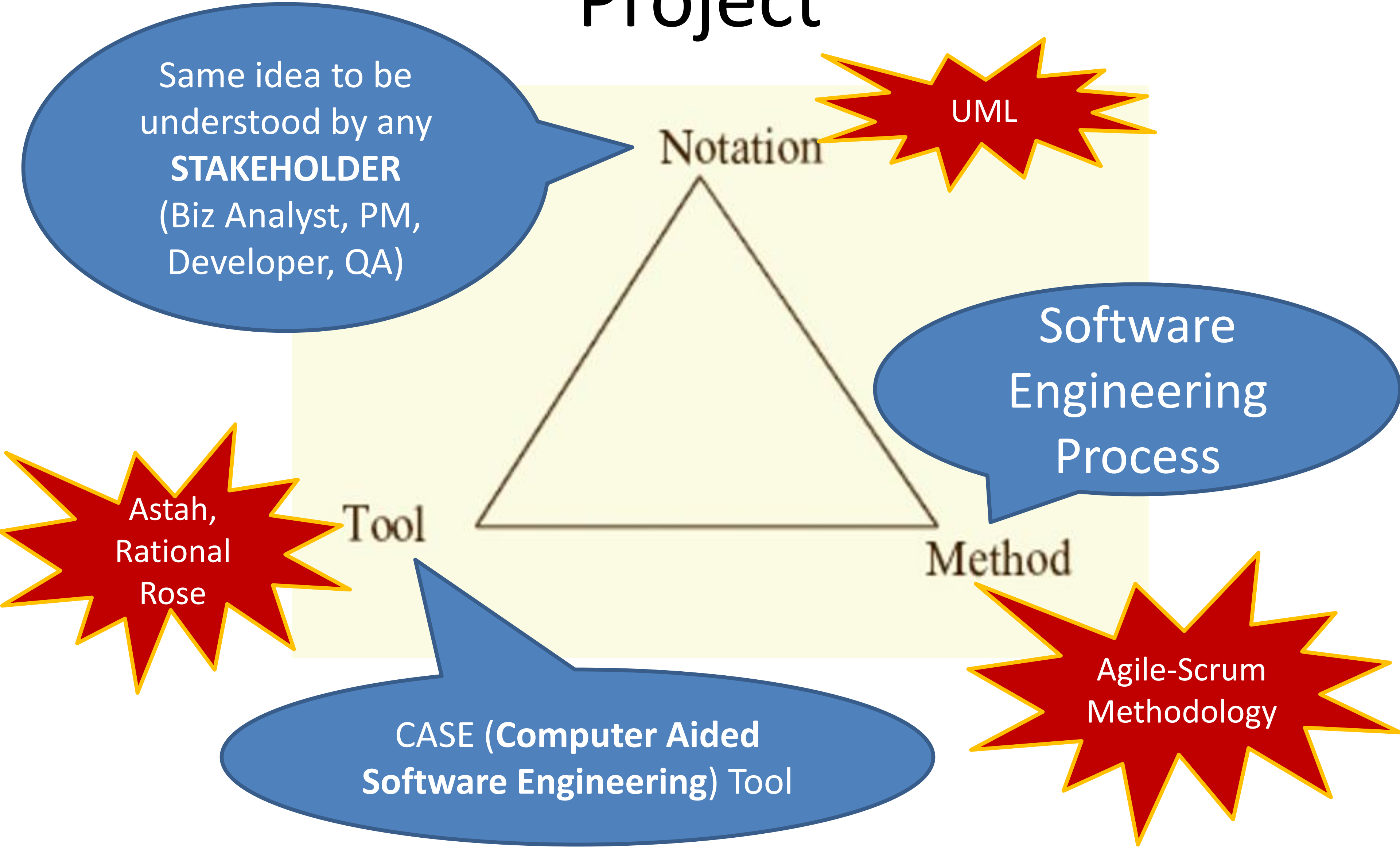
Agile/Scrum

Refactoring

PUTTING IT ALL TOGETHER

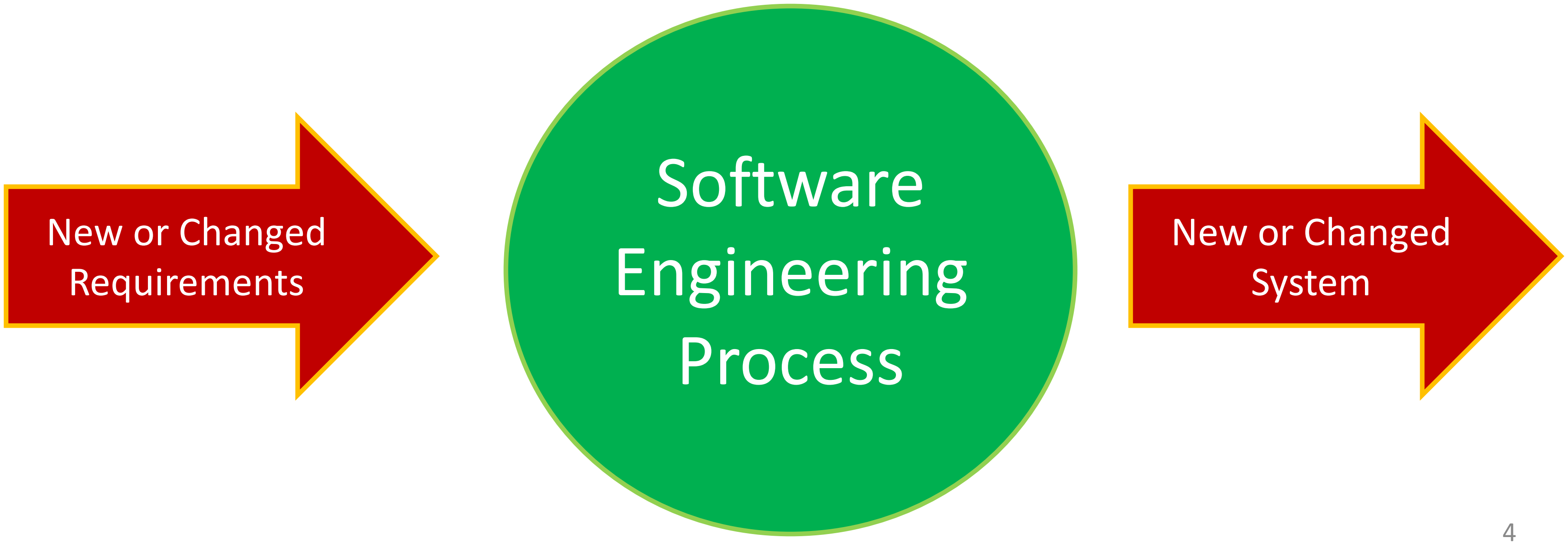
Juggling Code

Towards a Successful Software Project


Same idea to be understood by any **STAKEHOLDER** (Biz Analyst, PM, Developer, QA)
Notation
UML
Software Engineering Process
Tool
Method
Astah, Rational Rose
Agile-Scrum Methodology
CASE (**Computer Aided Software Engineering**) Tool

Software Engineering Process

- Defines **“Who”** is doing **“What”**,
- **“When”** to do it
- **“How”** to reach a certain goal



Software Development Perspective

Project Management Perspective

Agile:

Promotes **iterative** and **incremental** software development

Scrum:

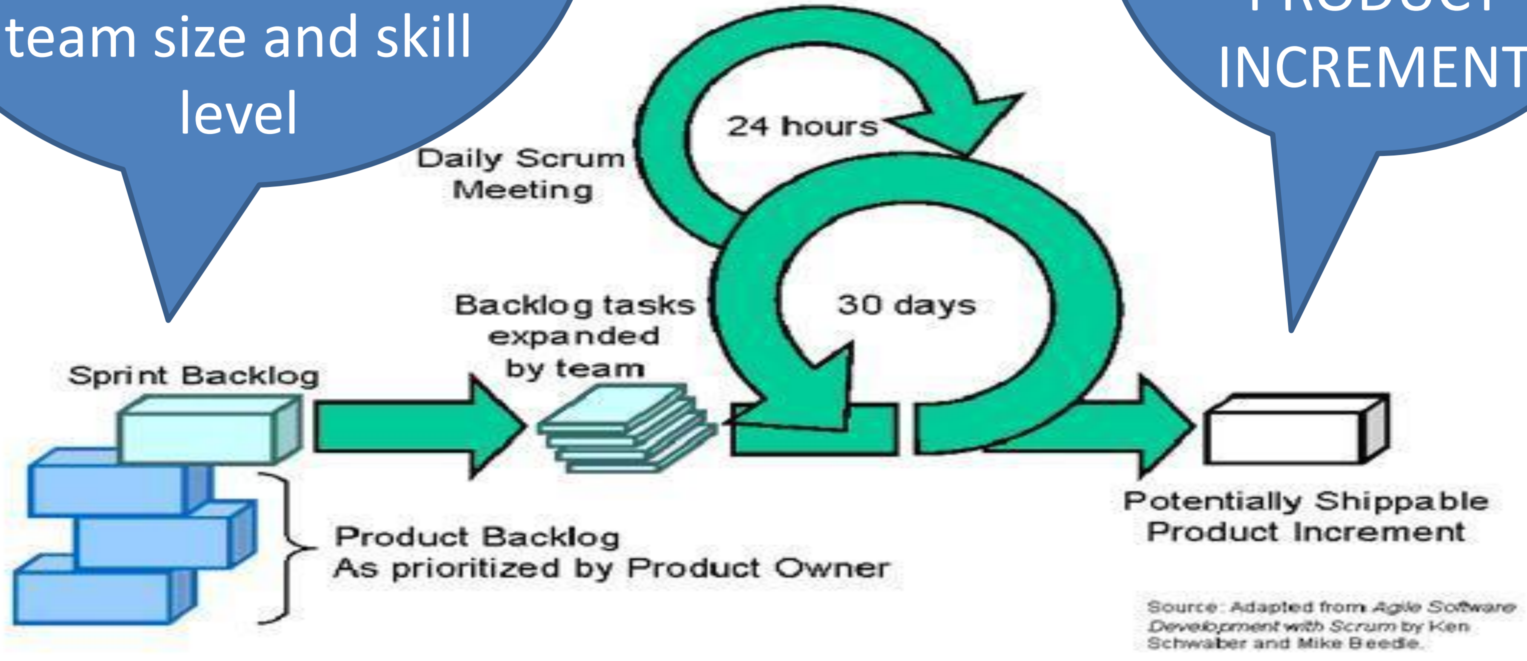
Most widely used project mgmt approach for Agile Development.

- Open Communications
- Time Boxing
- Priorities Set by Product Owner
- Demonstratable Results

Agile-Scrum Methodology

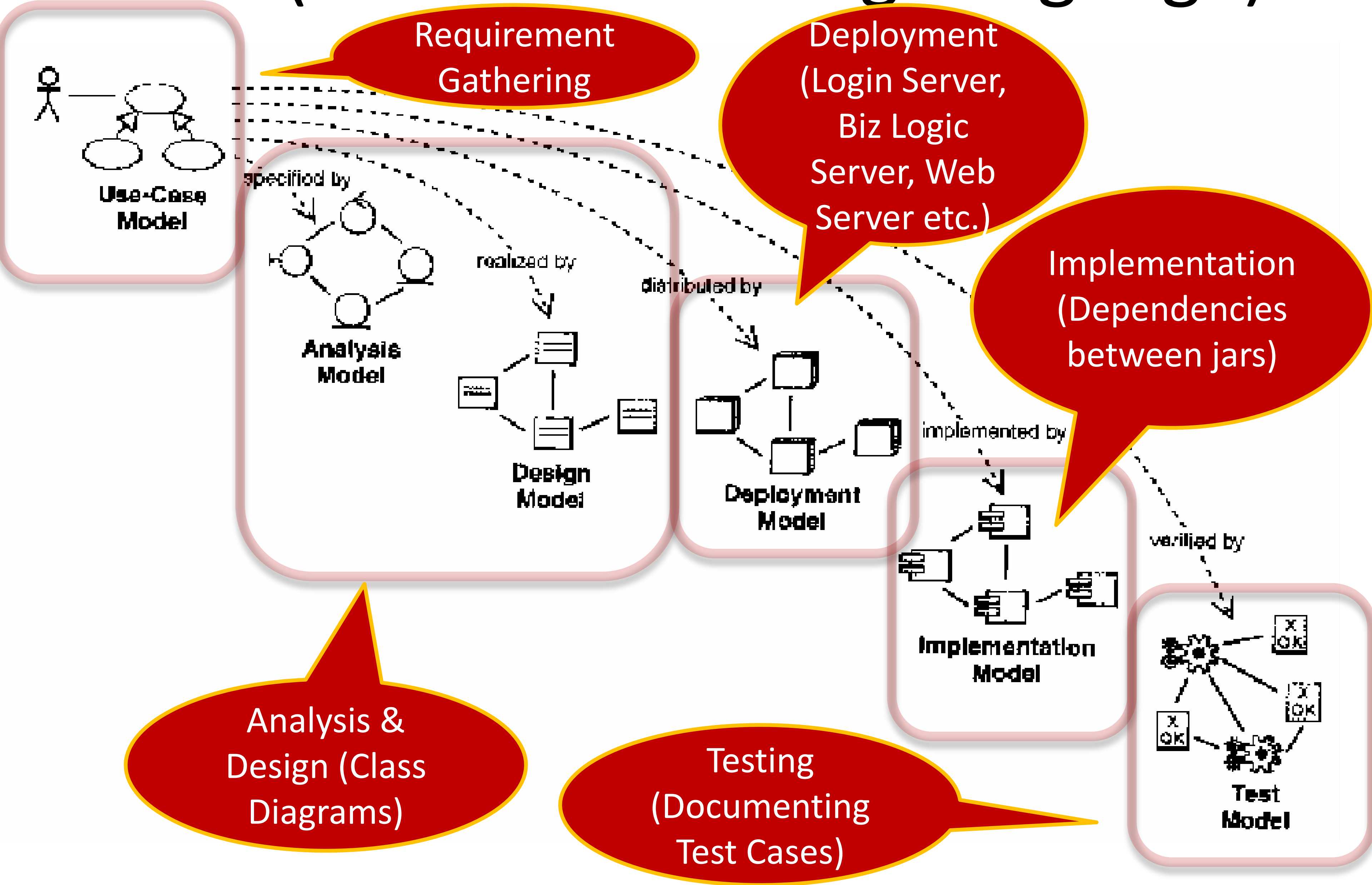
USER STORIES with HIGHEST priority selected for the next SPRINT, based on team size and skill level

After each SPRINT we deliver a SHIPPABLE PRODUCT INCREMENT

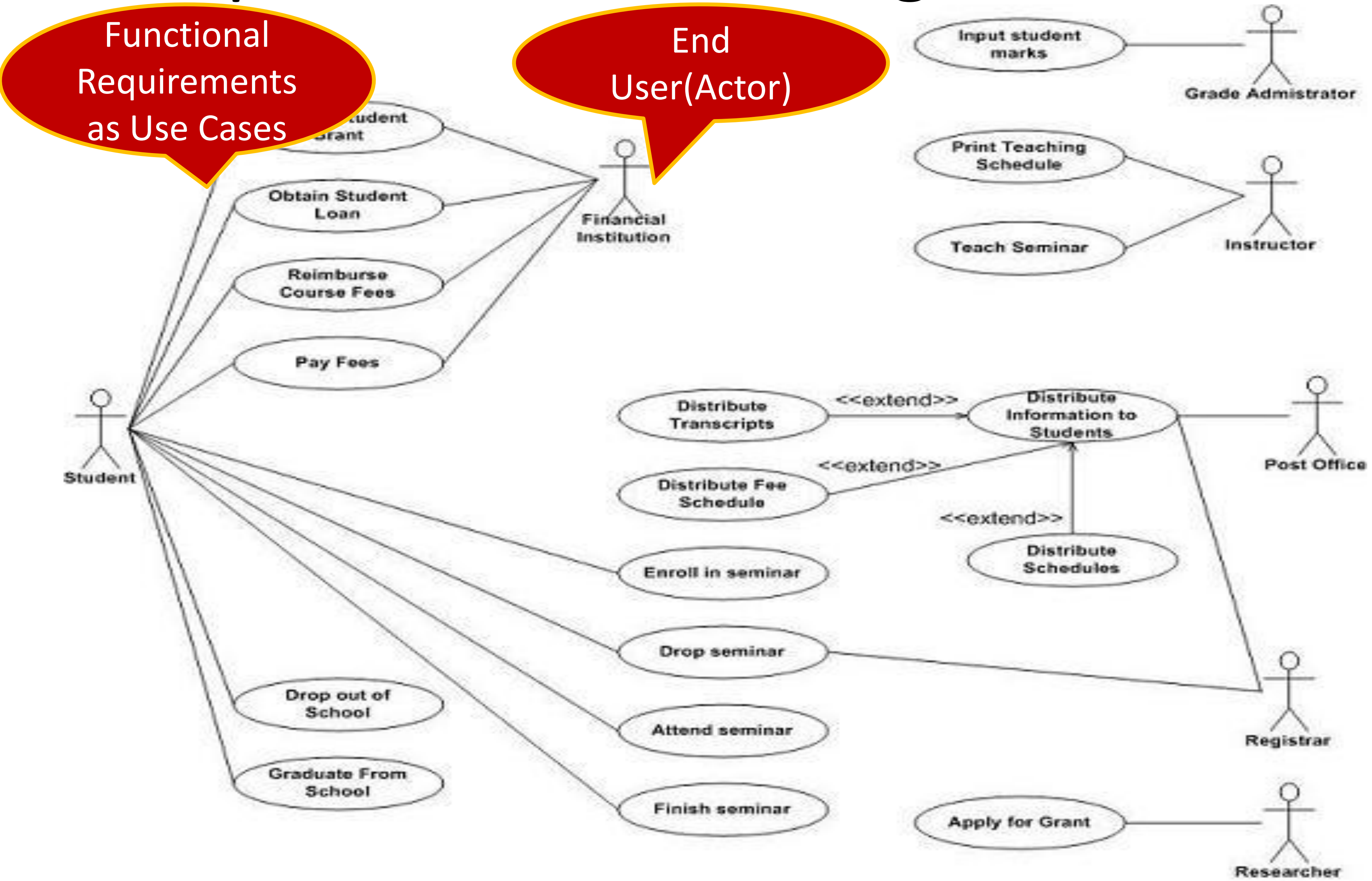


24 hours
Daily Scrum Meeting
30 days
Backlog tasks expanded by team
Sprint Backlog
Product Backlog As prioritized by Product Owner
Potentially Shippable Product Increment
Source: Adapted from Agile Software Development with Scrum by Ken Schwaber and Mike Beedle.

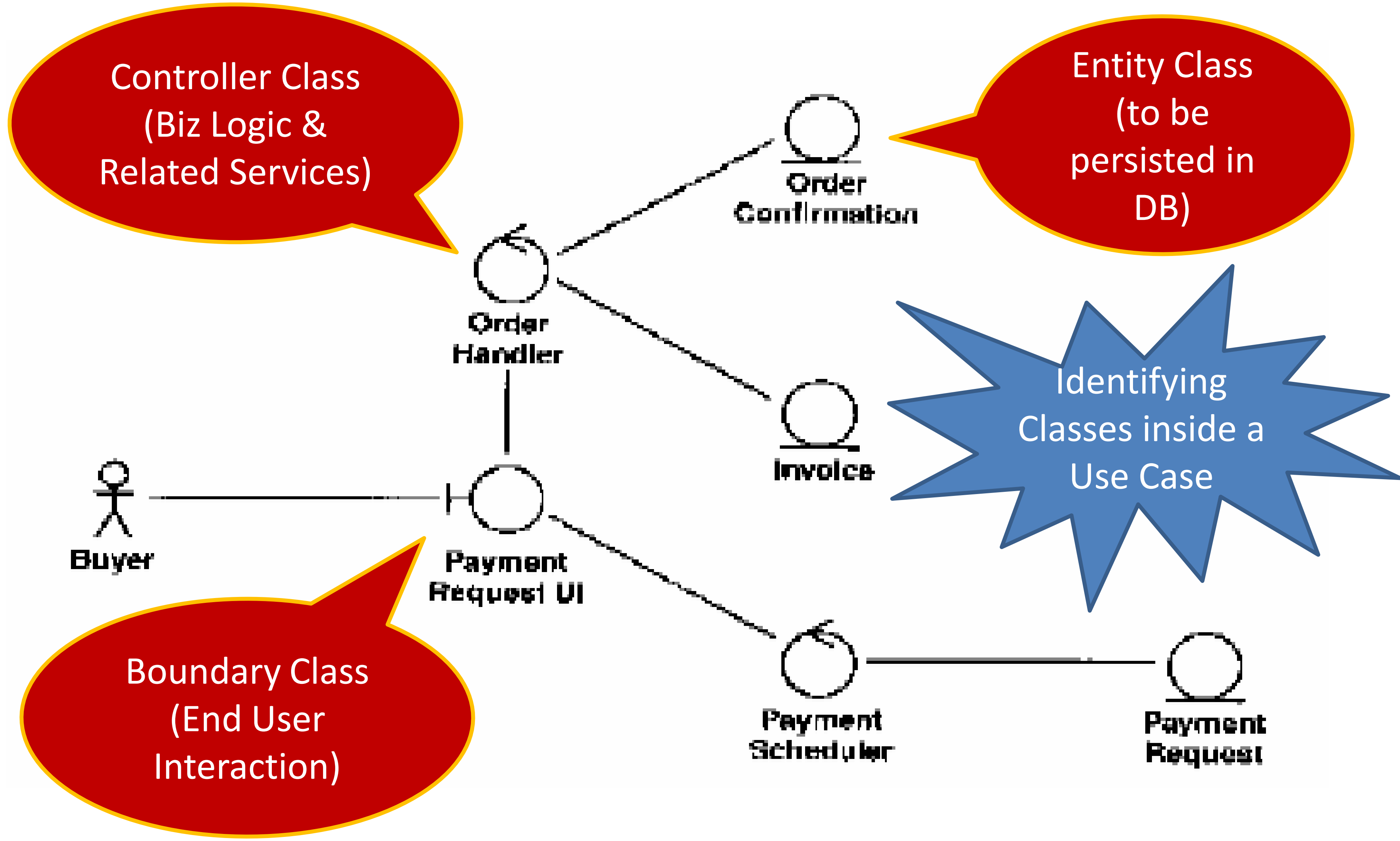
UML (Unified Modeling Language)



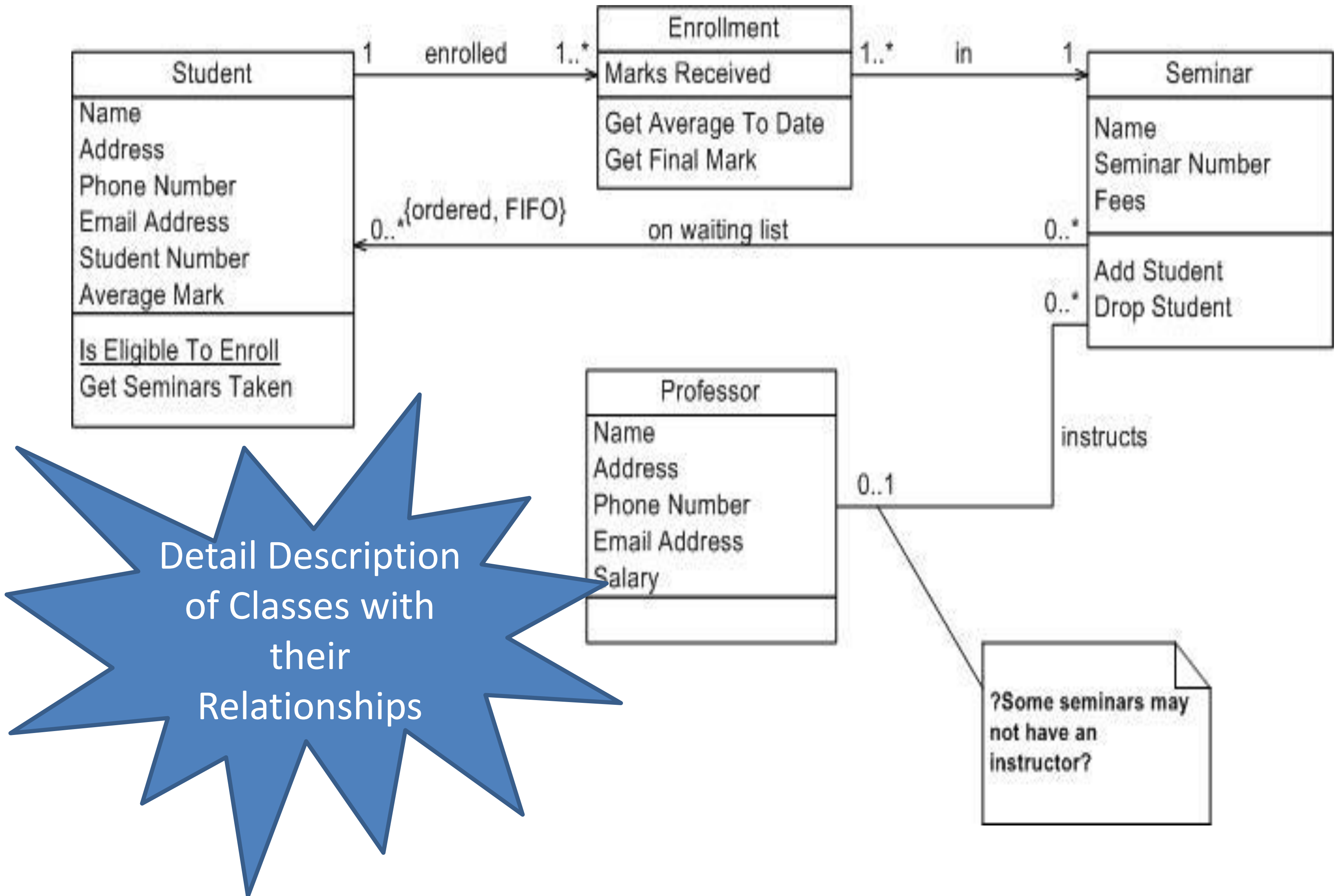
Requirement Gathering with UML



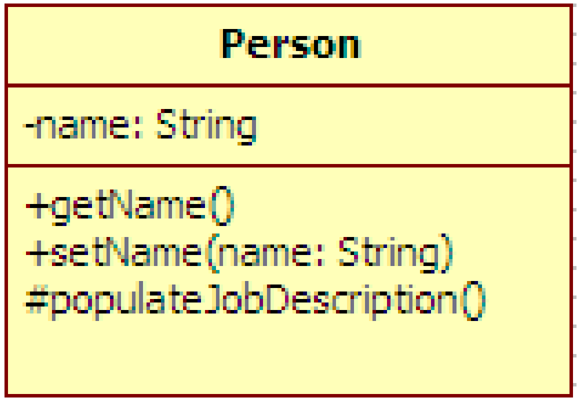
Analysis with UML

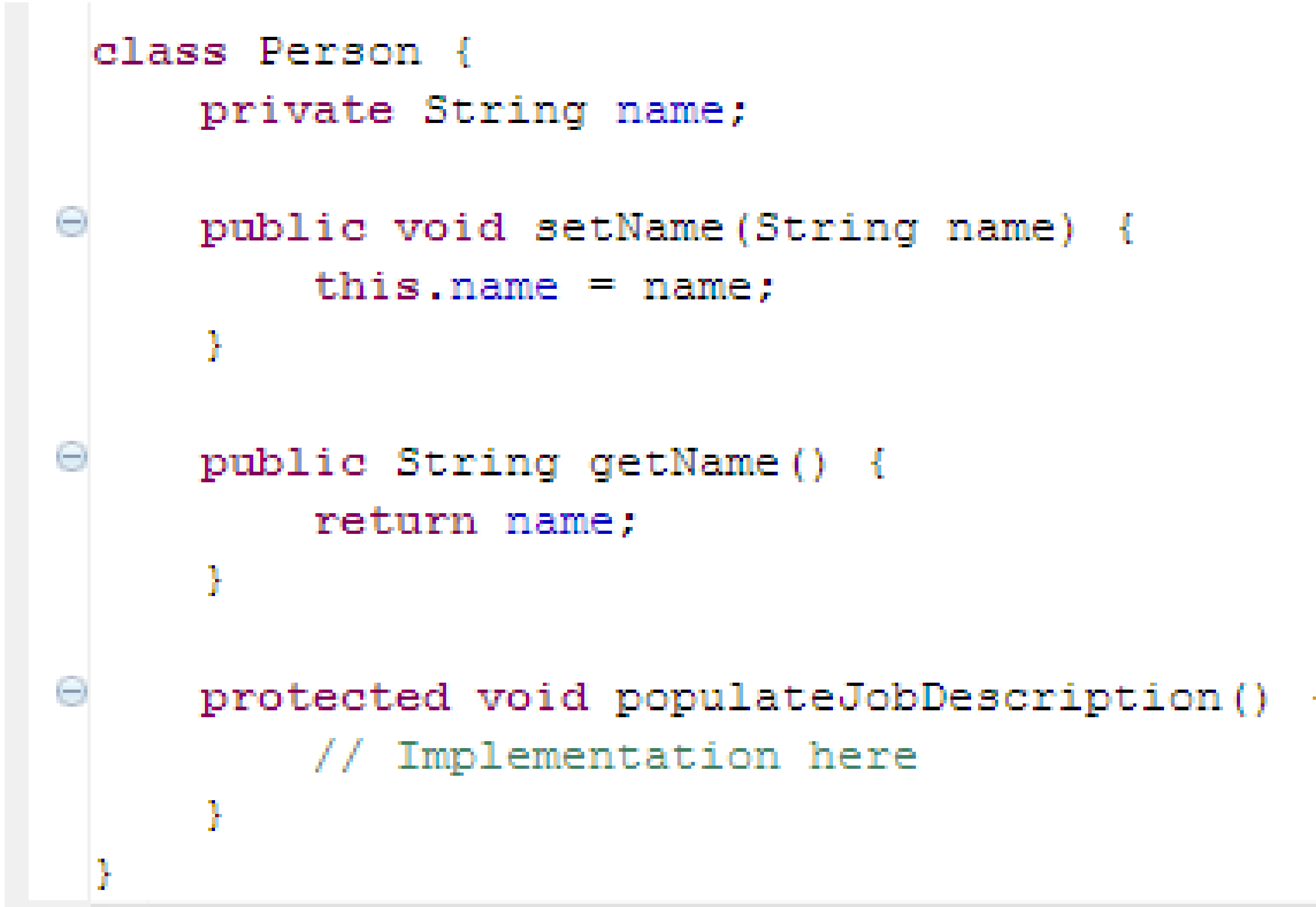

Controller Class (Biz Logic & Related Services)
Entity Class (to be persisted in DB)
Order Confirmation
Order Handler
Invoice
Identifying Classes inside a Use Case
Buyer
Payment Request UI
Boundary Class (End User Interaction)
Payment Scheduler
Payment Request

Design with UML



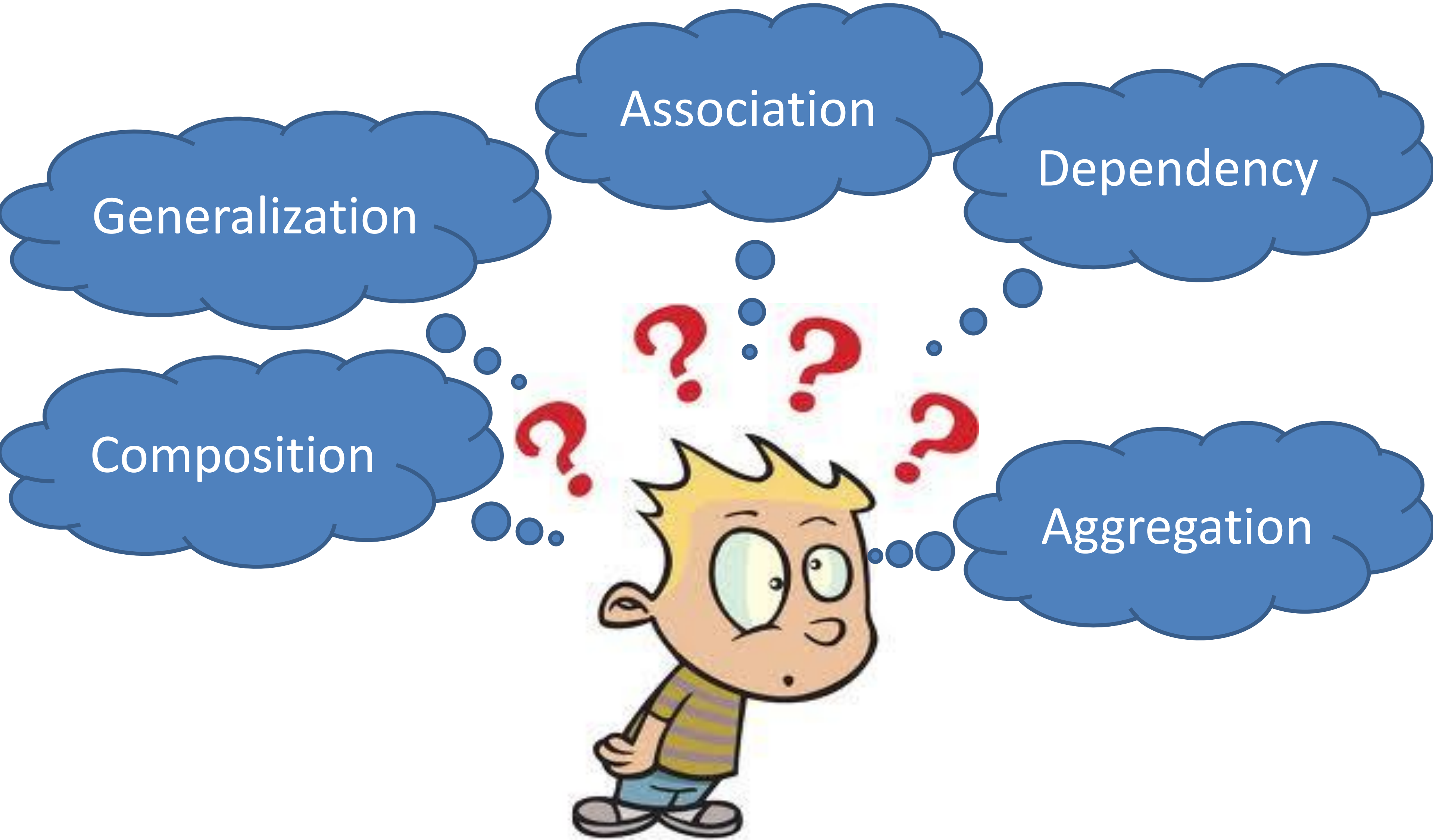
UML Class





```
class Person {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    protected void populateJobDescription()
        // Implementation here
    }
}
```

Class Diagram Relationships in Detail



Class Diagram Relationships in Detail

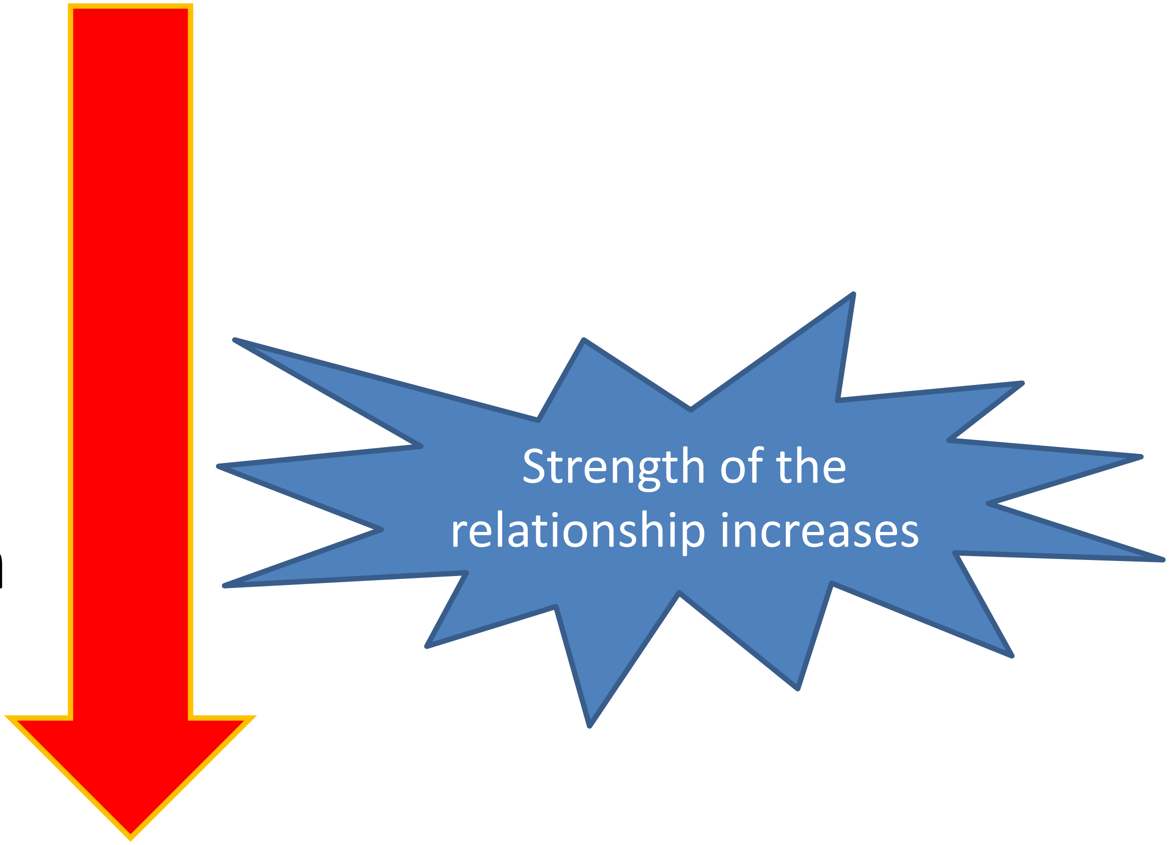
Dependency

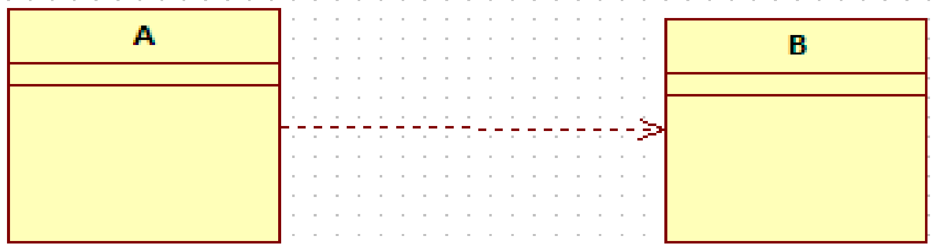
Association

Aggregation

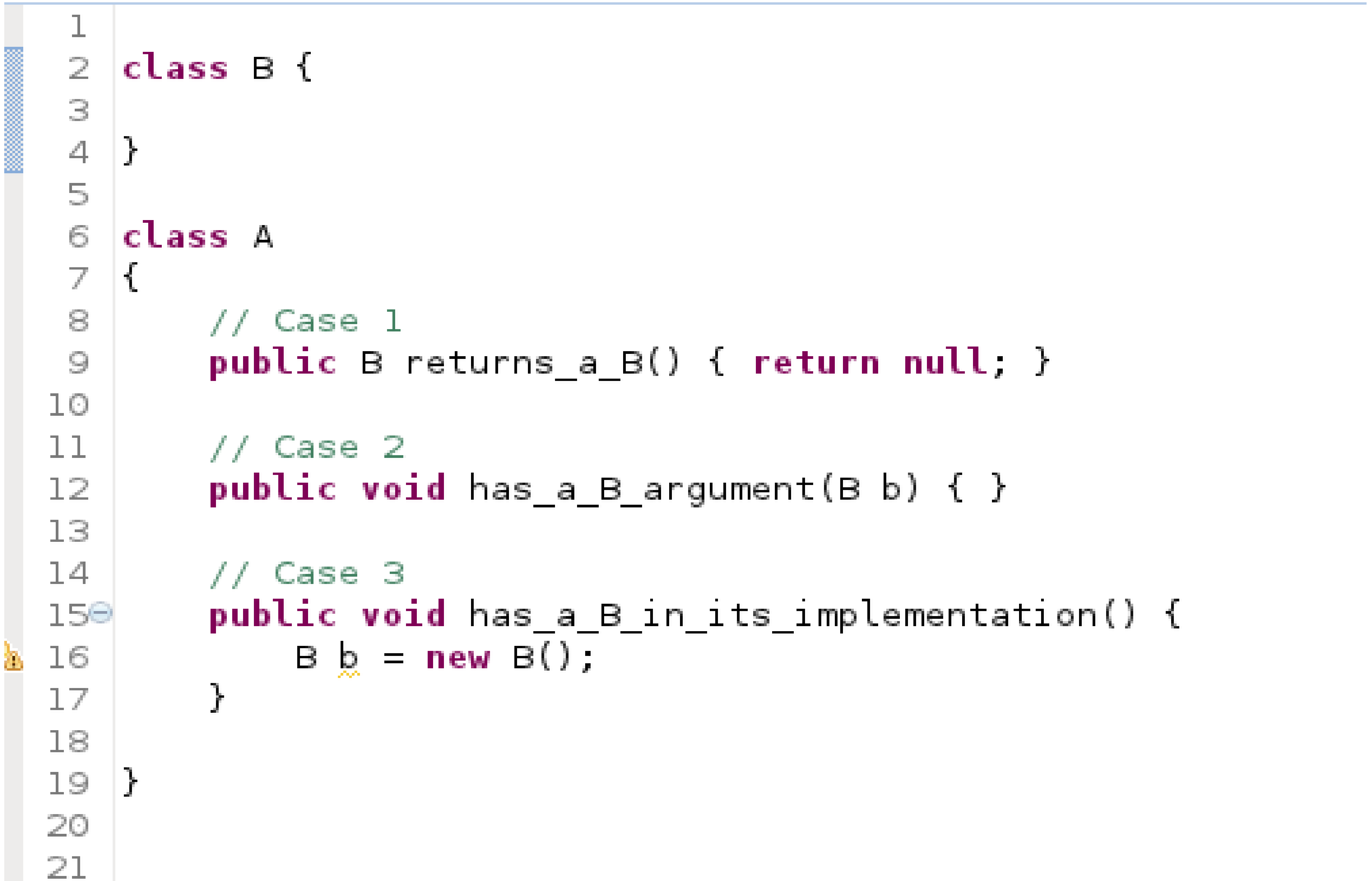
Composition

Generalization



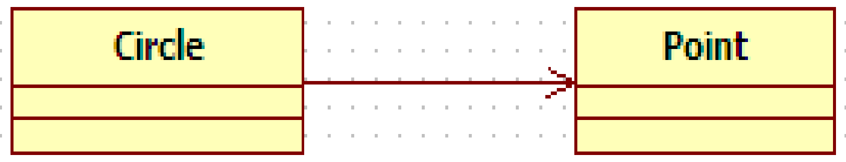


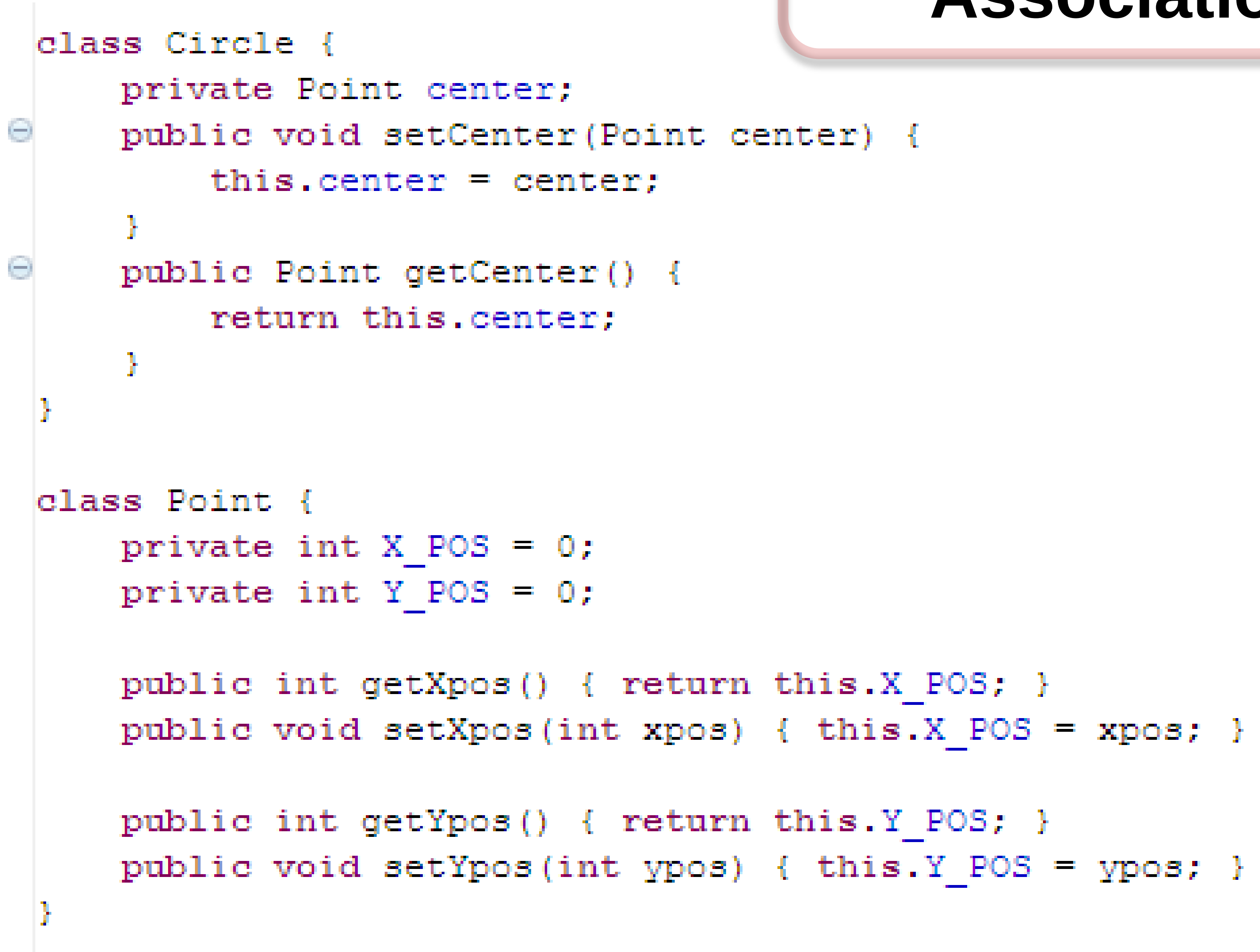
Dependency



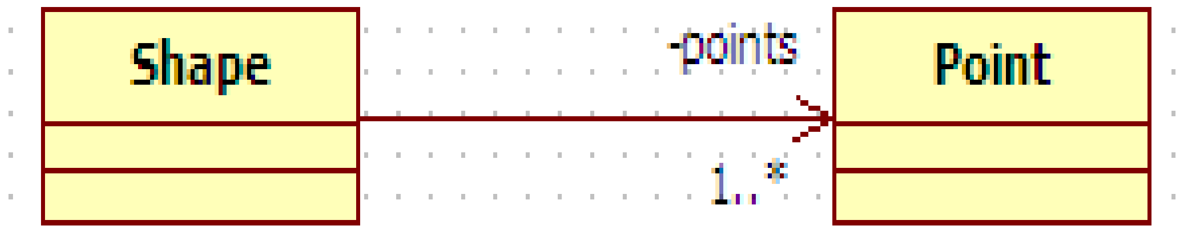
```
class B {

}

class A
{
	// Case 1
	public B returns_a_B() { return null; }

	// Case 2
	public void has_a_B_argument(B b) { }

	// Case 3
	public void has_a_B_in_its_implementation() {
		B b = new B();
	}

}


```

Uni-Directional Association



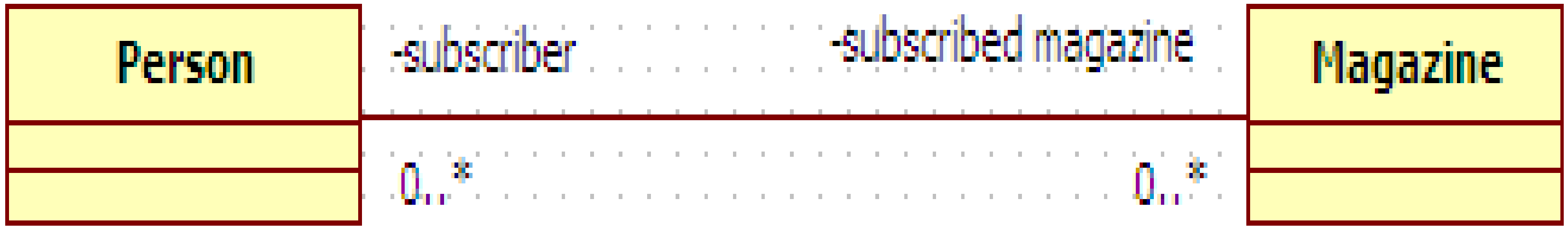


```
class Circle {
     private Point center;
     public void setCenter(Point center) {
          this.center = center;
     }
     public Point getCenter() {
          return this.center;
     }
}

class Point {
     private int X_POS = 0;
     private int Y_POS = 0;

     public int getXpos() { return this.X_POS; }
     public void setXpos(int xpos) { this.X_POS = xpos; }

     public int getYpos() { return this.Y_POS; }
     public void setYpos(int ypos) { this.Y_POS = ypos; }
}
```



```
import java.util.List;

class Shape {
    private List<Point> points;
    public void setPoints(List<Point> points) {
        this.points = points;
    }
    public List<Point> getPoints() {
        return this.points;
    }
}

class Point {
    private int X_POS = 0;
    private int Y_POS = 0;

    public int getXpos() { return this.X_POS; }
    public void setXpos(int xpos) { this.X_POS = xpos; }

    public int getYpos() { return this.Y_POS; }
    public void setYpos(int ypos) { this.Y_POS = ypos; }
}
```

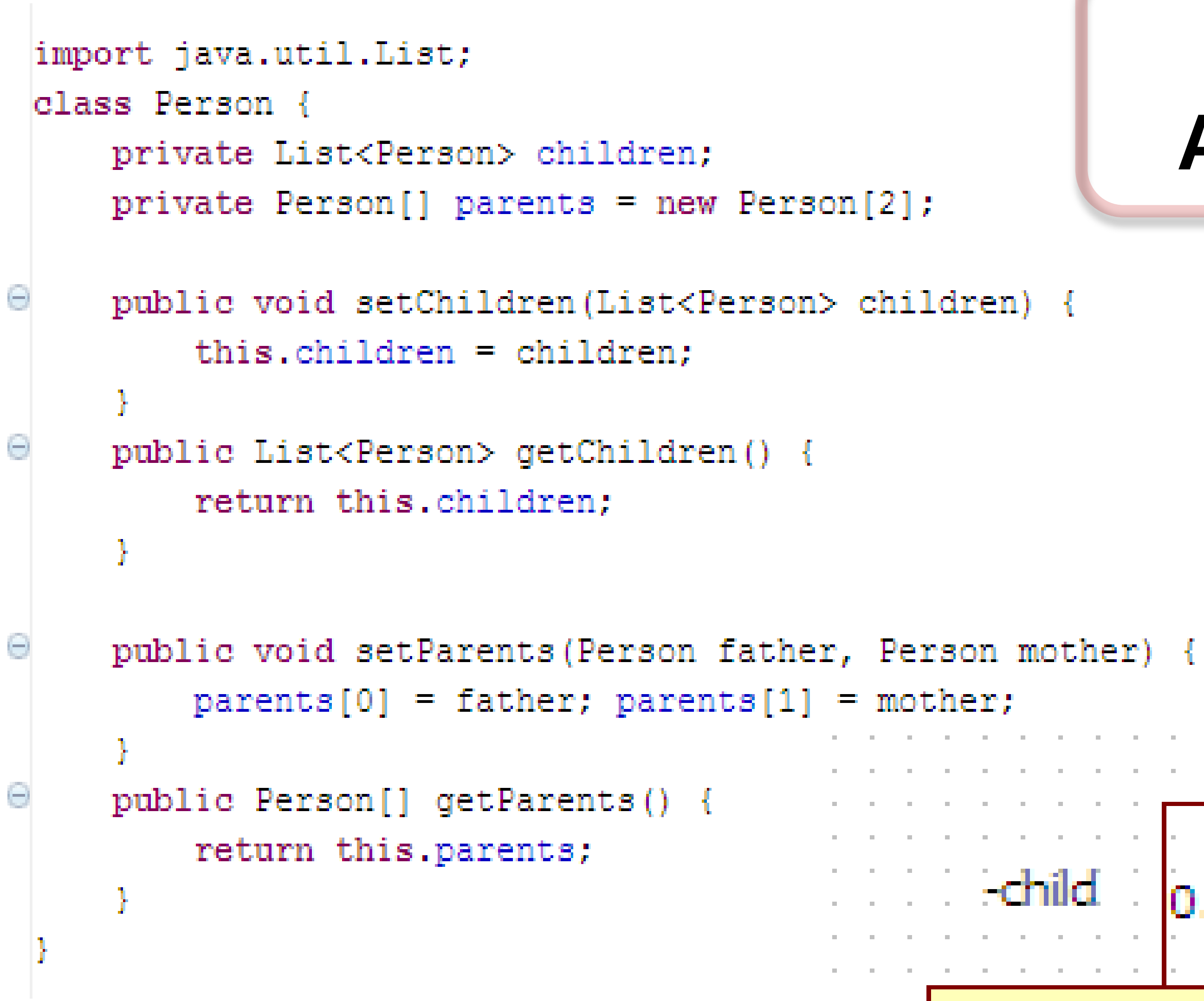
Association with Multiplicity



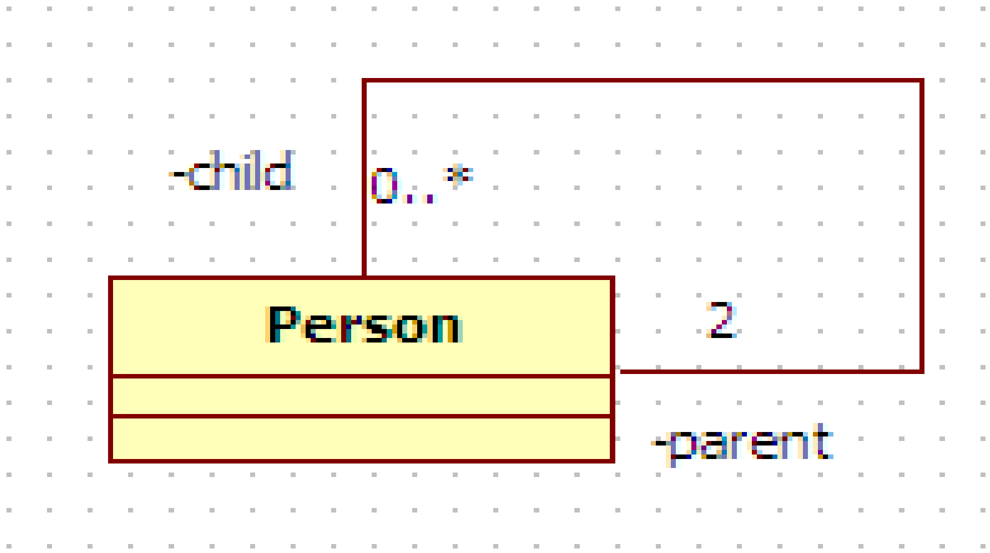
```
class Person {
    private List<Magazine> subscribedMagazines;
    public void setSubscribedMagazines(List<Magazine> subscribedMagazines) {
        this.subscribedMagazines = subscribedMagazines;
    }
    public List<Magazine> getSubscribedMagazines() {
        return this.subscribedMagazines;
    }
}

class Magazine {
    private List<Person> subscribers;
    public void setSubscribers(List<Person> subscribers) {
        this.subscribers = subscribers;
    }
    public List<Person> getSubscribers() {
        return this.subscribers;
    }
}
```

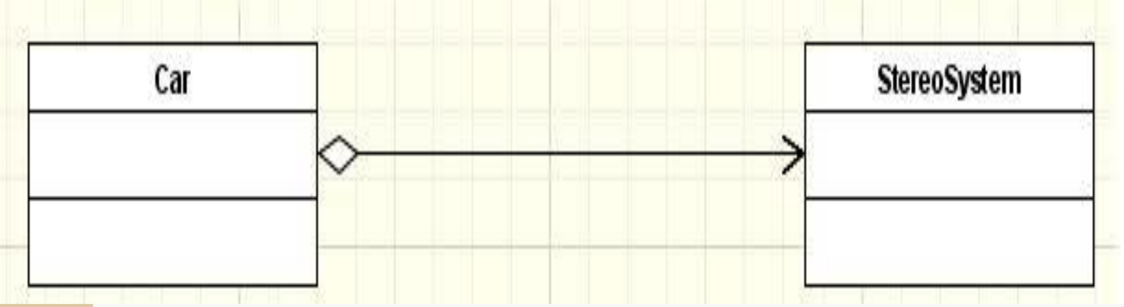
Bi-Directional Association

Reflexive Association

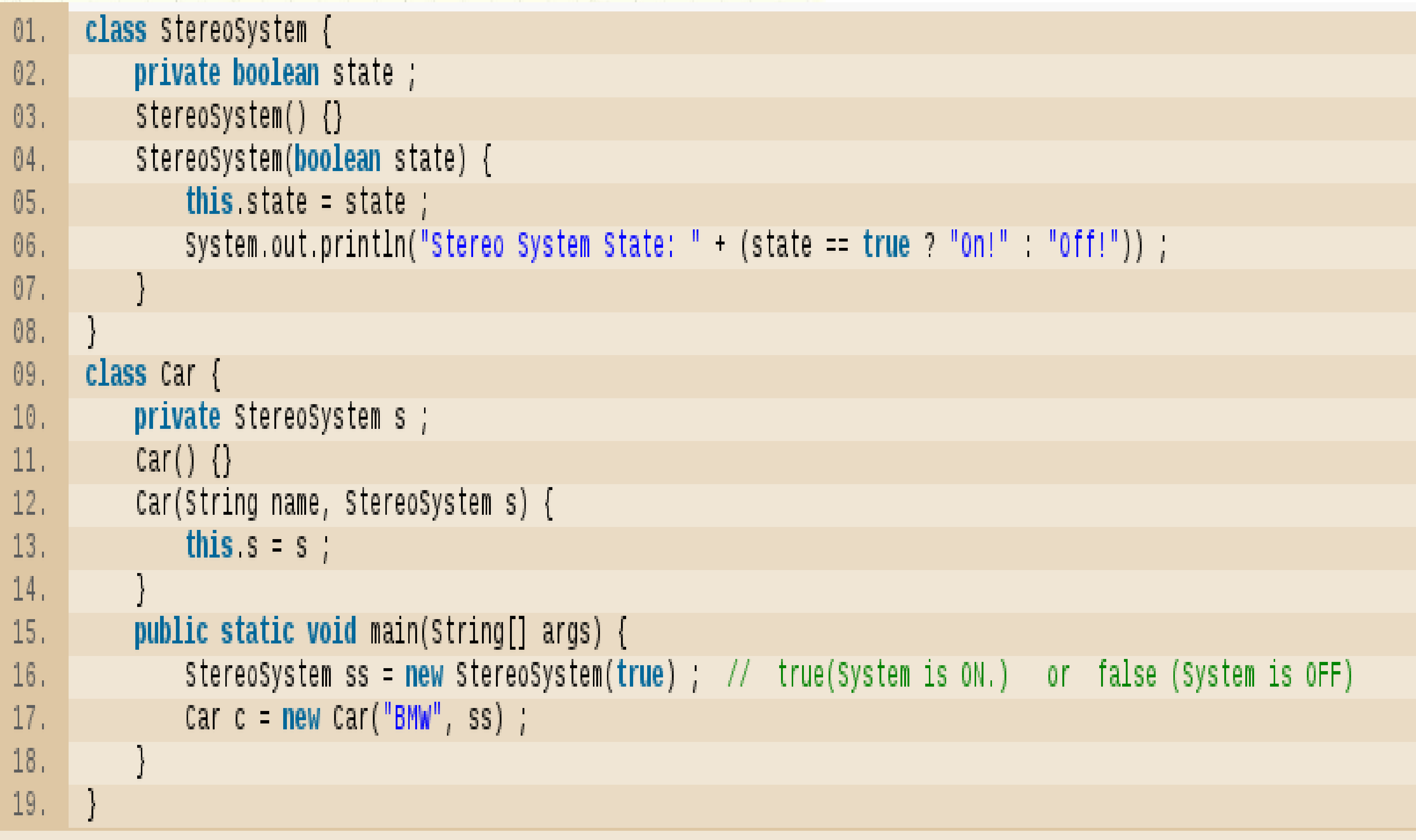


```
import java.util.List;
class Person {
    private List<Person> children;
    private Person[] parents = new Person[2];

    public void setChildren(List<Person> children) {
        this.children = children;
    }
    public List<Person> getChildren() {
        return this.children;
    }

    public void setParents(Person father, Person mother) {
        parents[0] = father; parents[1] = mother;
    }
    public Person[] getParents() {
        return this.parents;
    }
}
```





Aggregation



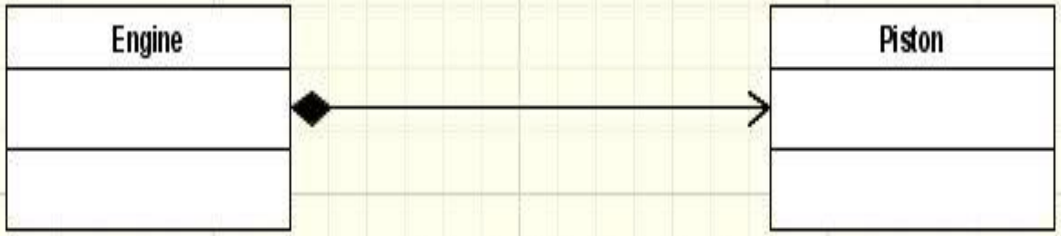
```
class StereoSystem {
    private boolean state ;
    StereoSystem() {}
    StereoSystem(boolean state) {
        this.state = state ;
        System.out.println("Stereo System State: " + (state == true ? "On!" : "Off!")) ;
    }
}
class Car {
    private StereoSystem s ;
    Car() {}
    Car(String name, StereoSystem s) {
        this.s = s ;
    }
    public static void main(String[] args) {
        StereoSystem ss = new StereoSystem(true) ;  //  true(System is ON.)   or  false (System is OFF)
        Car c = new Car("BMW", ss) ;
    }
}
```

Business Requirements behind Aggregation

- A stereo system can be sold separately without a car.
- A car can be sold without a stereo system.
- If a car is bundled with a stereo system both has to be sold together.

Business Requirements behind Aggregation (contd …)

- If a stereo system is broken after the purchase, the car can still be used.
  - Customer will purchase a NEW stereo system from us.
- If the car is broken after the purchase, the stereo system can still be used.
  - Customer might purchase a NEW car from us WITHOUT a stereo system.



Composition

```
import java.util.Date ;
class Piston {
    private Date pistonDate ;
    Piston() {
        pistonDate = new Date() ;
        System.out.println("Manufactured Date :: " + pistonDate) ;
    }
}
class Engine {
    private Piston piston ;
    Engine() {
        piston = new Piston() ;
    }
    public static void main(String[] args) {
        Engine engine = new Engine() ;
    }
}
```

Business Requirements behind Composition

- There can't be an engine without a piston.
- We don't sell pistons separately, we sell only engines.
- Thus there can't be a piston without an engine.

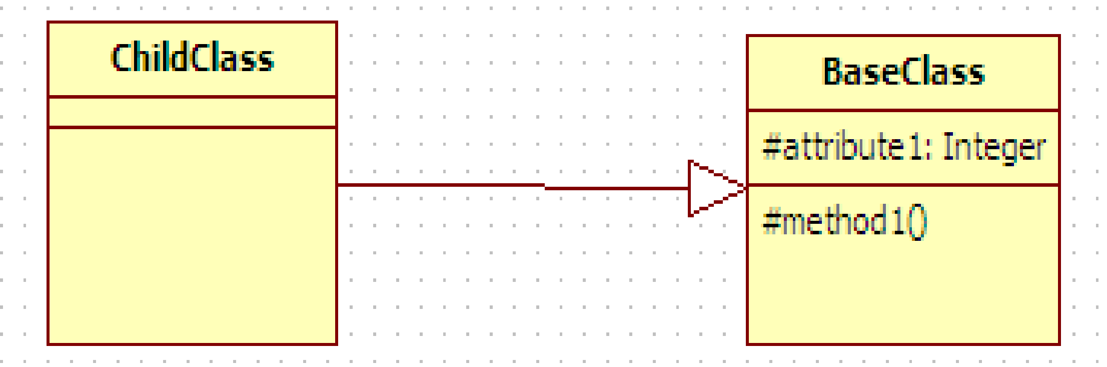
Business Requirements behind Composition (contd …)

- If an engine is broken due to some internal error (even other THAN in the PISTON), then customer has to buy a NEW engine from us.
- If the piston is broken and can't be repaired, the customer has to buy a NEW engine from us.

Aggregation vs. Composition vs. Business Domain

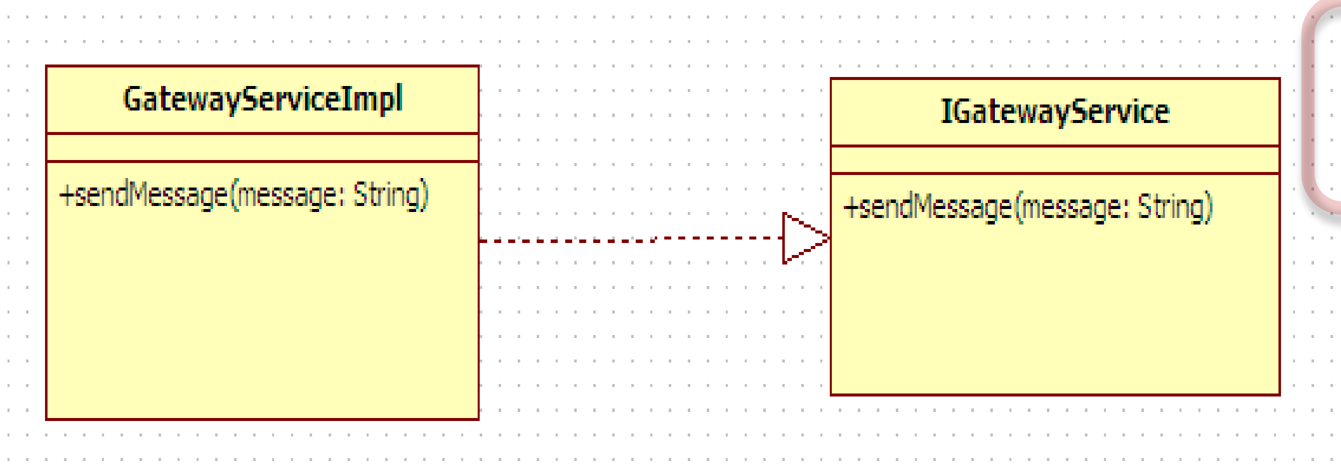
- In the piston, engine example; if the inventory sells pistons separately, then a piston can exist without an engine
- Thus the relationship will become an **aggregation,** not a composition

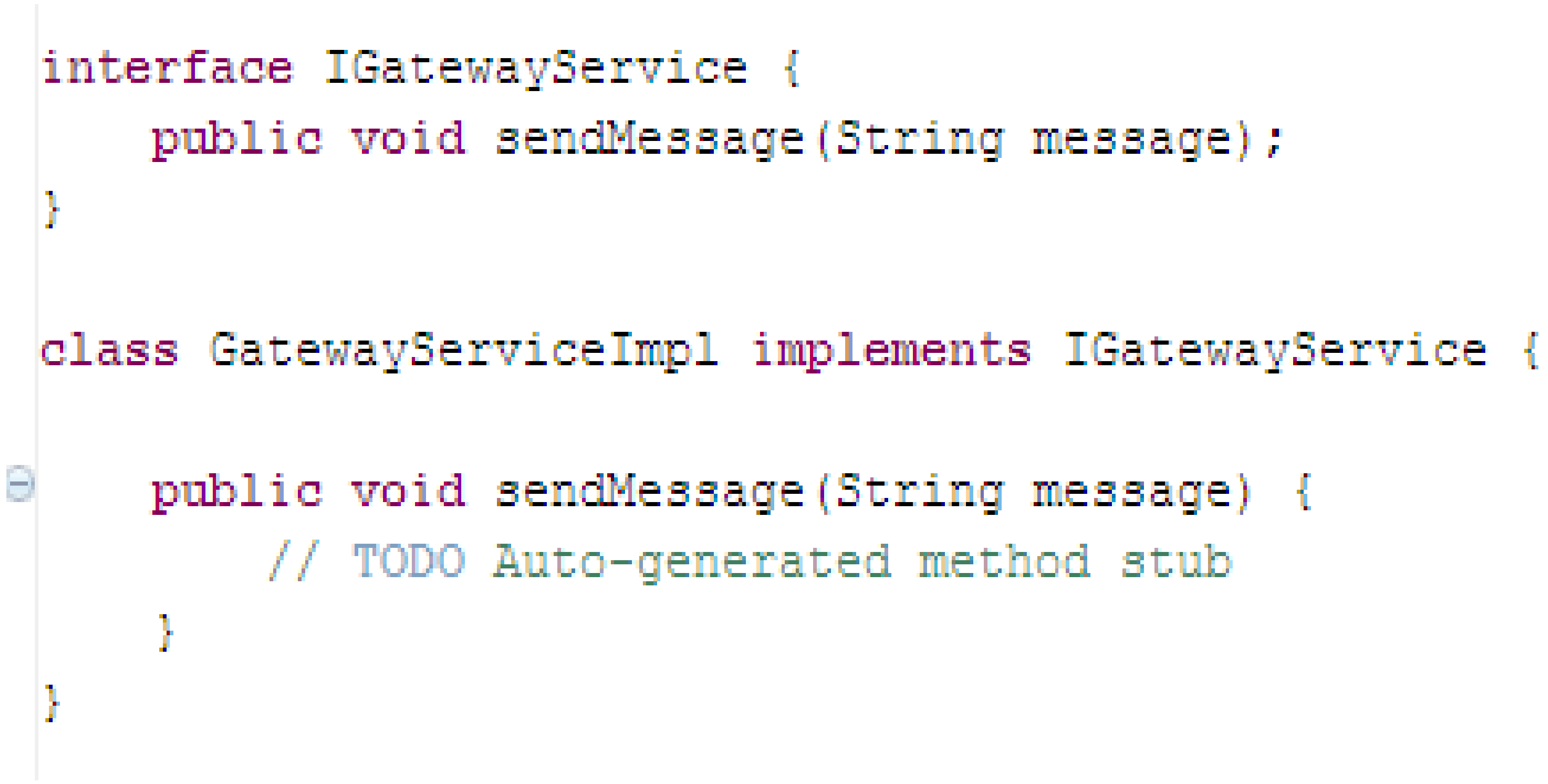
Generalization



```
class BaseClass {
    protected int attribute1;

    protected void method1() {

    }
}

class ChildClass extends BaseClass {
    // what if attribute1 is changed in BaseClass?
    // what if method1 is changed in BaseClass?
}
```

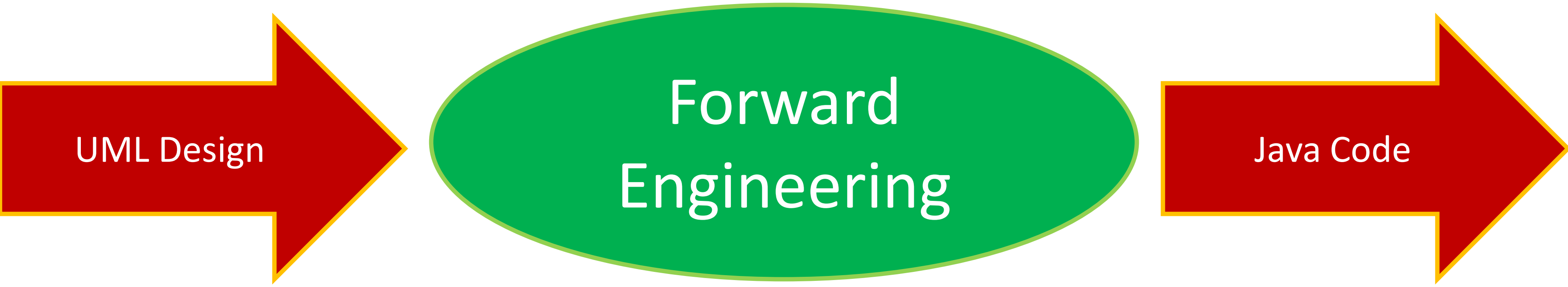
Realization

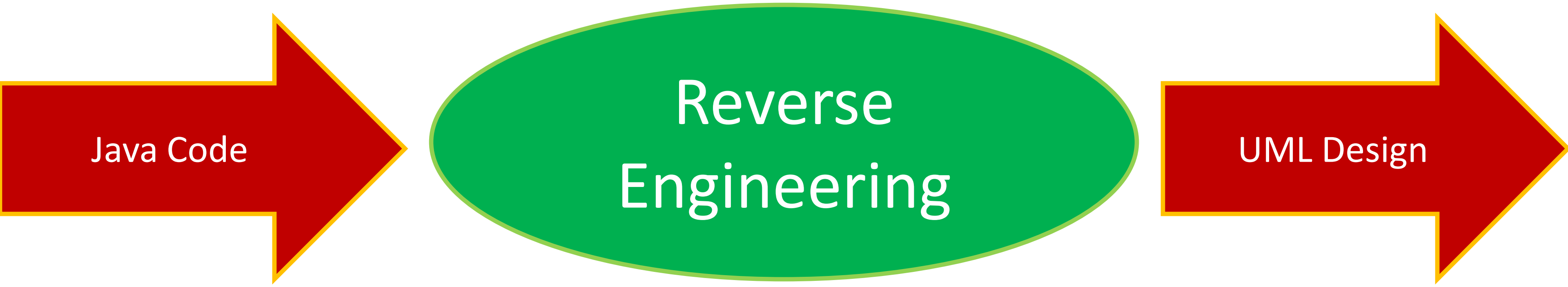




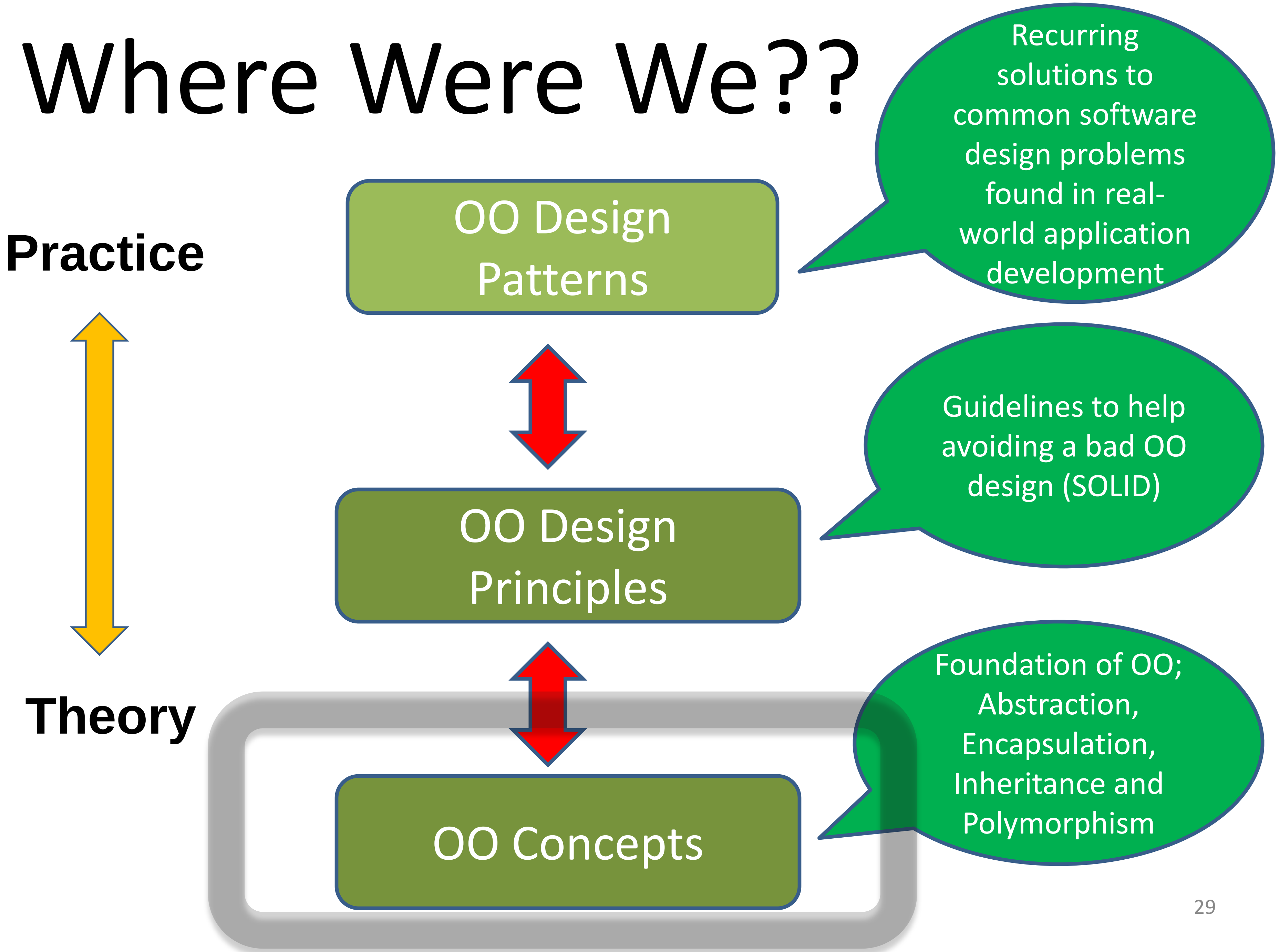
```
interface IGatewayService {
    public void sendMessage(String message);
}

class GatewayServiceImpl implements IGatewayService {

    public void sendMessage(String message) {
        // TODO Auto-generated method stub
    }
}
```

Forward Engineering vs. Reverse Engineering

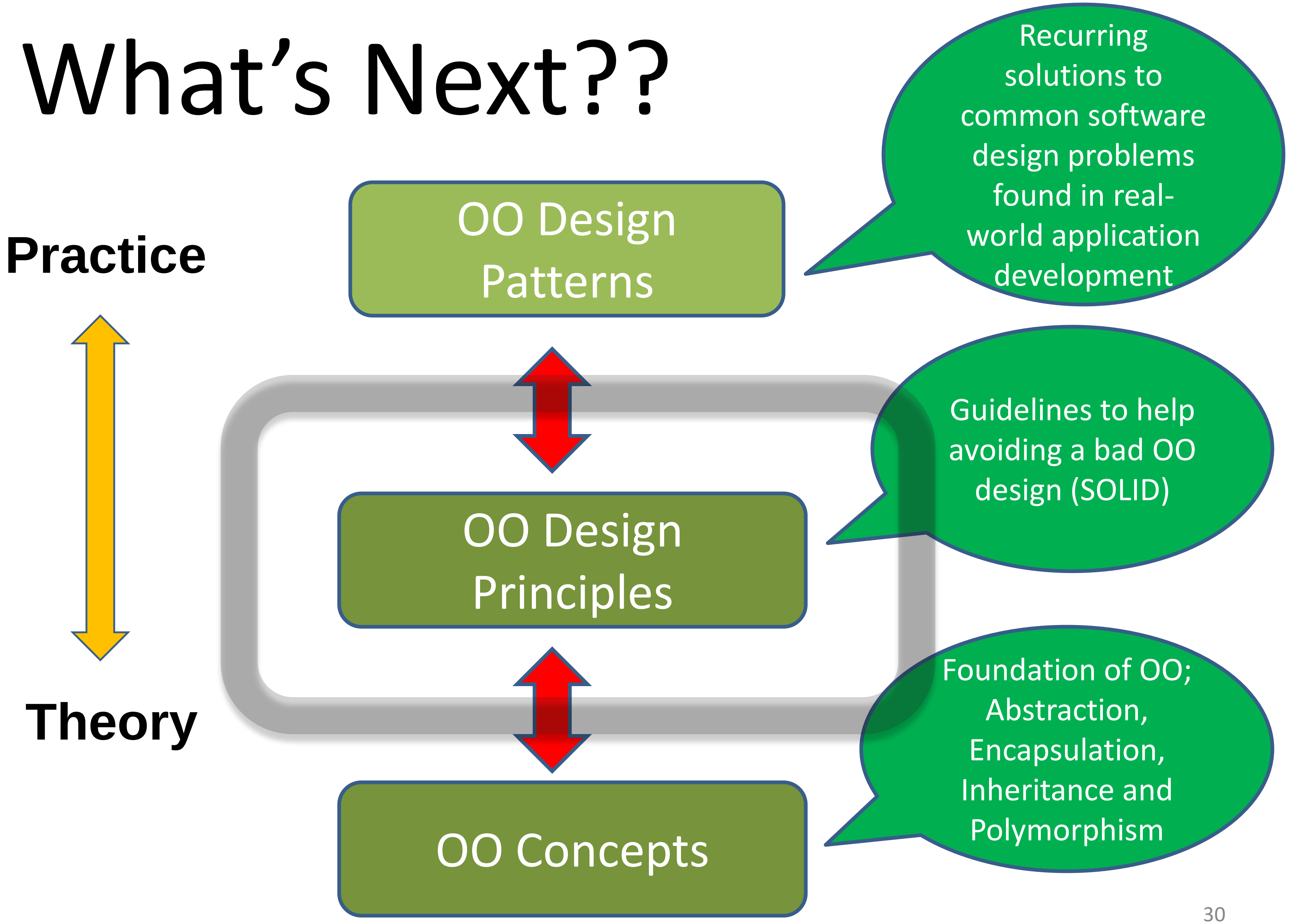




Where Were We??


Practice
Theory
OO Design Patterns
OO Design Principles
OO Concepts
Recurring solutions to common software design problems found in real-world application development
Guidelines to help avoiding a bad OO design (SOLID)
Foundation of OO; Abstraction, Encapsulation, Inheritance and Polymorphism

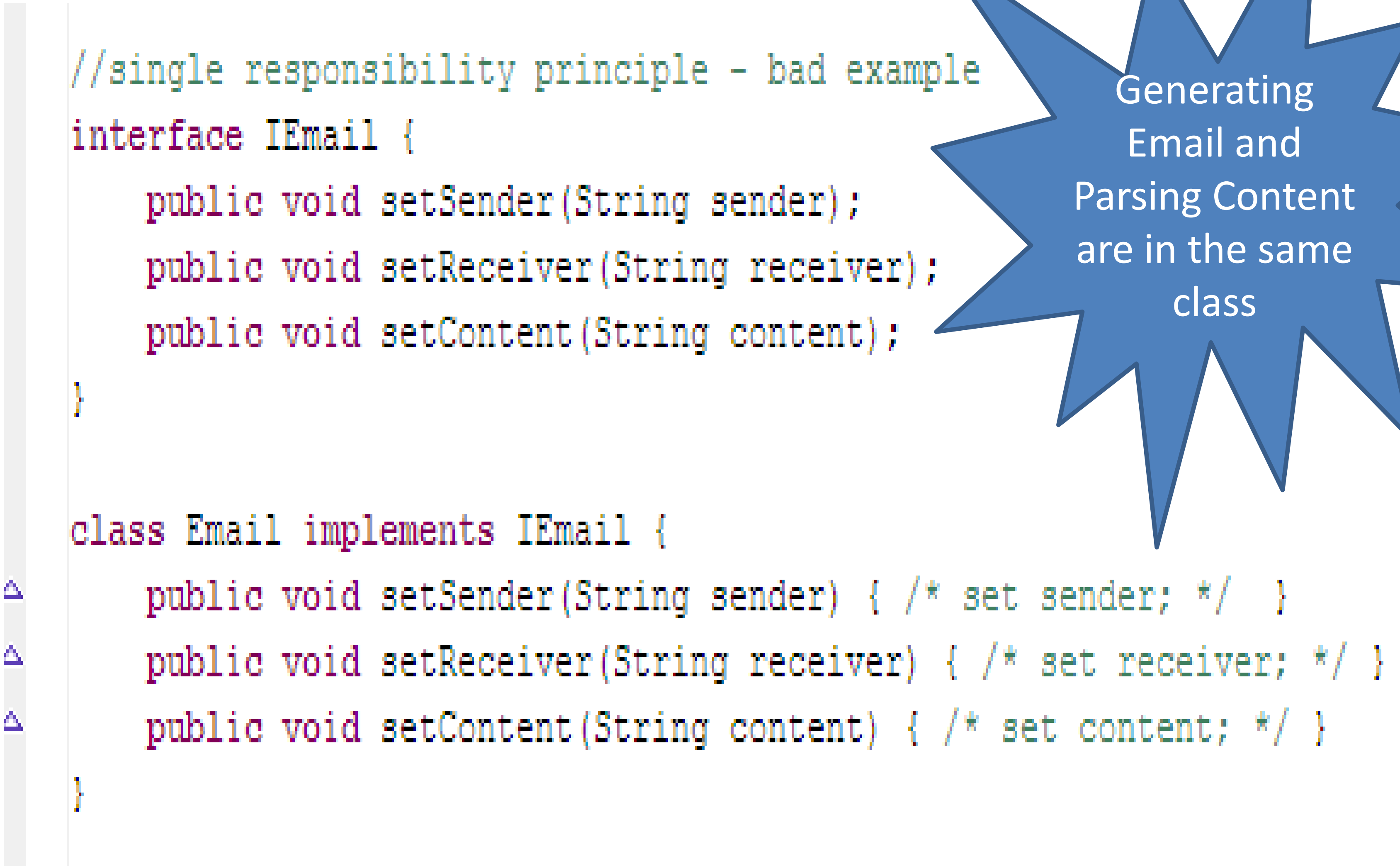
What's Next??


Practice
Theory
OO Design Patterns
OO Design Principles
OO Concepts
Recurring solutions to common software design problems found in real-world application development
Guidelines to help avoiding a bad OO design (SOLID)
Foundation of OO; Abstraction, Encapsulation, Inheritance and Polymorphism

OO Design Principles

- **SOLID**
  1. **S**ingle Responsibility Principle
  2. **O**pen-close Principle
  3. **L**iskov's Substitution Principle
  4. **I**nterface Segregation Principle
  5. **D**ependency Inversion Principle

Single Responsibility Principle

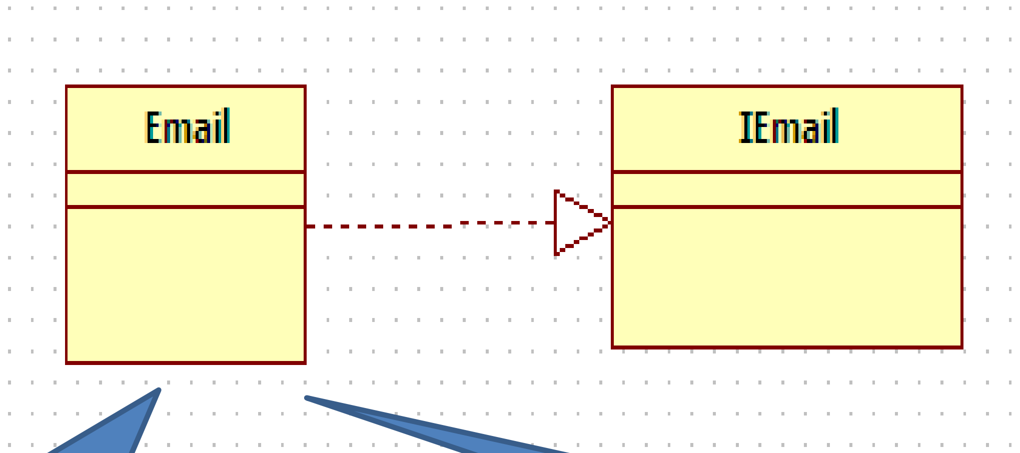


```
//single responsibility principle - bad example
interface IEmail {
    public void setSender(String sender);
    public void setReceiver(String receiver);
    public void setContent(String content);
}

class Email implements IEmail {
    public void setSender(String sender) { /* set sender; */  }
    public void setReceiver(String receiver) { /* set receiver; */ }
    public void setContent(String content) { /* set content; */ }
}
```

Generating Email and Parsing Content are in the same class

SRP Contd …

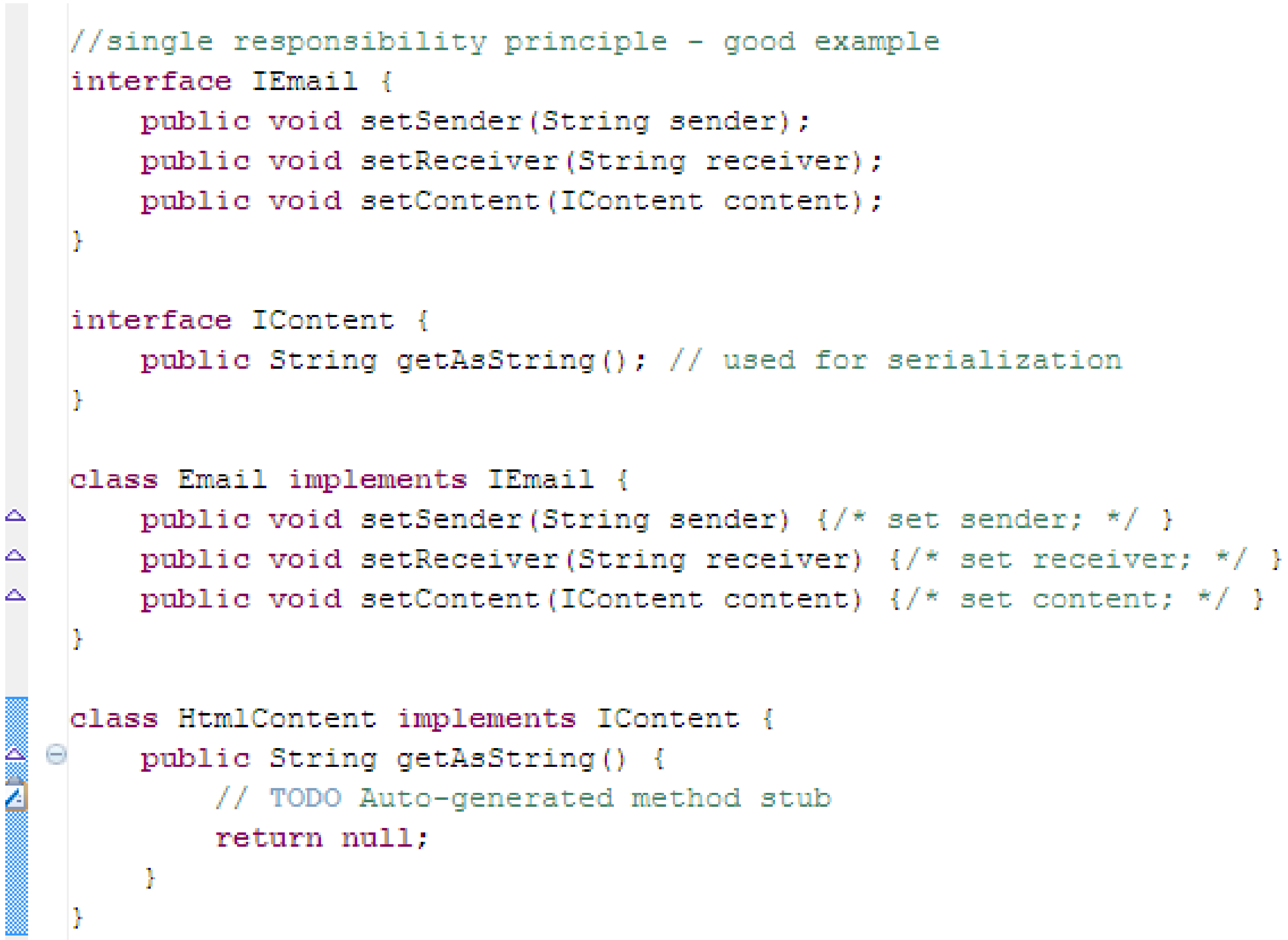
- A class should have ONLY **one reason** to **change**.



Need to change the implementation inside Email class, when you introduce a new content type (like html)

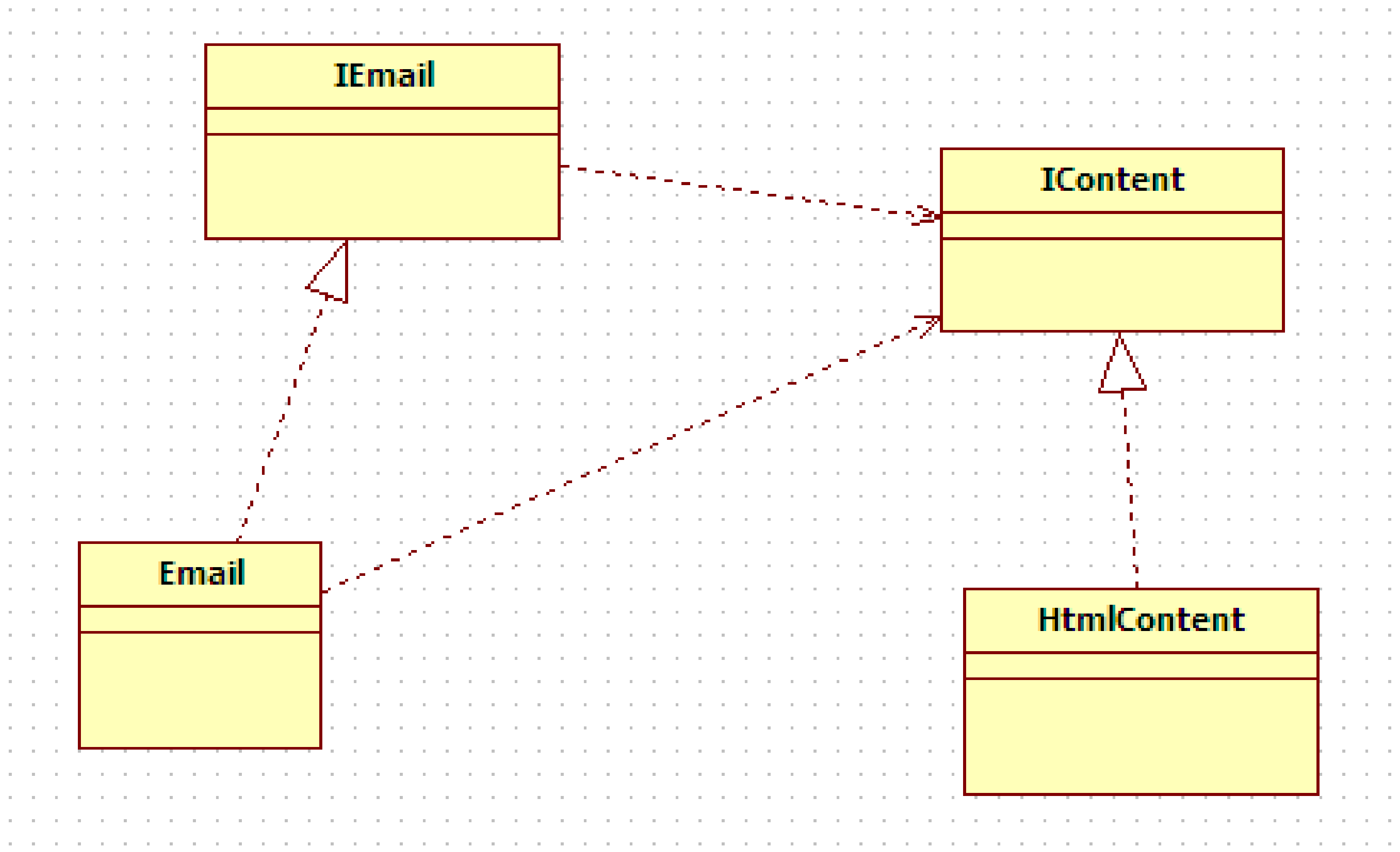
What if there were multiple implementations of IEmail and a new content type is introduced? Change both?

Refactoring with SRP

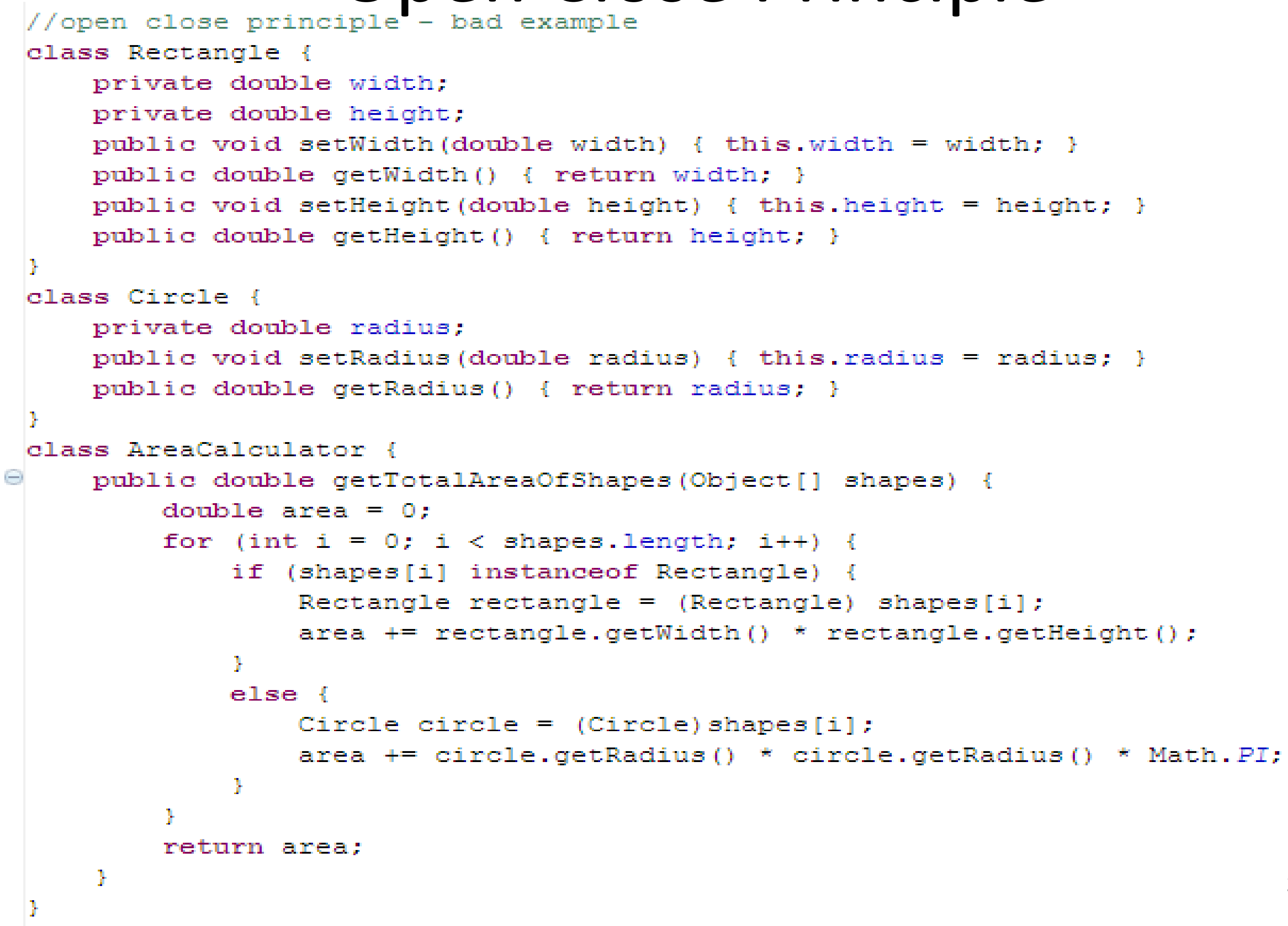


```
//single responsibility principle - good example
interface IEmail {
    public void setSender(String sender);
    public void setReceiver(String receiver);
    public void setContent(IContent content);
}

interface IContent {
    public String getAsString(); // used for serialization
}

class Email implements IEmail {
    public void setSender(String sender) {/* set sender; */ }
    public void setReceiver(String receiver) {/* set receiver; */ }
    public void setContent(IContent content) {/* set content; */ }
}

class HtmlContent implements IContent {
    public String getAsString() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

SRP Revised Design



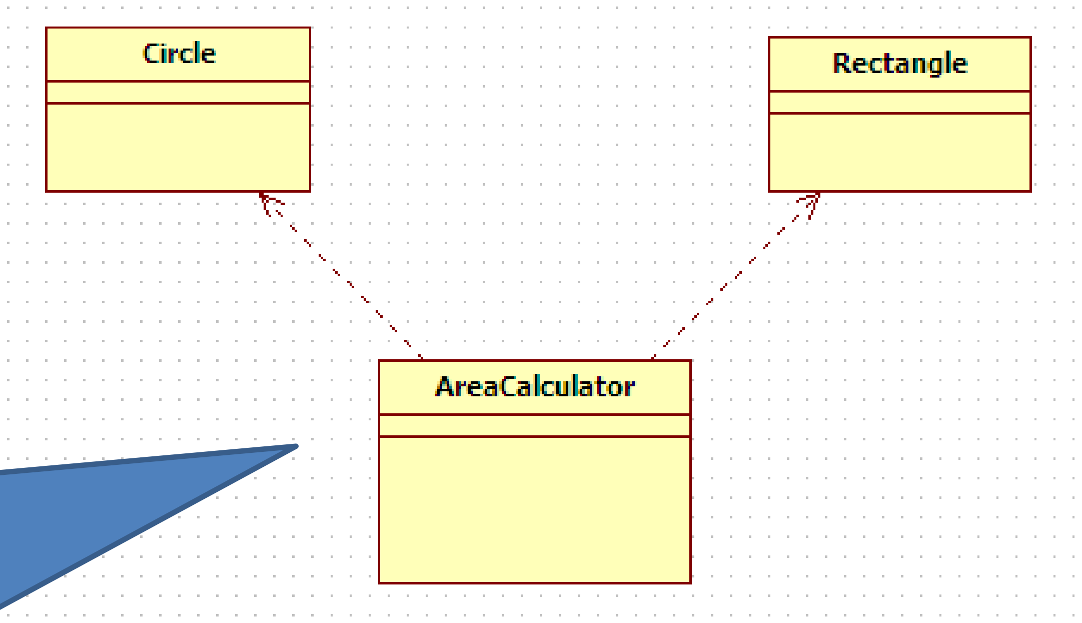
Open Close Principle



```
//open close principle - bad example
class Rectangle {
    private double width;
    private double height;
    public void setWidth(double width) { this.width = width; }
    public double getWidth() { return width; }
    public void setHeight(double height) { this.height = height; }
    public double getHeight() { return height; }
}
class Circle {
    private double radius;
    public void setRadius(double radius) { this.radius = radius; }
    public double getRadius() { return radius; }
}
class AreaCalculator {
    public double getTotalAreaOfShapes(Object[] shapes) {
        double area = 0;
        for (int i = 0; i < shapes.length; i++) {
            if (shapes[i] instanceof Rectangle) {
                Rectangle rectangle = (Rectangle) shapes[i];
                area += rectangle.getWidth() * rectangle.getHeight();
            }
            else {
                Circle circle = (Circle)shapes[i];
                area += circle.getRadius() * circle.getRadius() * Math.PI;
            }
        }
        return area;
    }
}
```

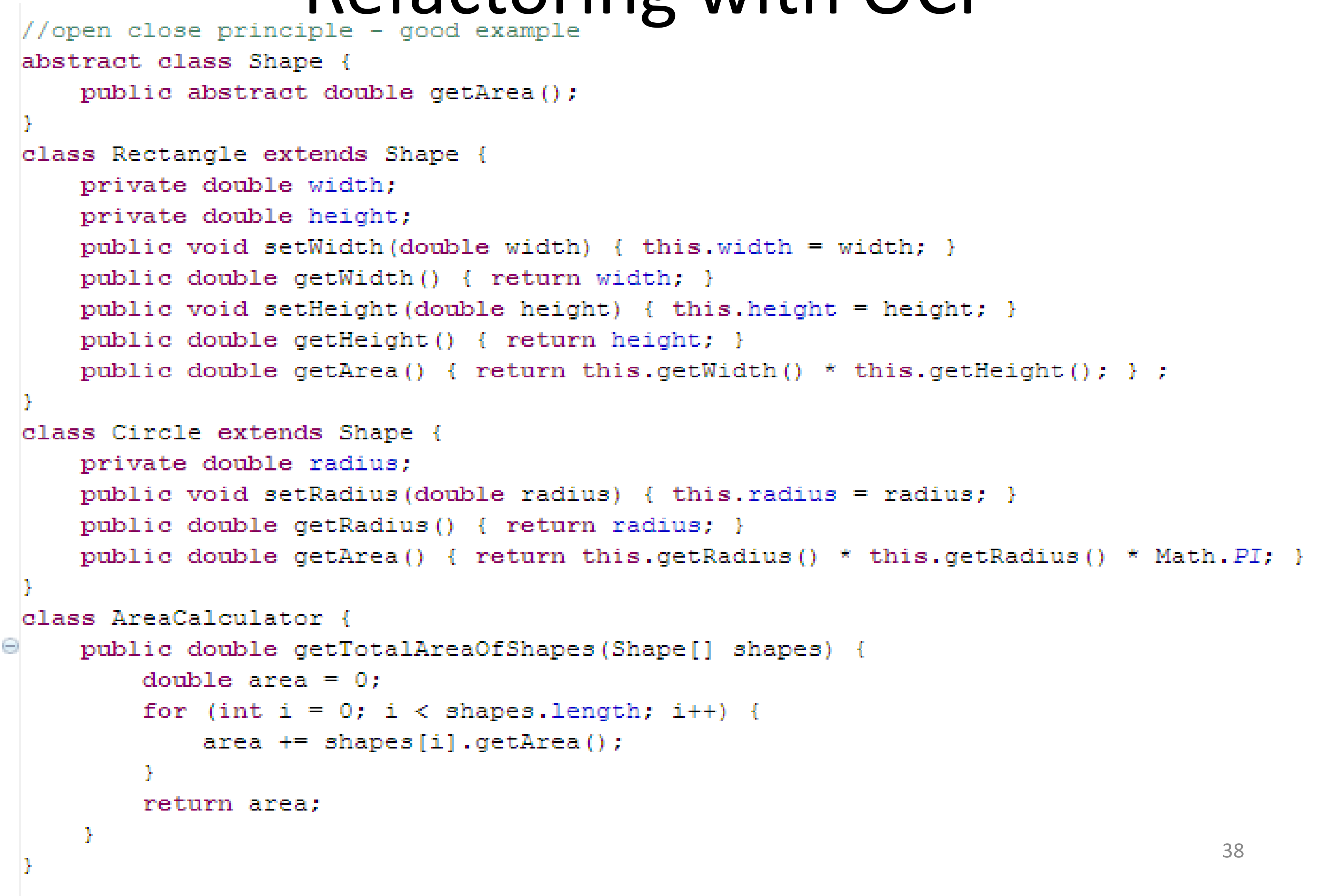
OCP Contd …

- Software entities like classes, modules and functions should be **open for extension** but **closed for modifications**.



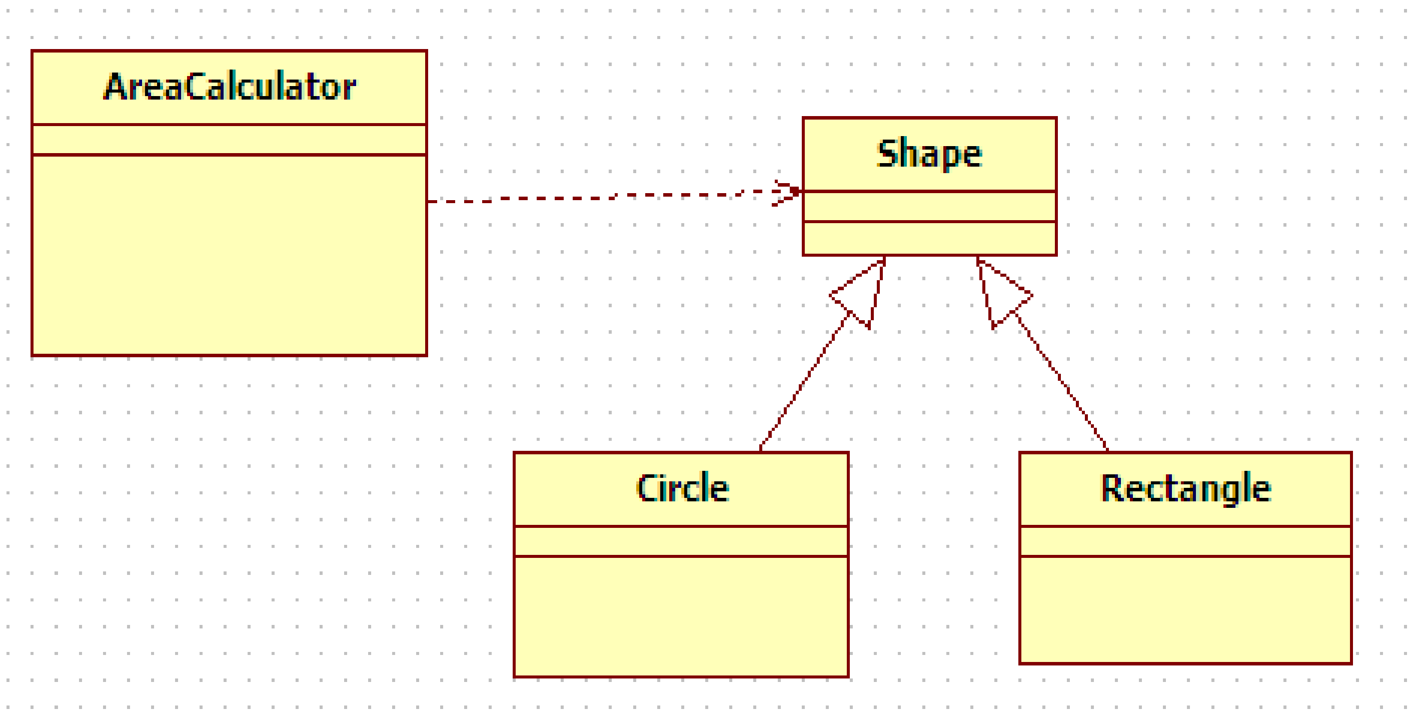
What if you introduce another shape Ellipse? What will happen to the code inside AreaCalculator?

Refactoring with OCP

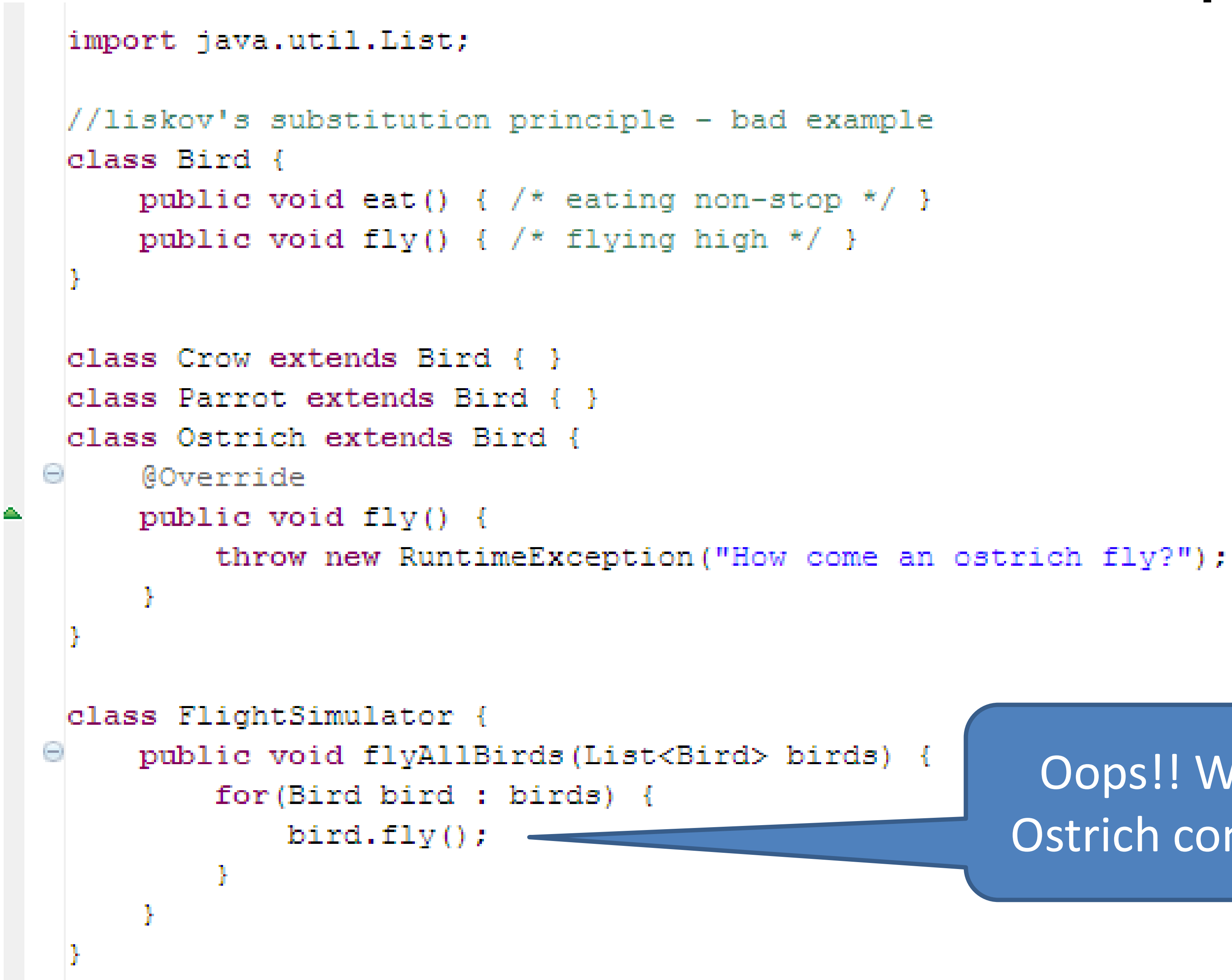


```java
//open close principle - good example
abstract class Shape {
    public abstract double getArea();
}
class Rectangle extends Shape {
    private double width;
    private double height;
    public void setWidth(double width) { this.width = width; }
    public double getWidth() { return width; }
    public void setHeight(double height) { this.height = height; }
    public double getHeight() { return height; }
    public double getArea() { return this.getWidth() * this.getHeight(); } ;
}
class Circle extends Shape {
    private double radius;
    public void setRadius(double radius) { this.radius = radius; }
    public double getRadius() { return radius; }
    public double getArea() { return this.getRadius() * this.getRadius() * Math.PI; }
}
class AreaCalculator {
    public double getTotalAreaOfShapes(Shape[] shapes) {
        double area = 0;
        for (int i = 0; i < shapes.length; i++) {
            area += shapes[i].getArea();
        }
        return area;
    }
}
```

OCP Revised Design



Liskov's Substitution Principle

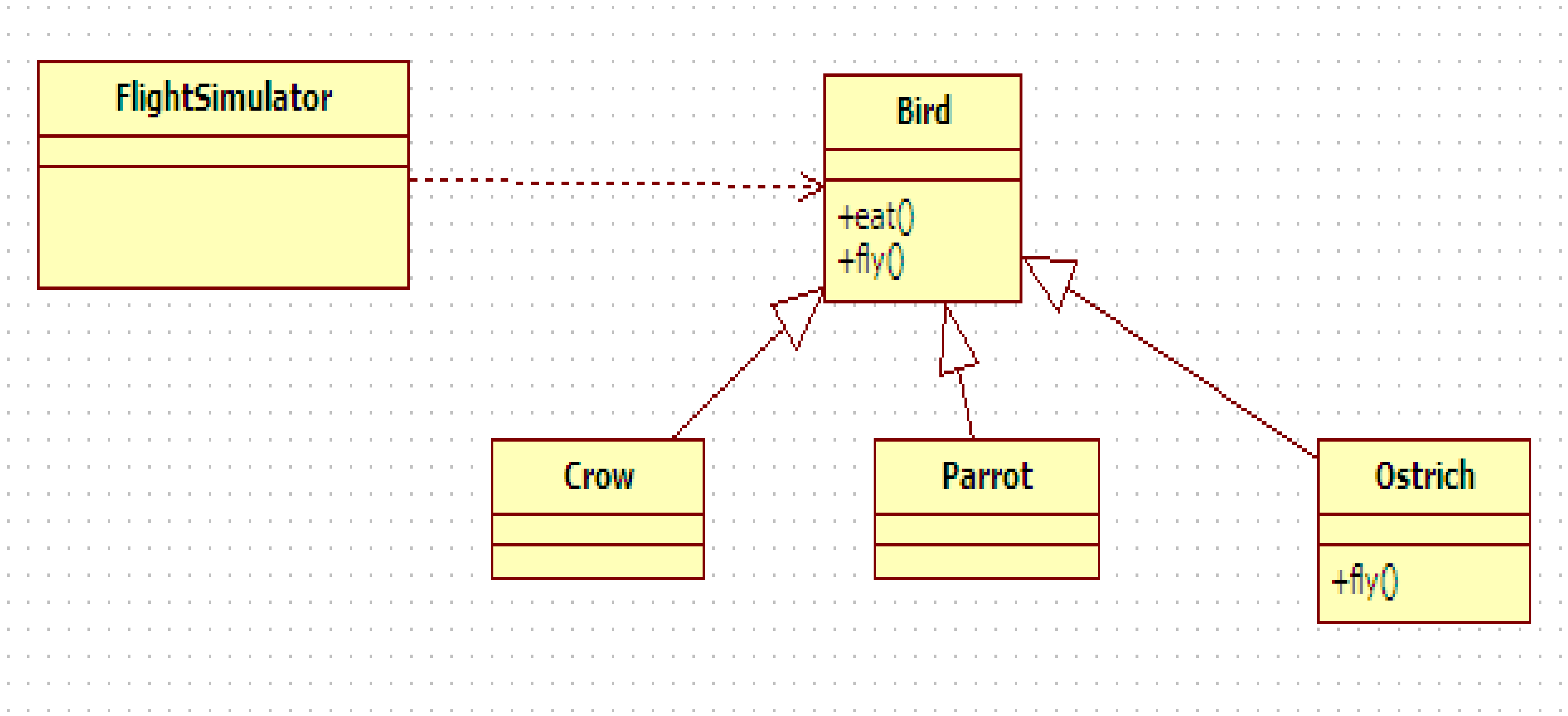


```java
import java.util.List;

//liskov's substitution principle - bad example
class Bird {
    public void eat() { /* eating non-stop */ }
    public void fly() { /* flying high */ }
}

class Crow extends Bird { }
class Parrot extends Bird { }
class Ostrich extends Bird {
    @Override
    public void fly() {
        throw new RuntimeException("How come an ostrich fly?");
    }
}

class FlightSimulator {
    public void flyAllBirds(List<Bird> birds) {
        for(Bird bird : birds) {
            bird.fly();
        }
    }
}
```

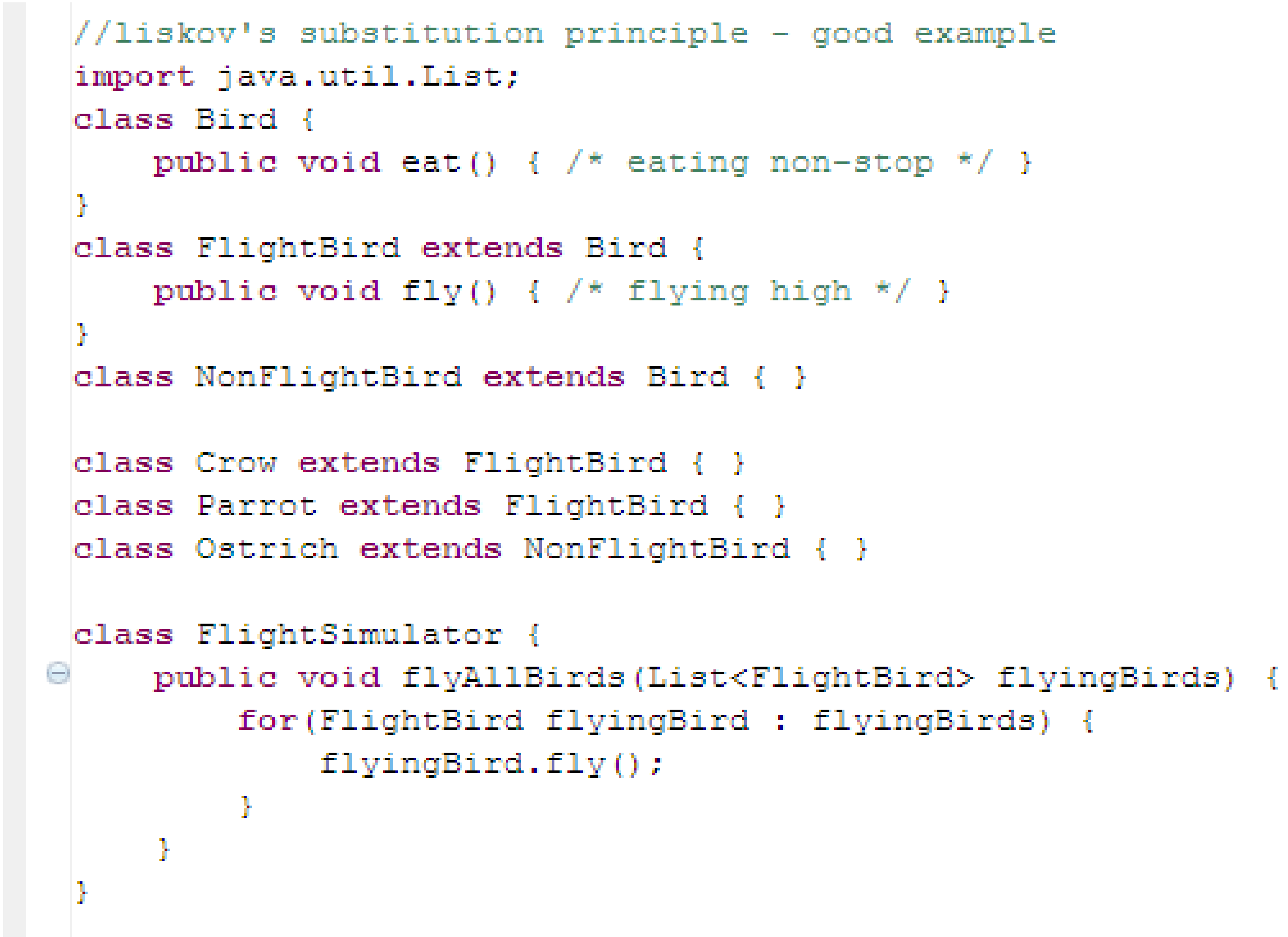
Oops!! What if an Ostrich comes here?

LSP Contd …

- Derived types must be **completely substitutable** for their base types.

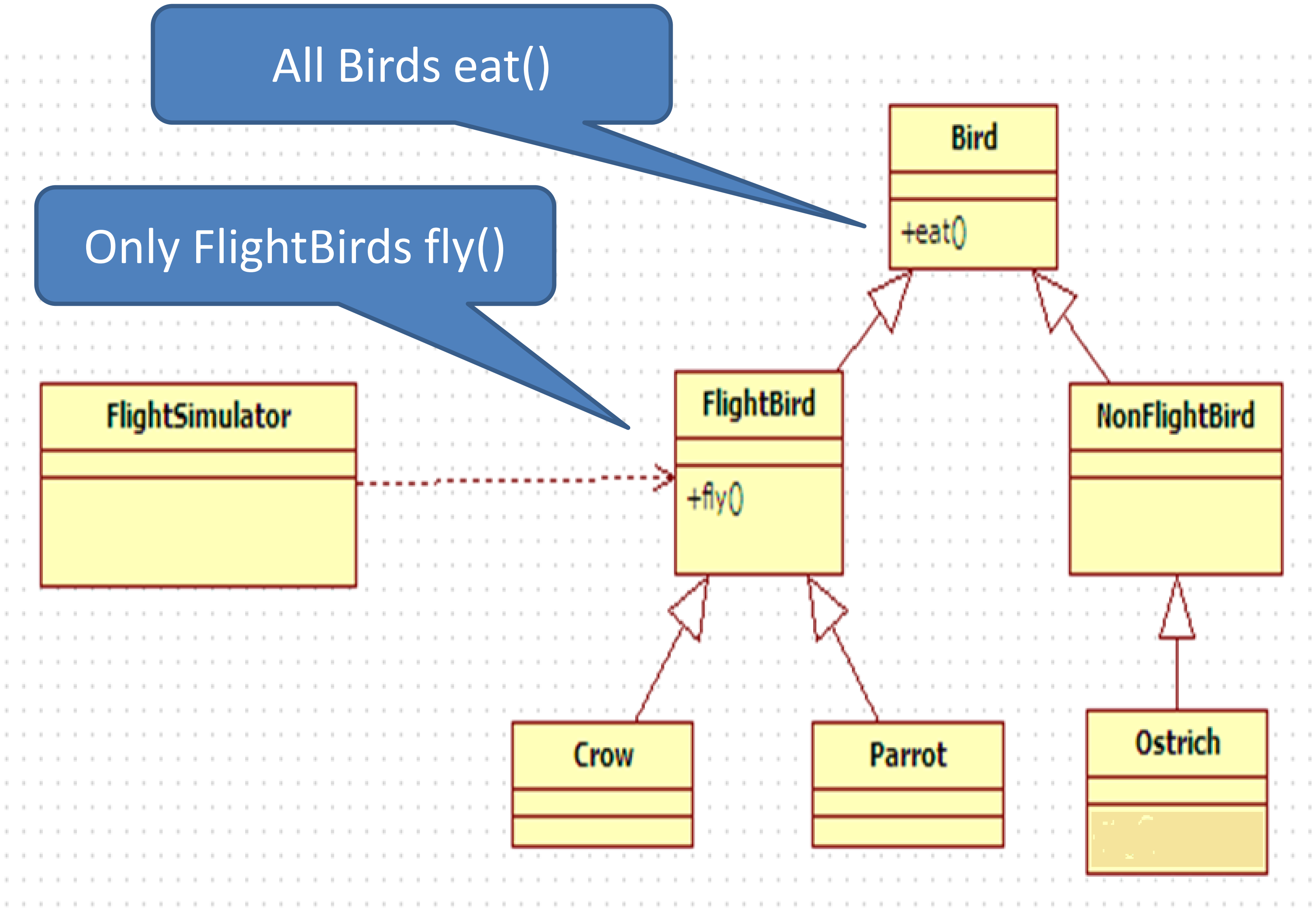


Refactoring with LSP

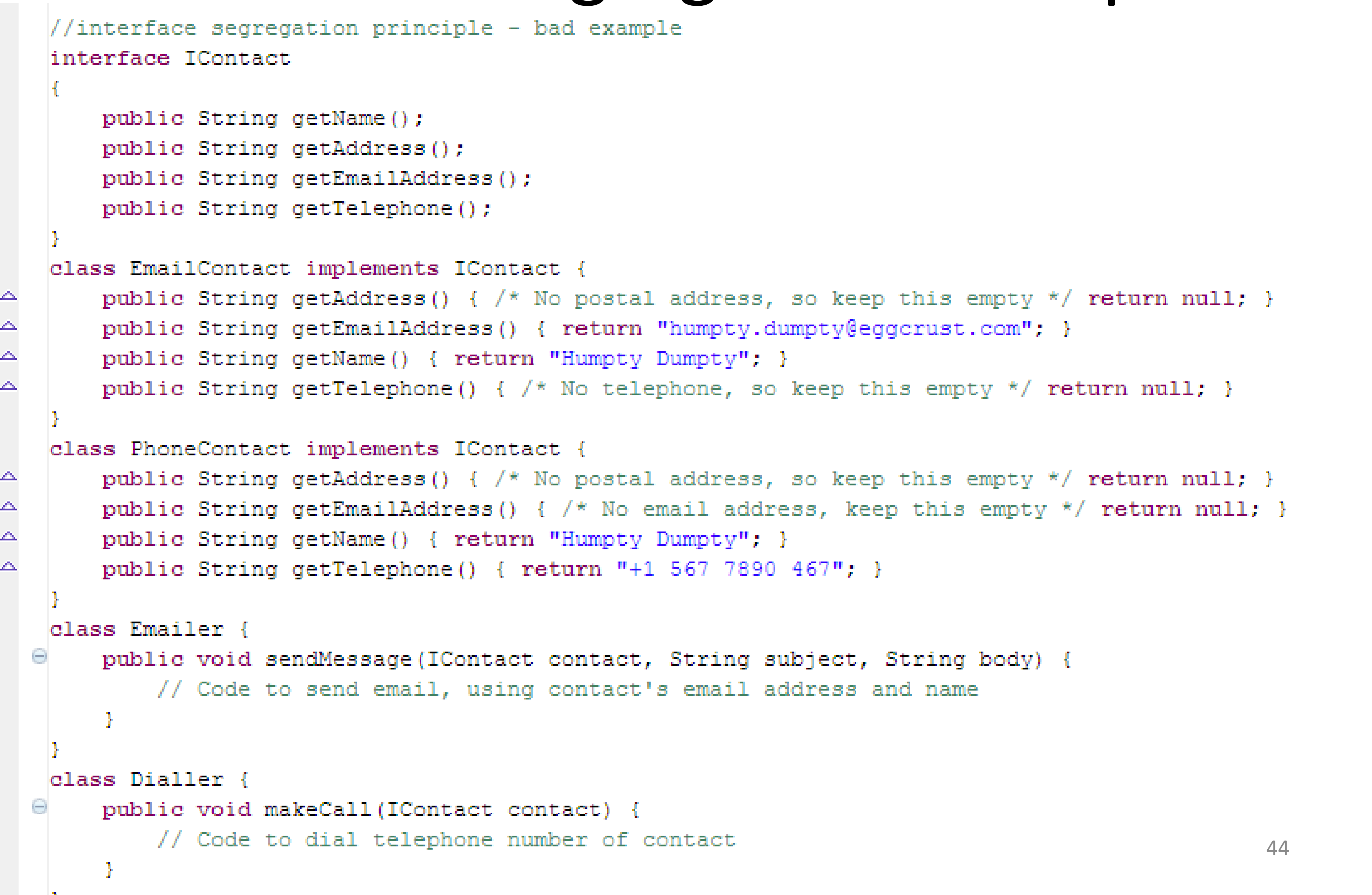


```
//liskov's substitution principle - good example
import java.util.List;
class Bird {
    public void eat() { /* eating non-stop */ }
}
class FlightBird extends Bird {
    public void fly() { /* flying high */ }
}
class NonFlightBird extends Bird { }

class Crow extends FlightBird { }
class Parrot extends FlightBird { }
class Ostrich extends NonFlightBird { }

class FlightSimulator {
    public void flyAllBirds(List<FlightBird> flyingBirds) {
        for(FlightBird flyingBird : flyingBirds) {
            flyingBird.fly();
        }
    }
}
```

LSP Revised Design



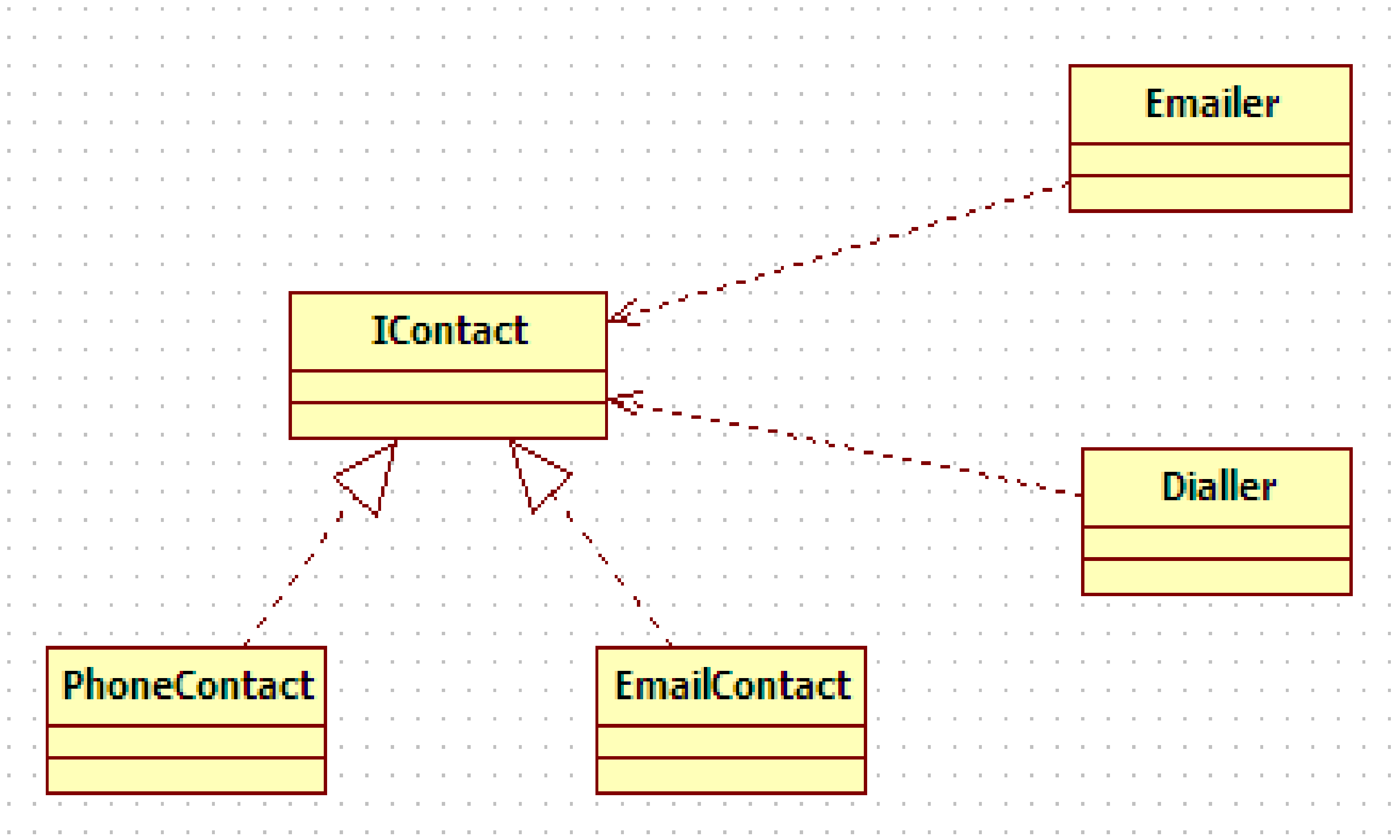
Interface Segregation Principle



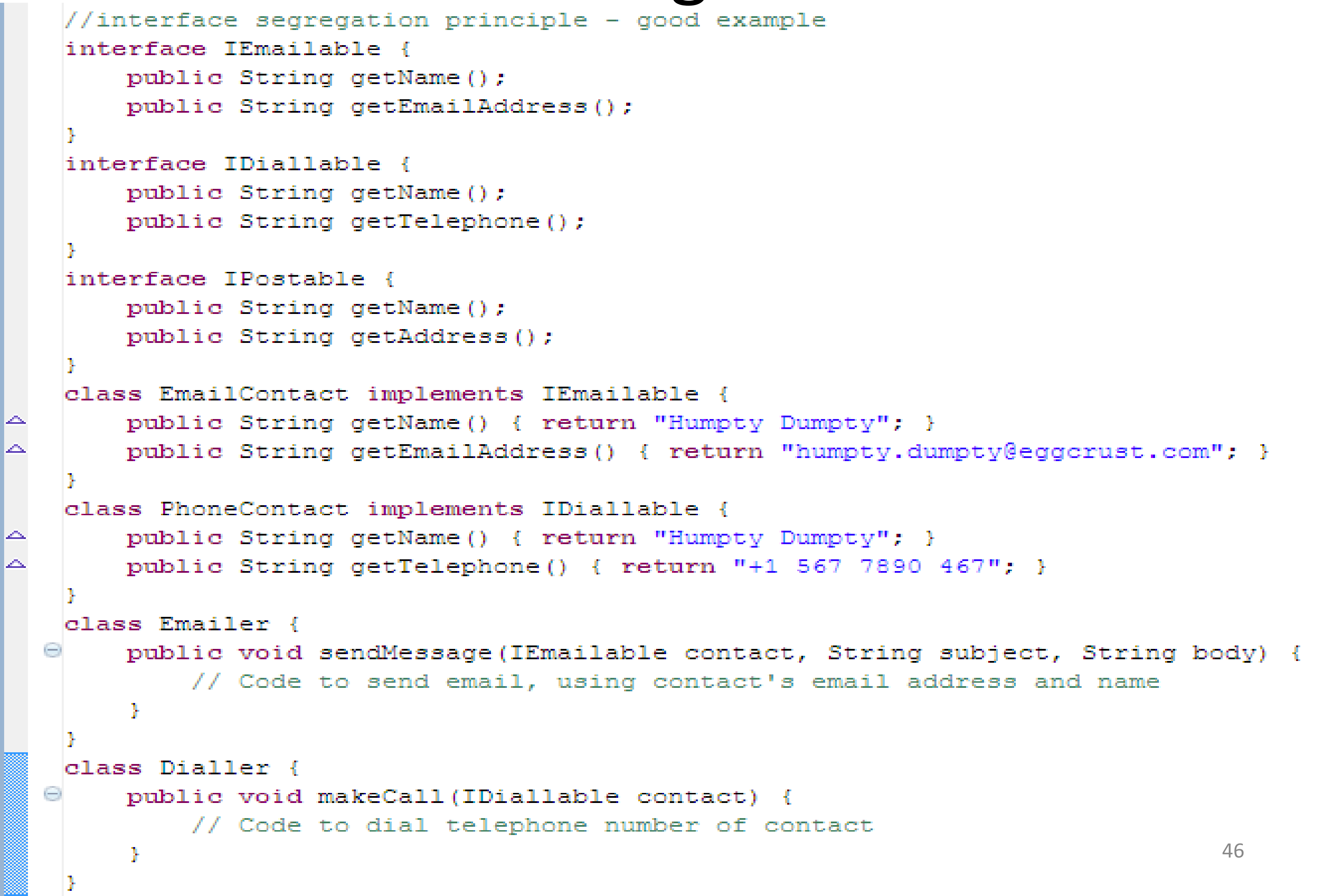
```
//interface segregation principle - bad example
interface IContact
{
    public String getName();
    public String getAddress();
    public String getEmailAddress();
    public String getTelephone();
}
class EmailContact implements IContact {
    public String getAddress() { /* No postal address, so keep this empty */ return null; }
    public String getEmailAddress() { return "humpty.dumpty@eggcrust.com"; }
    public String getName() { return "Humpty Dumpty"; }
    public String getTelephone() { /* No telephone, so keep this empty */ return null; }
}
class PhoneContact implements IContact {
    public String getAddress() { /* No postal address, so keep this empty */ return null; }
    public String getEmailAddress() { /* No email address, keep this empty */ return null; }
    public String getName() { return "Humpty Dumpty"; }
    public String getTelephone() { return "+1 567 7890 467"; }
}
class Emailer {
    public void sendMessage(IContact contact, String subject, String body) {
        // Code to send email, using contact's email address and name
    }
}
class Dialler {
    public void makeCall(IContact contact) {
        // Code to dial telephone number of contact
    }
```

ISP Contd …

- Clients should **NOT** be forced to depend upon interfaces that they **DON'T** use.

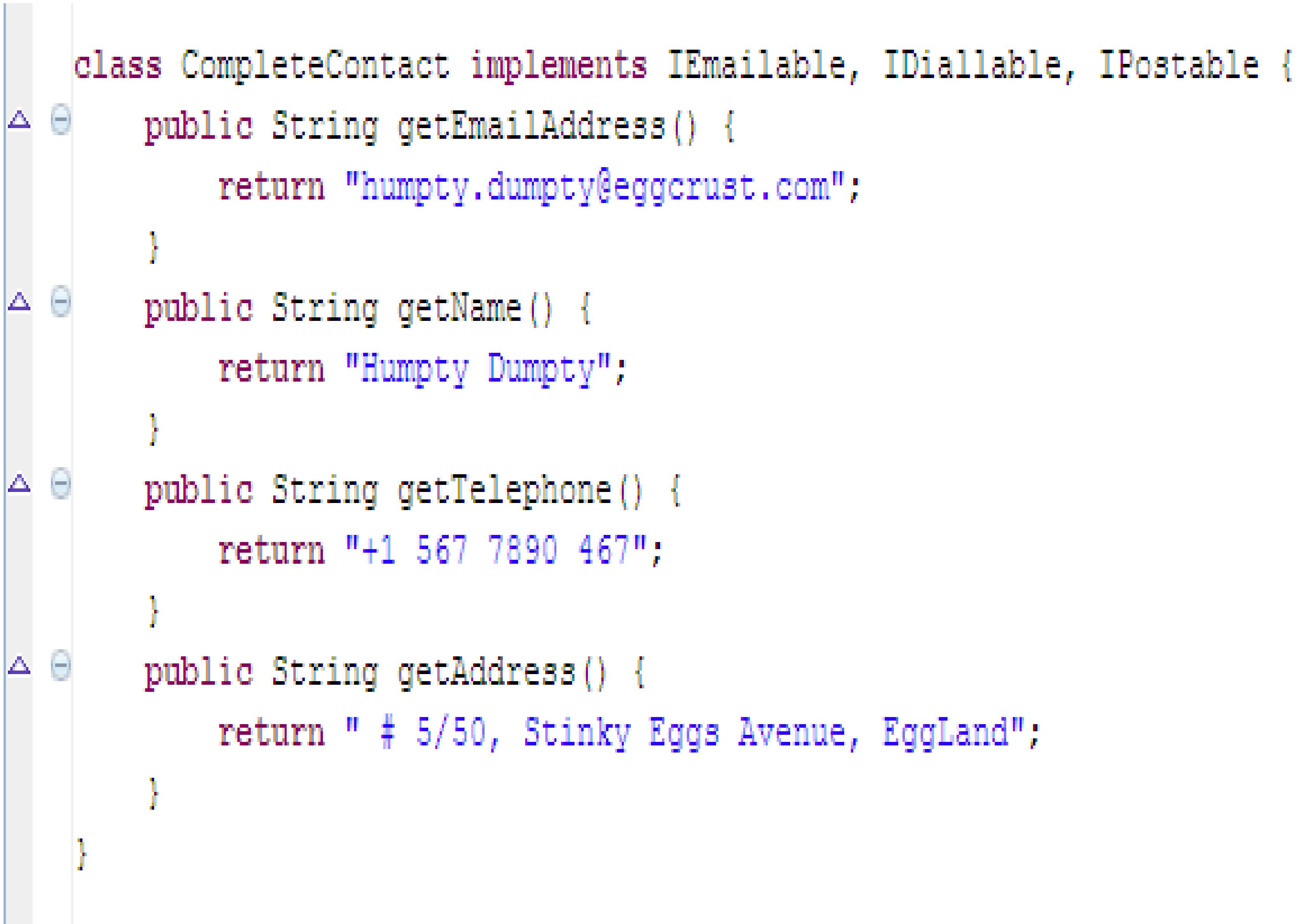


Refactoring with ISP



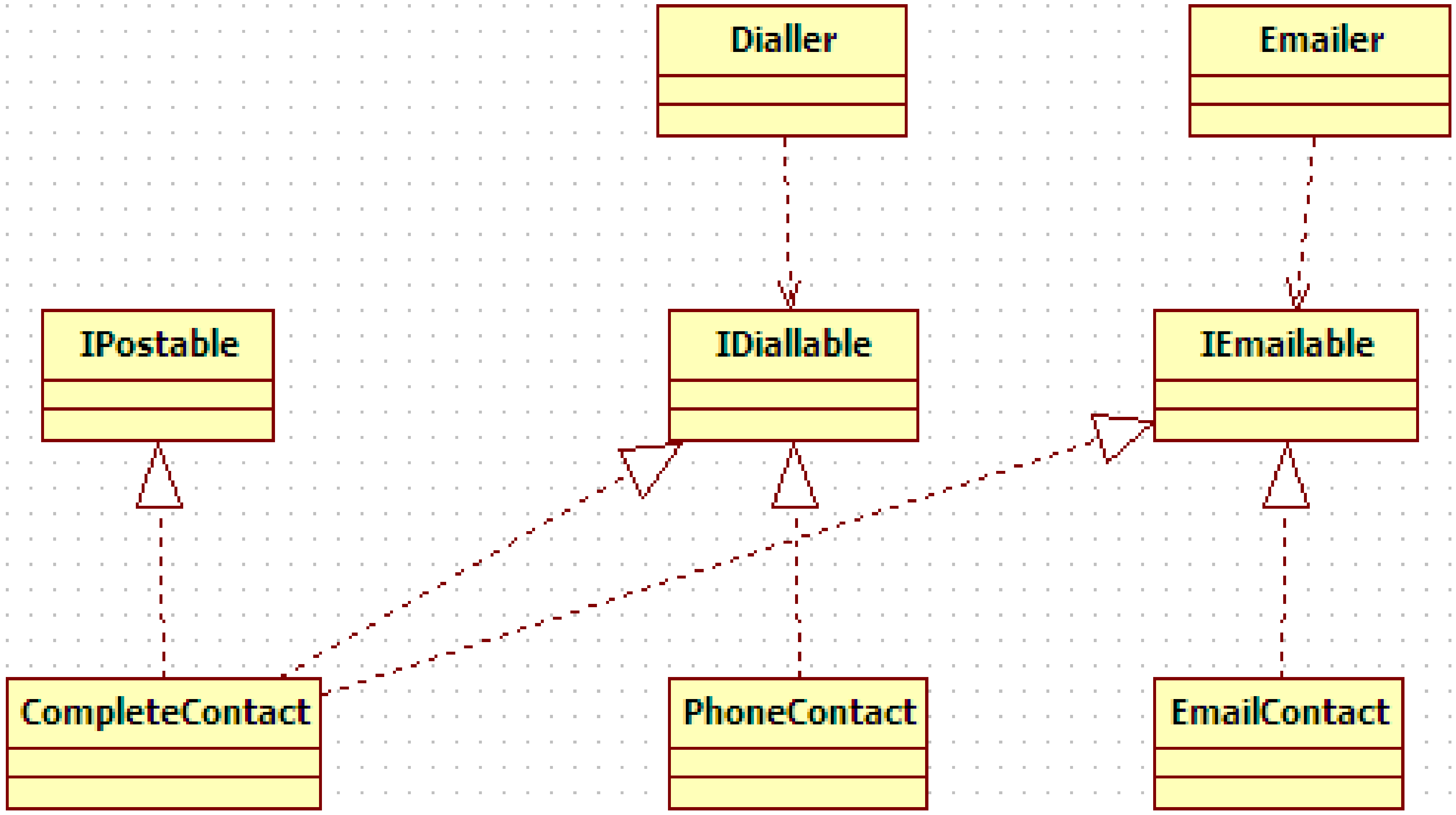
```java
//interface segregation principle - good example
interface IEmailable {
    public String getName();
    public String getEmailAddress();
}
interface IDiallable {
    public String getName();
    public String getTelephone();
}
interface IPostable {
    public String getName();
    public String getAddress();
}
class EmailContact implements IEmailable {
    public String getName() { return "Humpty Dumpty"; }
    public String getEmailAddress() { return "humpty.dumpty@eggcrust.com"; }
}
class PhoneContact implements IDiallable {
    public String getName() { return "Humpty Dumpty"; }
    public String getTelephone() { return "+1 567 7890 467"; }
}
class Emailer {
    public void sendMessage(IEmailable contact, String subject, String body) {
        // Code to send email, using contact's email address and name
    }
}
class Dialler {
    public void makeCall(IDiallable contact) {
        // Code to dial telephone number of contact
    }
}
```

Refactoring with ISP

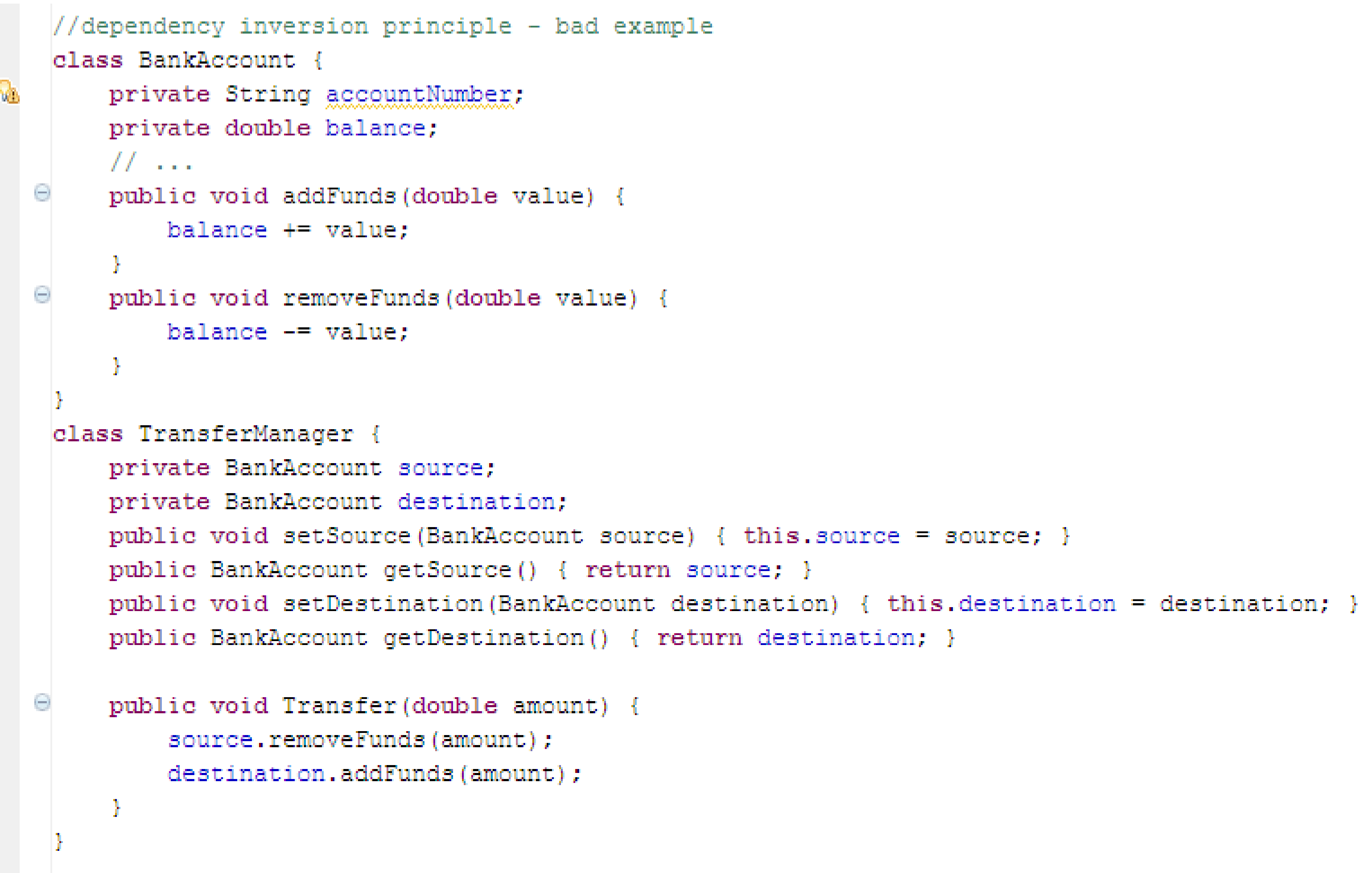


```
class CompleteContact implements IEmailable, IDiallable, IPostable {
    public String getEmailAddress() {
        return "humpty.dumpty@eggcrust.com";
    }
    public String getName() {
        return "Humpty Dumpty";
    }
    public String getTelephone() {
        return "+1 567 7890 467";
    }
    public String getAddress() {
        return " # 5/50, Stinky Eggs Avenue, EggLand";
    }
}
```

ISP Revised Design



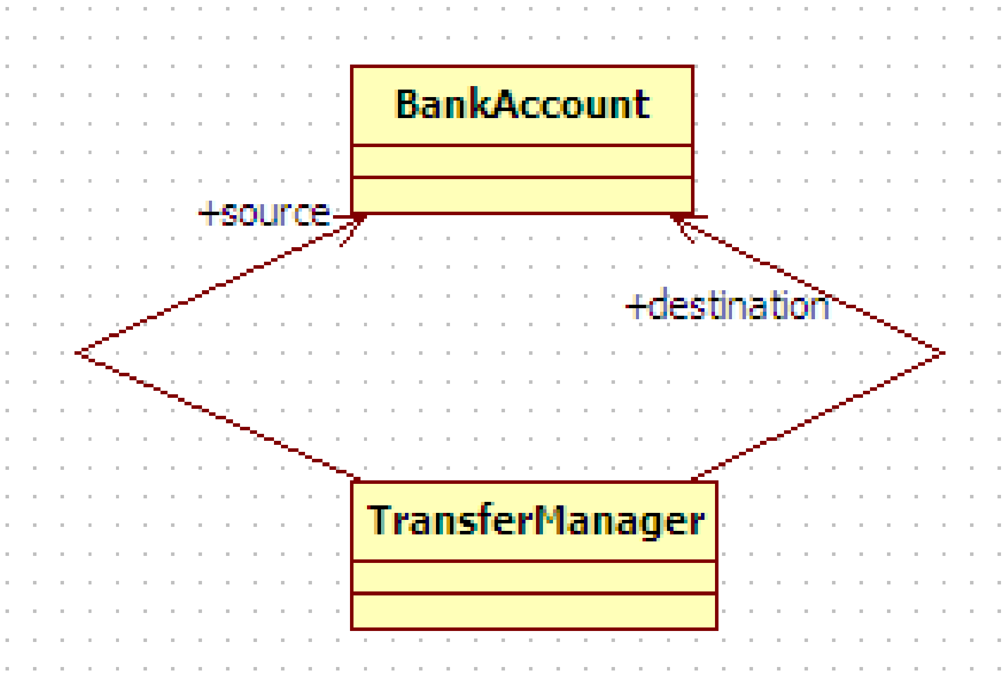
Dependency Inversion Principle



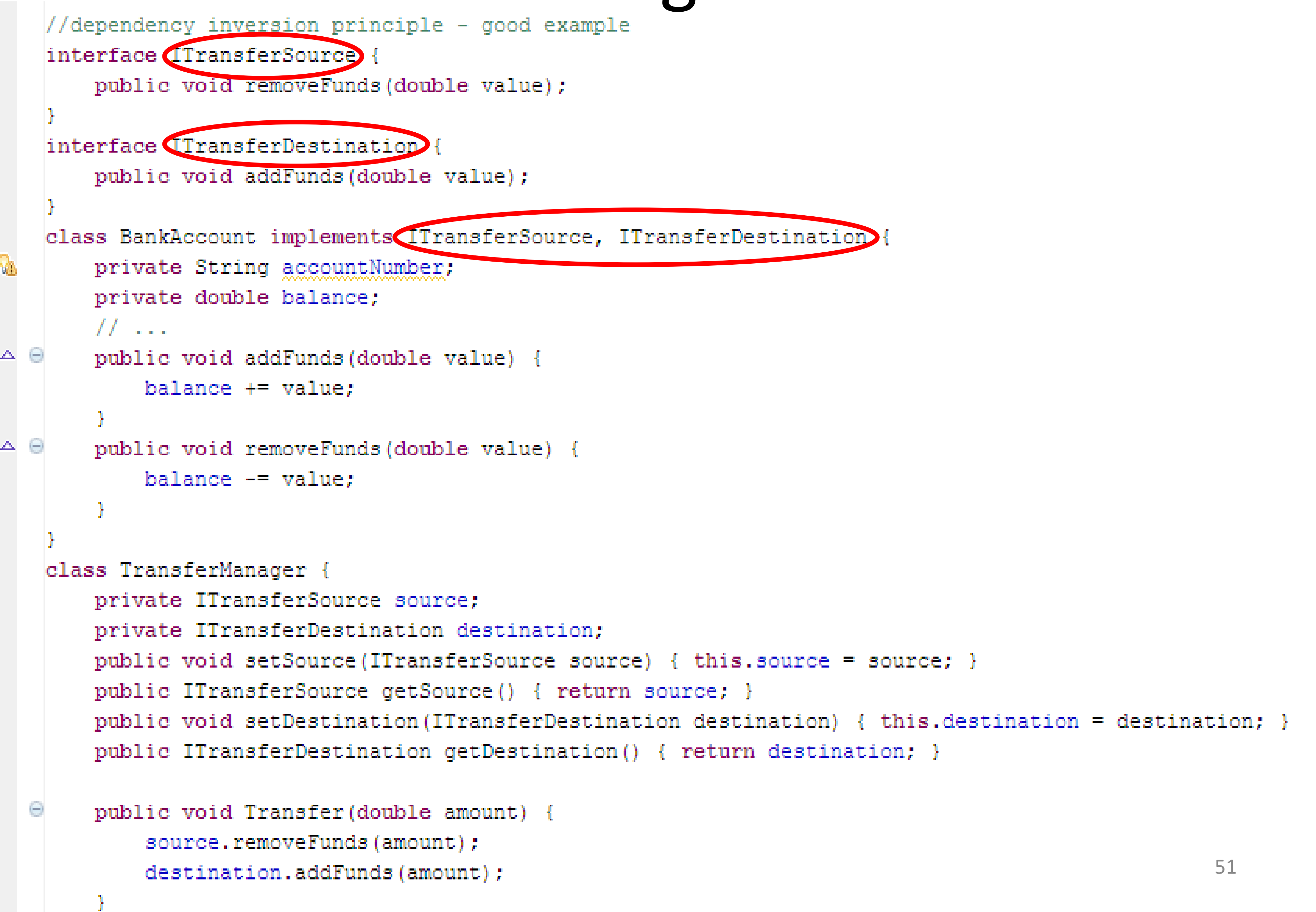
```java
//dependency inversion principle - bad example
class BankAccount {
    private String accountNumber;
    private double balance;
    // ...
    public void addFunds(double value) {
        balance += value;
    }
    public void removeFunds(double value) {
        balance -= value;
    }
}
class TransferManager {
    private BankAccount source;
    private BankAccount destination;
    public void setSource(BankAccount source) { this.source = source; }
    public BankAccount getSource() { return source; }
    public void setDestination(BankAccount destination) { this.destination = destination; }
    public BankAccount getDestination() { return destination; }

    public void Transfer(double amount) {
        source.removeFunds(amount);
        destination.addFunds(amount);
    }
}
```

DIP Contd …

- High-level modules should not depend on low-level modules. Both should depend on **abstractions**.
- **Abstractions** should not depend on **details**. **Details** should depend on **abstractions**.



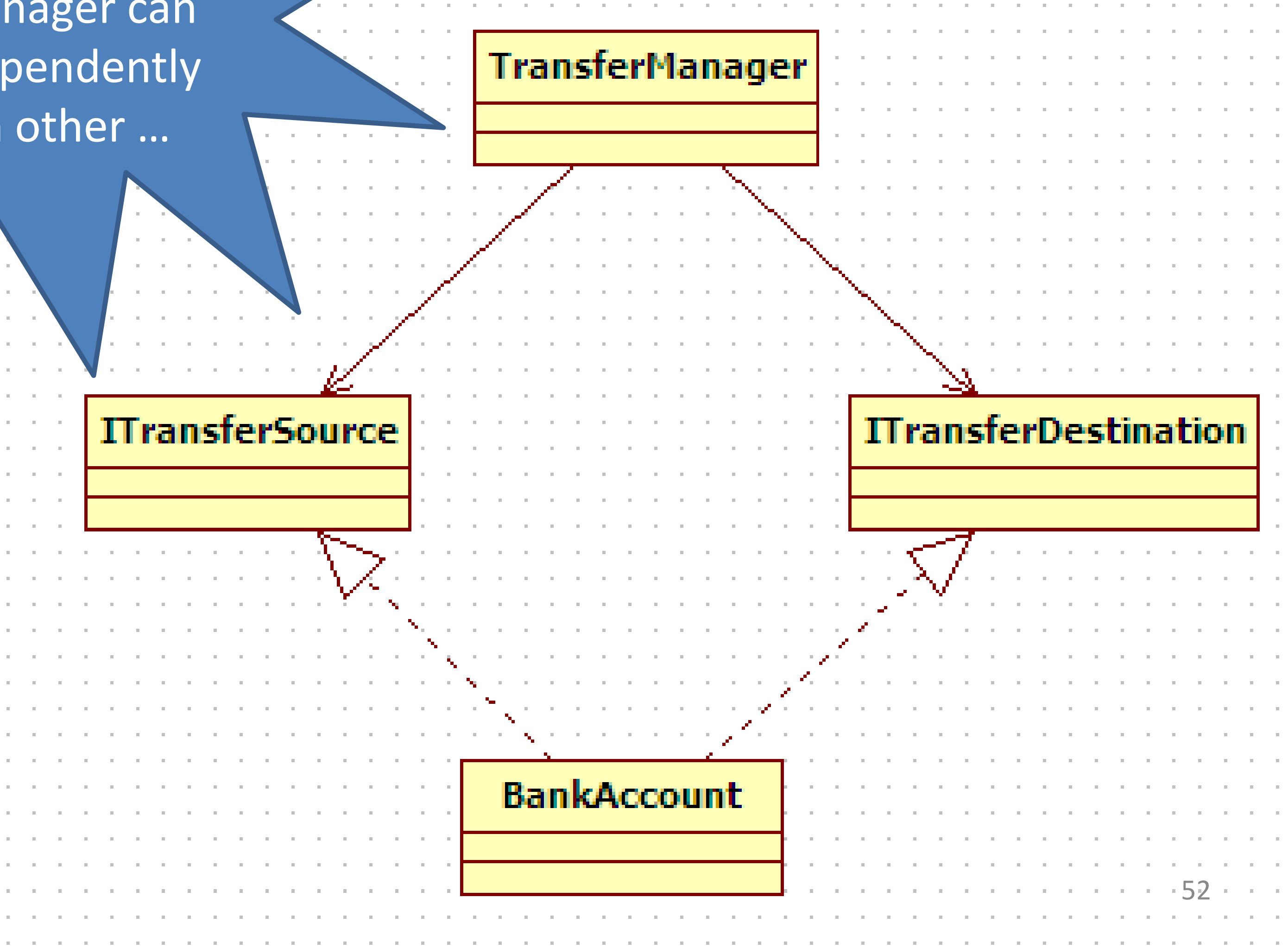
Refactoring with DIP



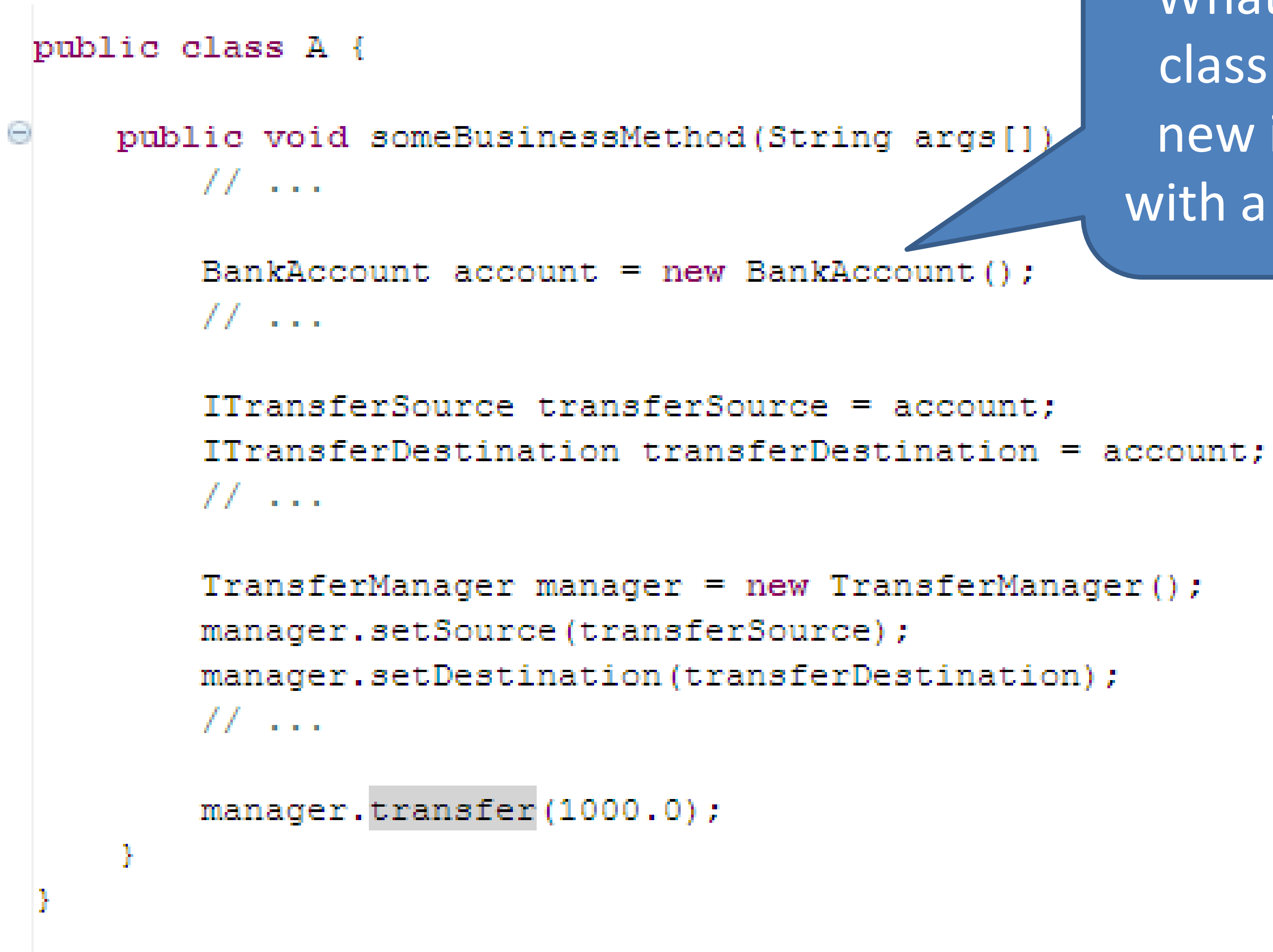
```
//dependency inversion principle - good example
interface ITransferSource {
    public void removeFunds(double value);
}
interface ITransferDestination {
    public void addFunds(double value);
}
class BankAccount implements ITransferSource, ITransferDestination {
    private String accountNumber;
    private double balance;
    // ...
    public void addFunds(double value) {
        balance += value;
    }
    public void removeFunds(double value) {
        balance -= value;
    }
}
class TransferManager {
    private ITransferSource source;
    private ITransferDestination destination;
    public void setSource(ITransferSource source) { this.source = source; }
    public ITransferSource getSource() { return source; }
    public void setDestination(ITransferDestination destination) { this.destination = destination; }
    public ITransferDestination getDestination() { return destination; }

    public void Transfer(double amount) {
        source.removeFunds(amount);
        destination.addFunds(amount);
    }
```

DIP Revised Design

Thanks to DIP, BankAccount and TransferManager can evolve independently from each other …



What about Other Connections ???



```
public class A {

    public void someBusinessMethod(String args[]){
        // ...

        BankAccount account = new BankAccount();
        // ...

        ITransferSource transferSource = account;
        ITransferDestination transferDestination = account;
        // ...

        TransferManager manager = new TransferManager();
        manager.setSource(transferSource);
        manager.setDestination(transferDestination);
        // ...

        manager.transfer(1000.0);
    }
}
```

What if BankAccount class is replaced by a new implementation with a new class name?

About connections contd …

Thanks to DIP, we don’t need to change a single piece of code within TransferManager class …

But still we need to change the place which **BankAccount** is **initialized** in **someBusinessMethod()** in class A

XML vs. DIP

- Why not we specify the **connections** of these objects in an **xml file**?
- Why not write a **framework** to **initialize** and **connect** these objects by **reading the XML file**?
- Then we can straight away call **transfer()** method on an object created from **TransferManager** class

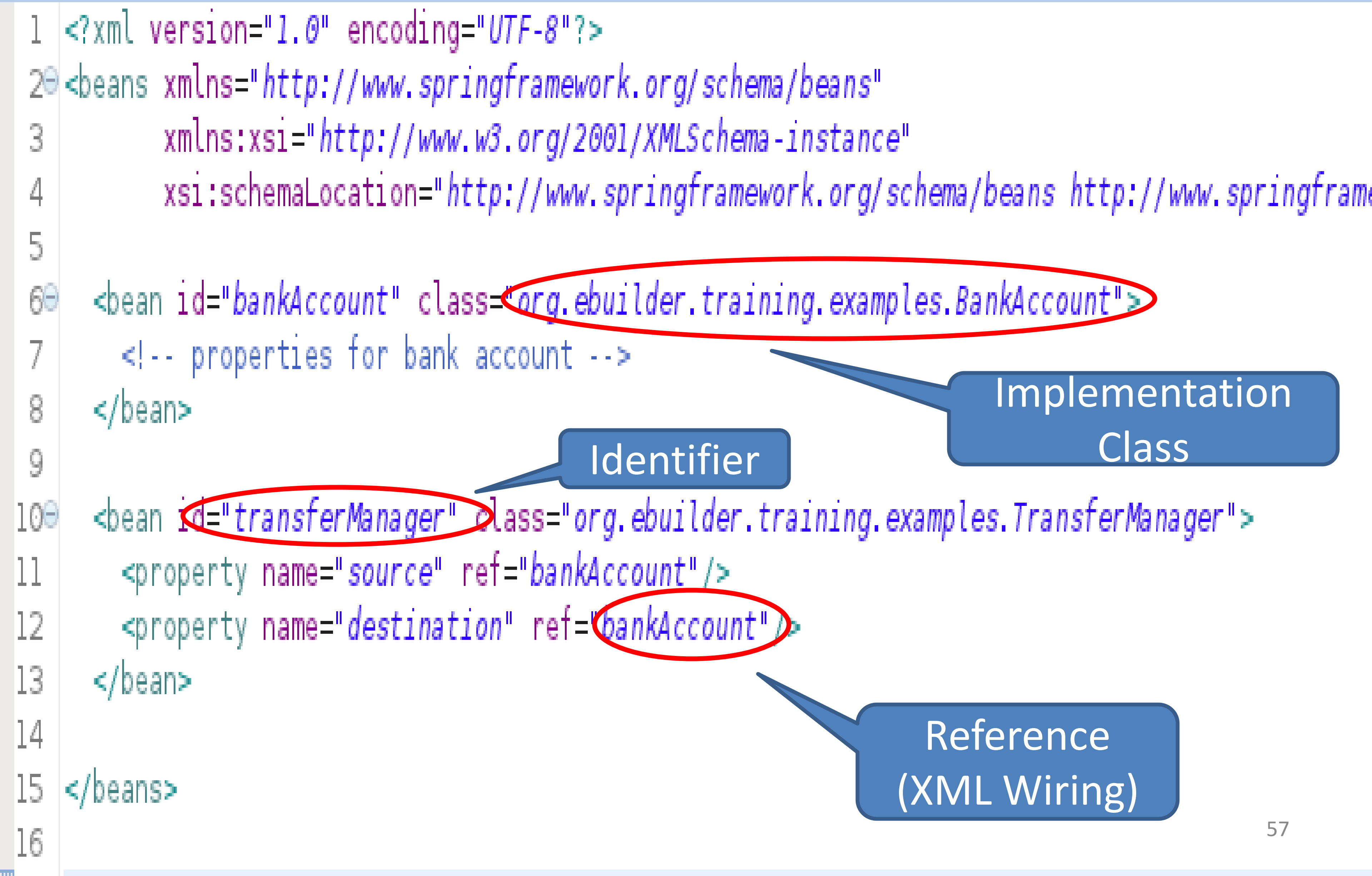
XML vs. DIP Contd …

DIP

XML Wiring

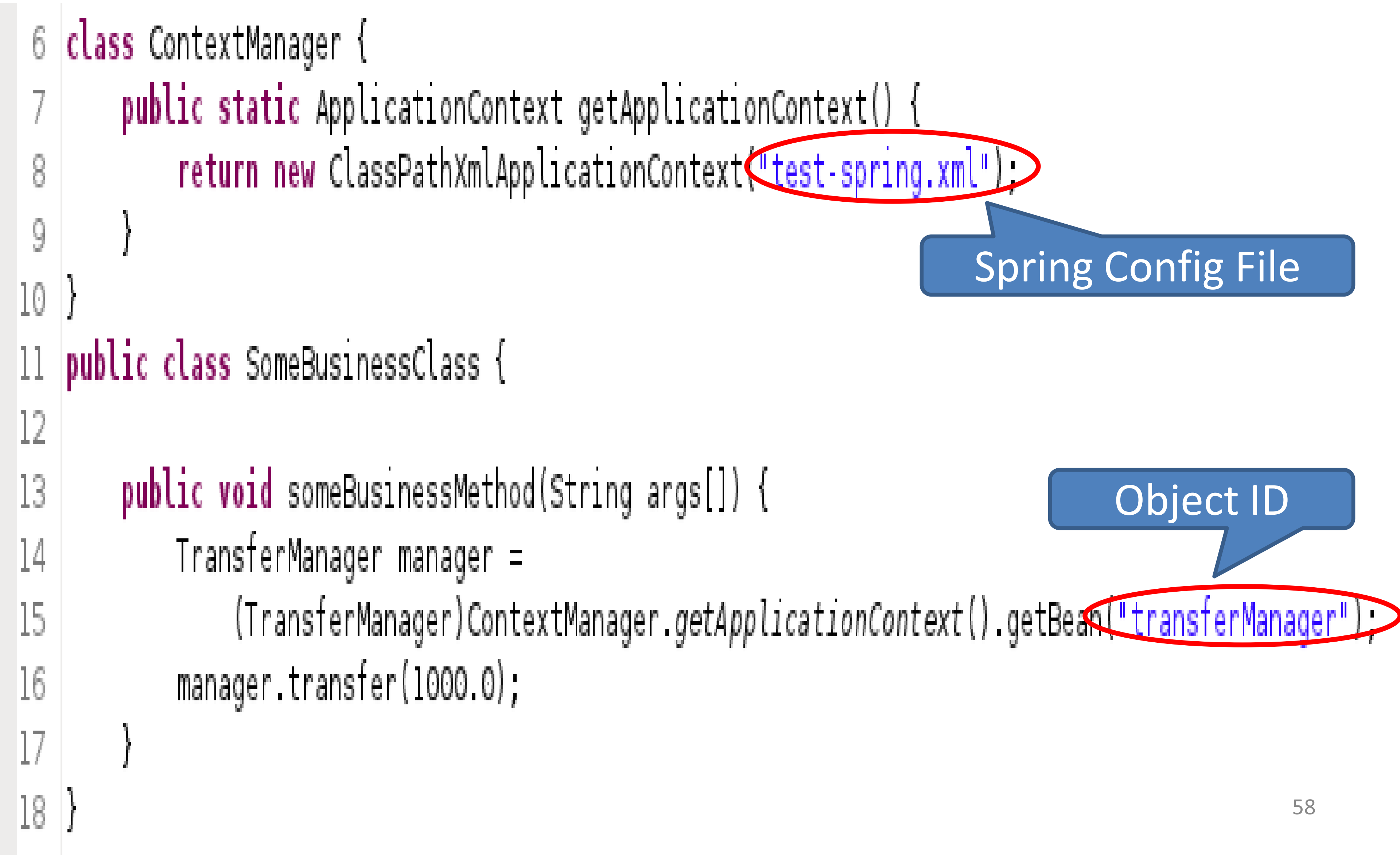
Birth of
Spring Framework

Part of Sample Spring XML File



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfram

  <bean id="bankAccount" class="org.ebuilder.training.examples.BankAccount">
    <!-- properties for bank account -->
  </bean>

  <bean id="transferManager" class="org.ebuilder.training.examples.TransferManager">
    <property name="source" ref="bankAccount"/>
    <property name="destination" ref="bankAccount"/>
  </bean>

</beans>

```

Biz Class Refactored



```java
class ContextManager {
    public static ApplicationContext getApplicationContext() {
        return new ClassPathXmlApplicationContext("test-spring.xml");
    }
}
public class SomeBusinessClass {

    public void someBusinessMethod(String args[]) {
        TransferManager manager =
            (TransferManager)ContextManager.getApplicationContext().getBean("transferManager");
        manager.transfer(1000.0);
    }
}
```

OO Design Principles

Agile/Scrum

UML

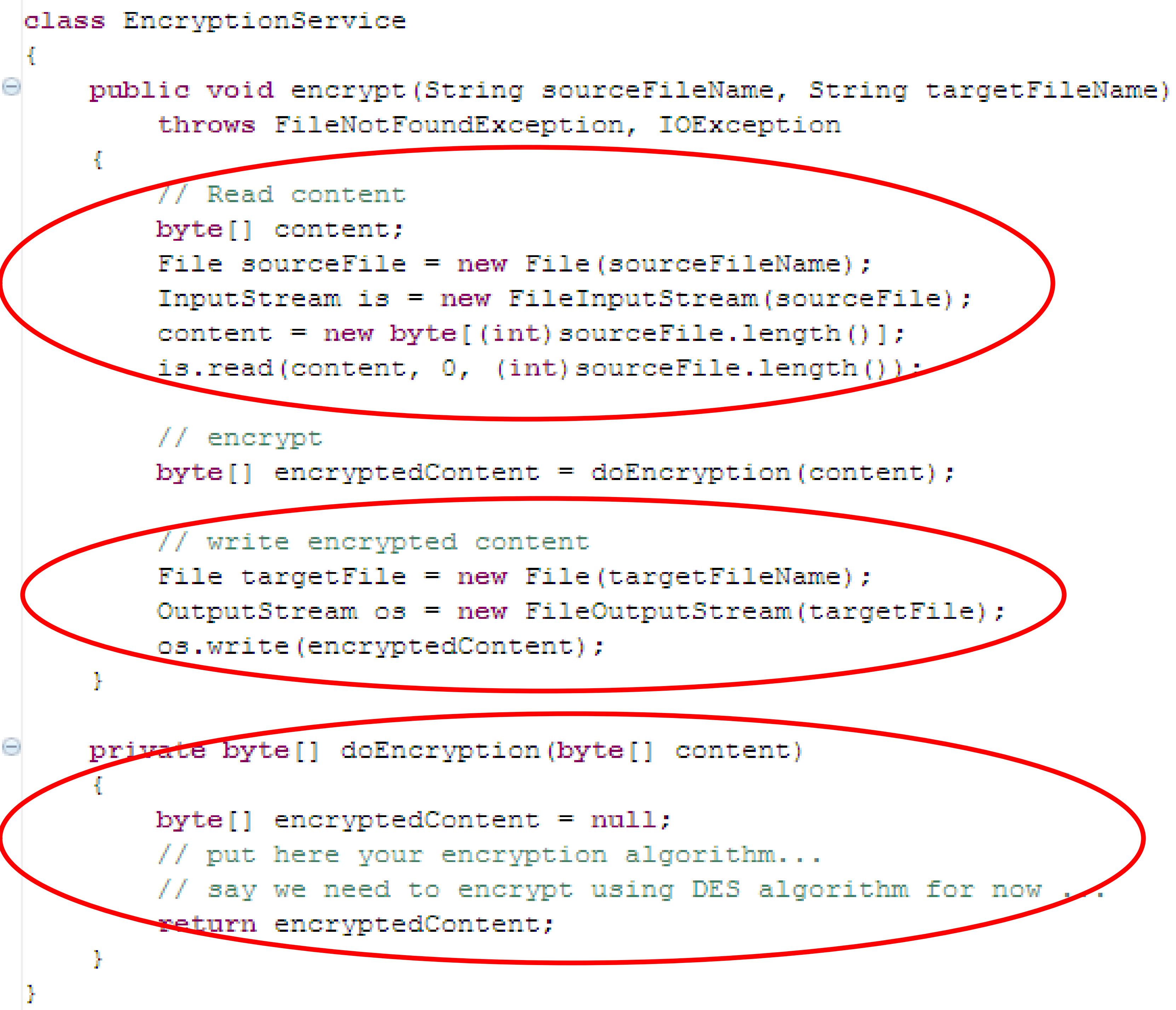
Project "eMystery"

- Sprint 1:
  - An encryption service, which is able to provide an encrypted file for a given plain file.
  - Encryption is done using DES encryption algorithm

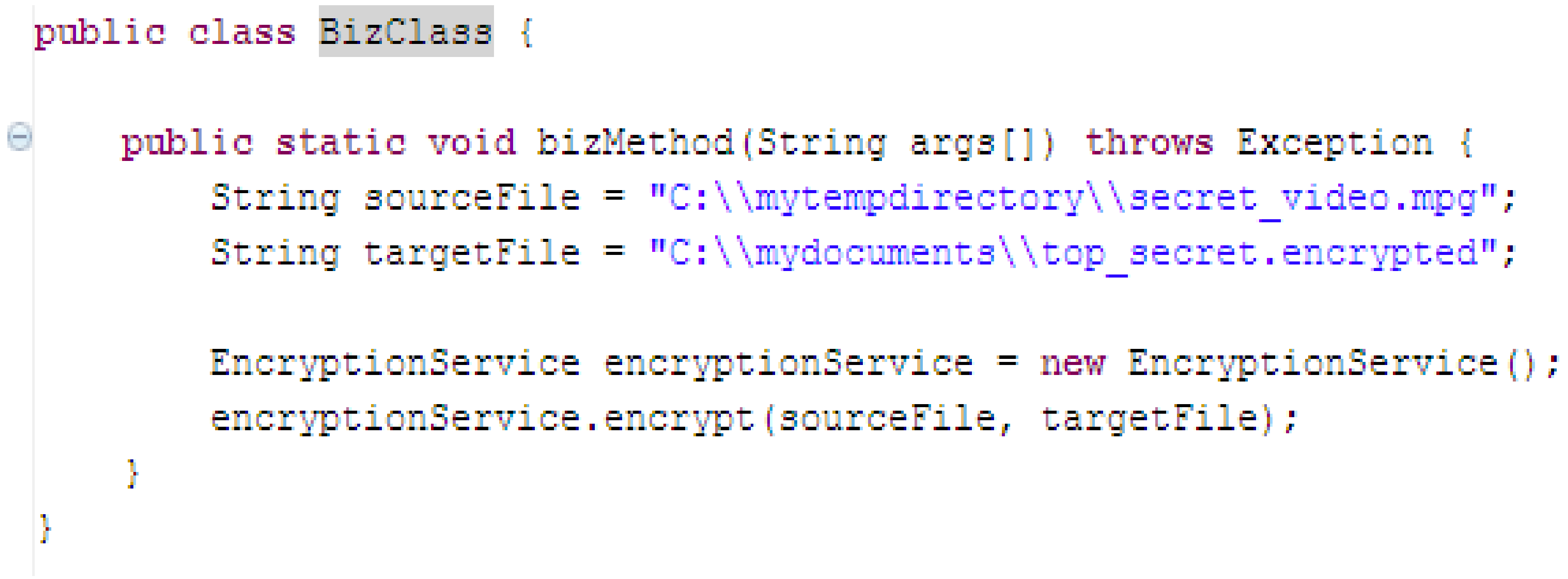
Time to get started

PM: This functionality is critical. We MUST deliver it by EoD today.

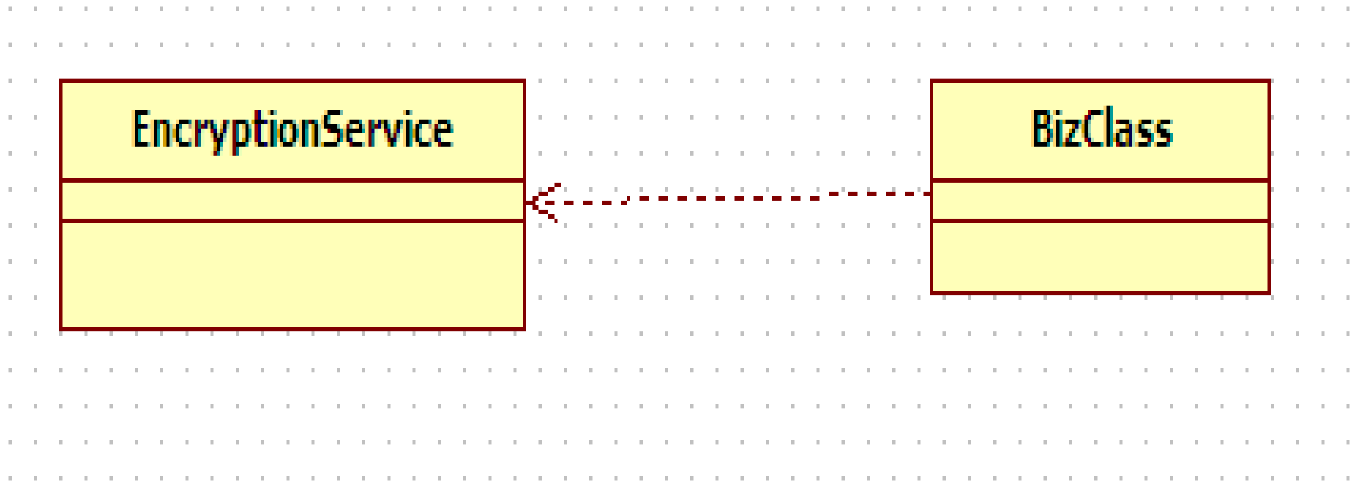
Developer: Oops!! Can’t go home today; if I apply OO Design on this. Time to code straight away.



```
class EncryptionService
{
    public void encrypt(String sourceFileName, String targetFileName)
        throws FileNotFoundException, IOException
    {
        // Read content
        byte[] content;
        File sourceFile = new File(sourceFileName);
        InputStream is = new FileInputStream(sourceFile);
        content = new byte[(int)sourceFile.length()];
        is.read(content, 0, (int)sourceFile.length());

        // encrypt
        byte[] encryptedContent = doEncryption(content);

        // write encrypted content
        File targetFile = new File(targetFileName);
        OutputStream os = new FileOutputStream(targetFile);
        os.write(encryptedContent);
    }

    private byte[] doEncryption(byte[] content)
    {
        byte[] encryptedContent = null;
        // put here your encryption algorithm...
        // say we need to encrypt using DES algorithm for now ...
        return encryptedContent;
    }
}
```



```
public class BizClass {

    public static void bizMethod(String args[]) throws Exception {
        String sourceFile = "C:\\mytempdirectory\\secret_video.mpg";
        String targetFile = "C:\\mydocuments\\top_secret.encrypted";

        EncryptionService encryptionService = new EncryptionService();
        encryptionService.encrypt(sourceFile, targetFile);
    }
}
```



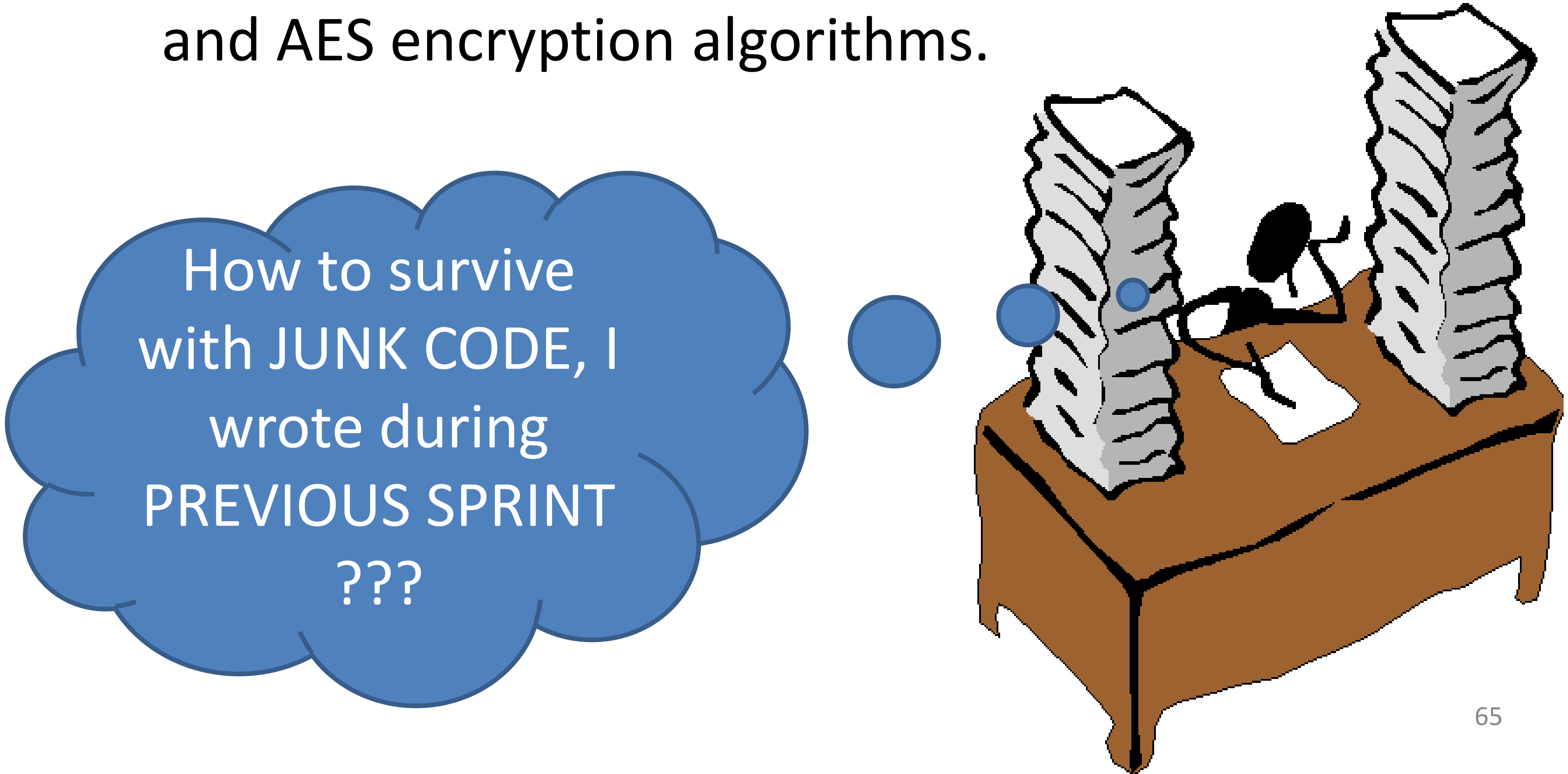
SRP is already violated

Encryption Service handles 4 responsibilities …

1. Expose encryption as a service
2. Handle data retrieval based on different sources
3. Handle encryption based on different targets
4. Handle encryption based on different algorithms

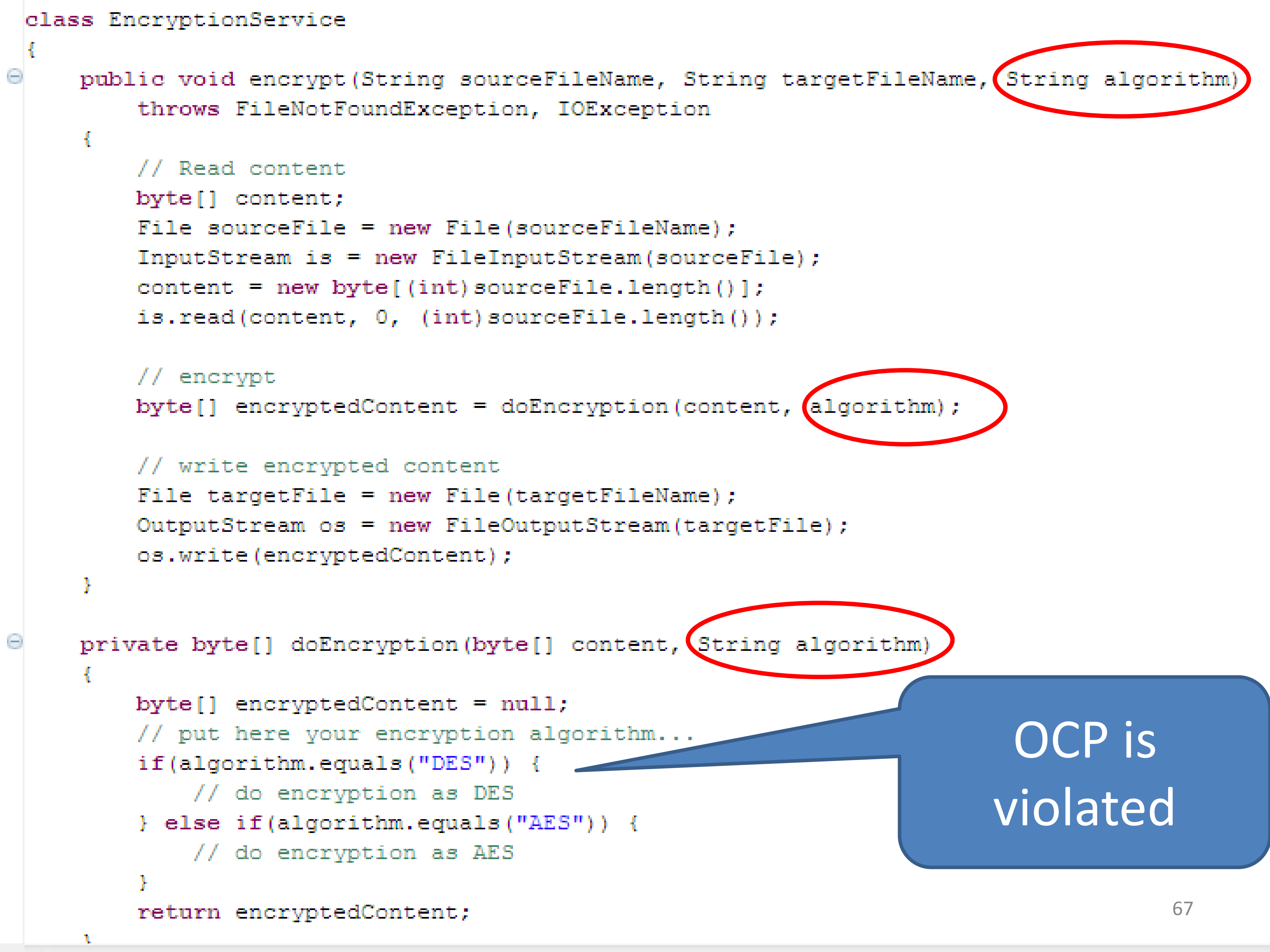
Project "eMystery"

- Sprint 2:
  - Our encryption service, must support both DES and AES encryption algorithms.


How to survive with JUNK CODE, I wrote during PREVIOUS SPRINT ???

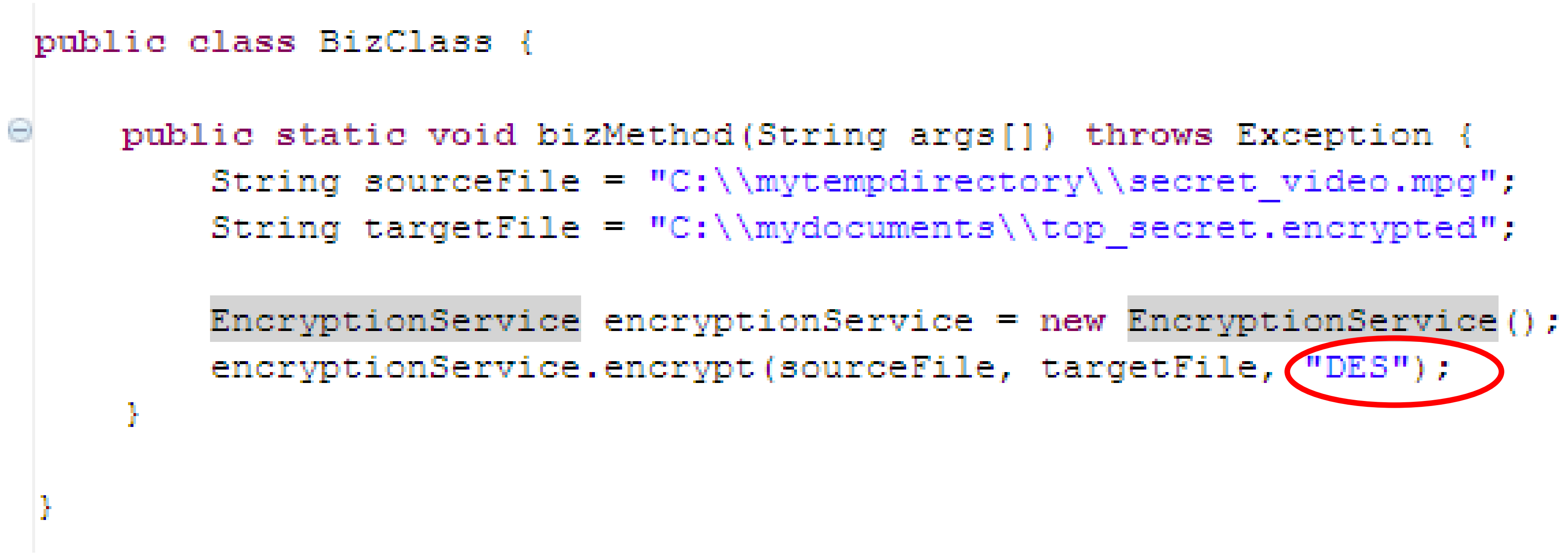
Survival Fix (“**Ana**” Fix)

Time to survive for this sprint

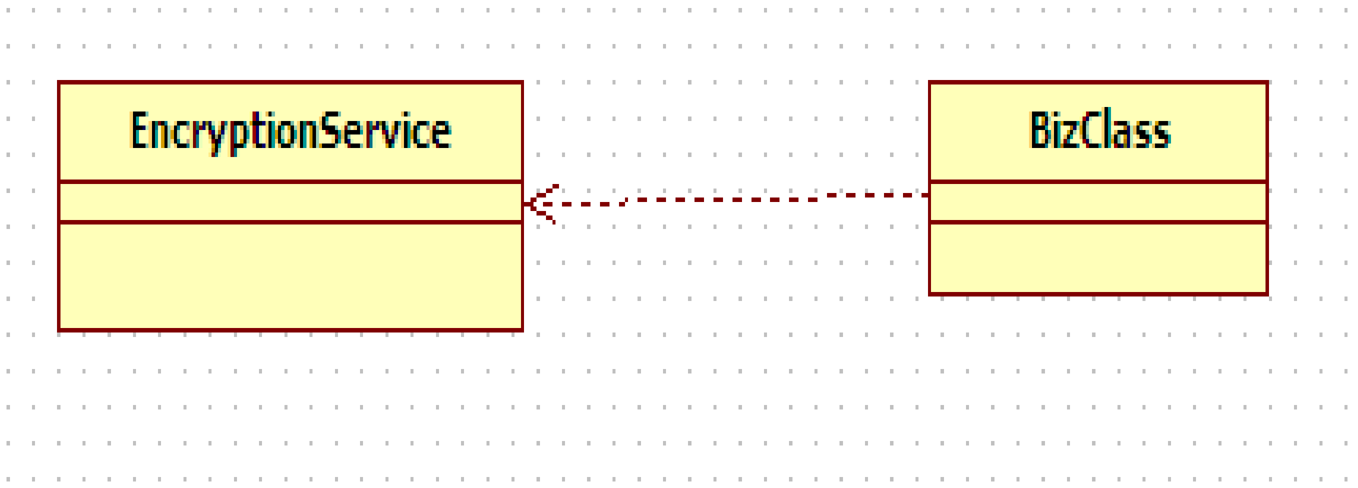


```
class EncryptionService
{
    public void encrypt(String sourceFileName, String targetFileName, String algorithm)
        throws FileNotFoundException, IOException
    {
        // Read content
        byte[] content;
        File sourceFile = new File(sourceFileName);
        InputStream is = new FileInputStream(sourceFile);
        content = new byte[(int)sourceFile.length()];
        is.read(content, 0, (int)sourceFile.length());

        // encrypt
        byte[] encryptedContent = doEncryption(content, algorithm);

        // write encrypted content
        File targetFile = new File(targetFileName);
        OutputStream os = new FileOutputStream(targetFile);
        os.write(encryptedContent);
    }

    private byte[] doEncryption(byte[] content, String algorithm)
    {
        byte[] encryptedContent = null;
        // put here your encryption algorithm...
        if(algorithm.equals("DES")) {
            // do encryption as DES
        } else if(algorithm.equals("AES")) {
            // do encryption as AES
        }
        return encryptedContent;
    }
```

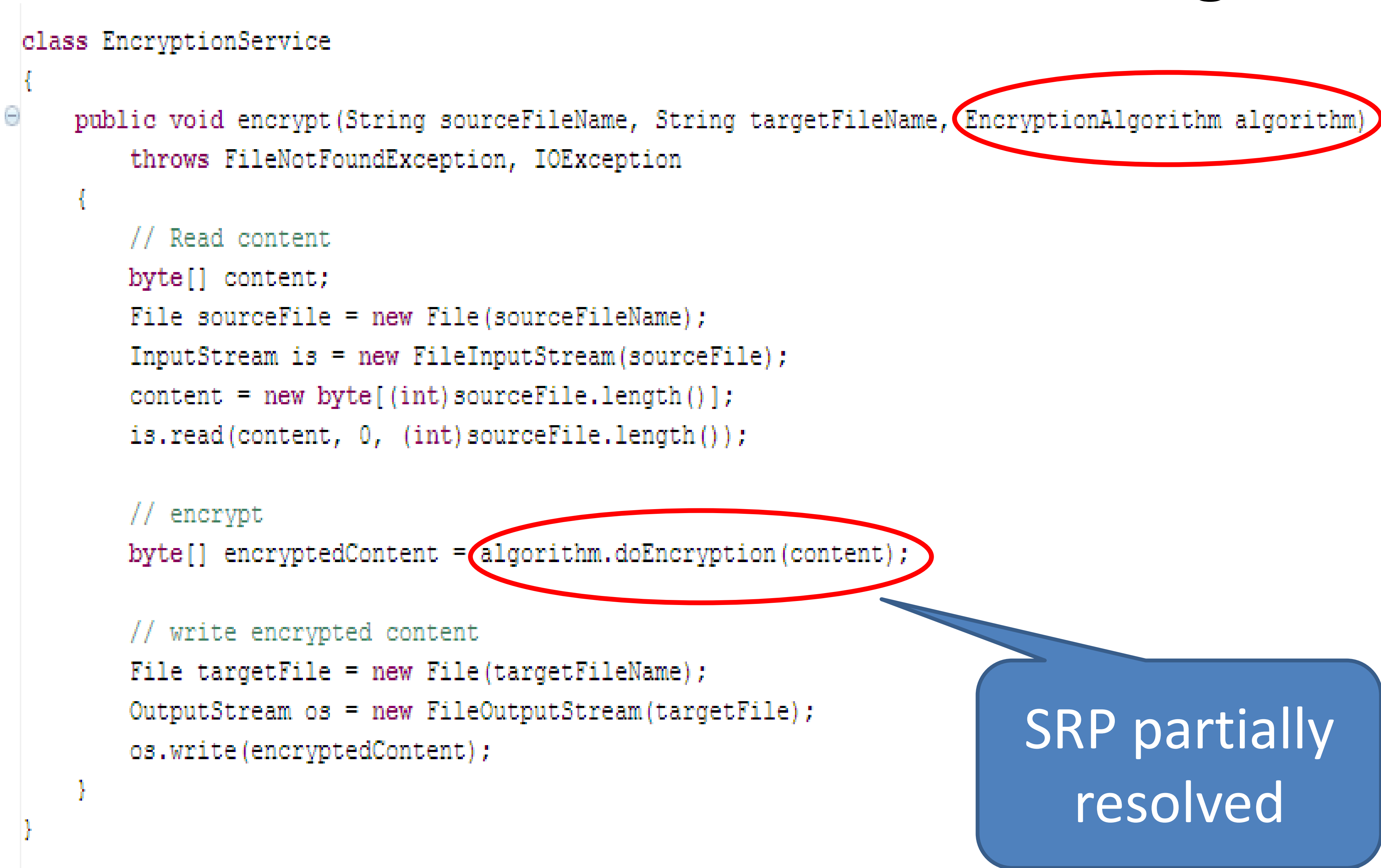
OCP is violated



```
public class BizClass {

    public static void bizMethod(String args[]) throws Exception {
        String sourceFile = "C:\\mytempdirectory\\secret_video.mpg";
        String targetFile = "C:\\mydocuments\\top_secret.encrypted";

        EncryptionService encryptionService = new EncryptionService();
        encryptionService.encrypt(sourceFile, targetFile, "DES");
    }

}
```

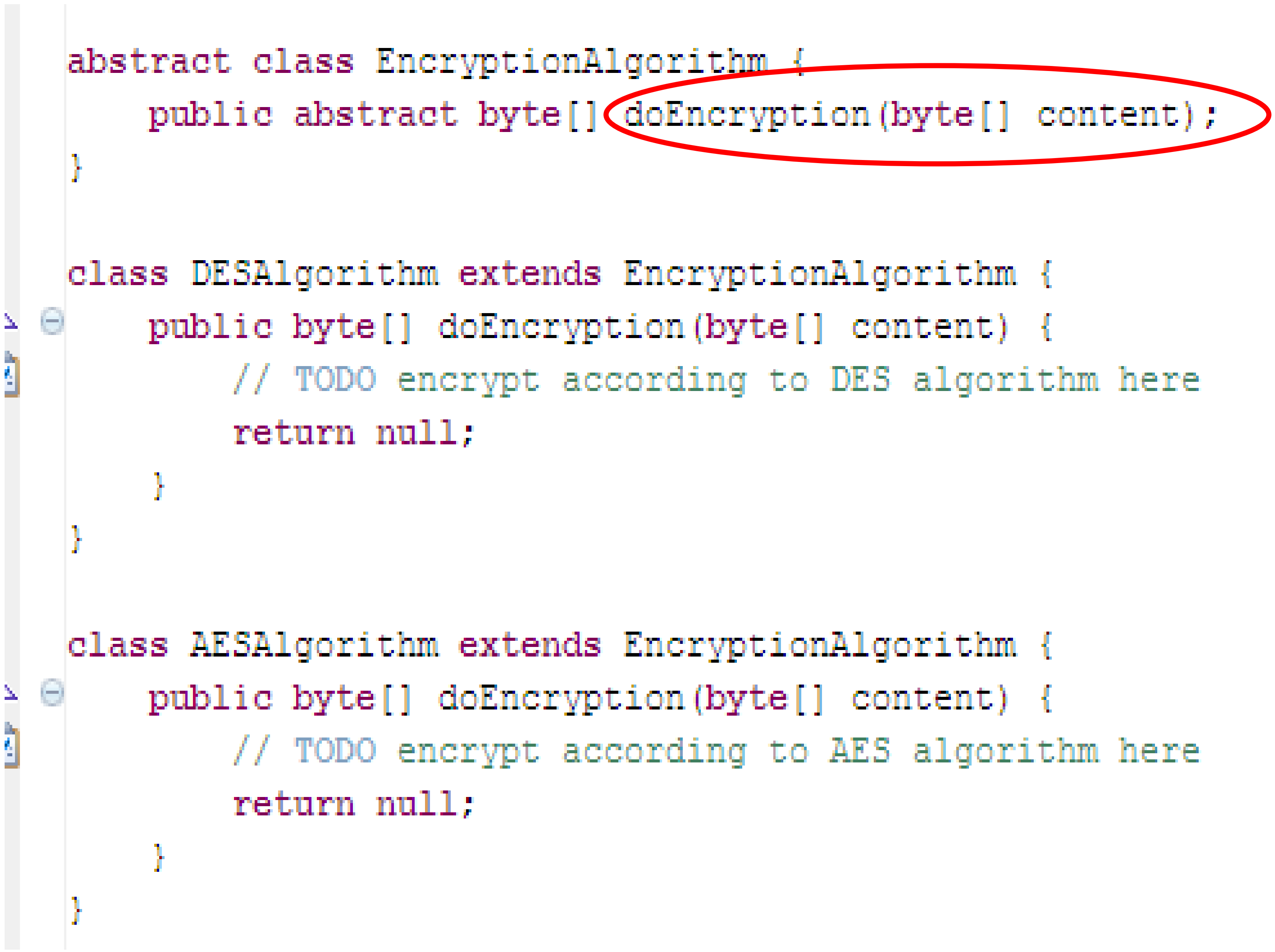


Correct Fix with OO Refactoring

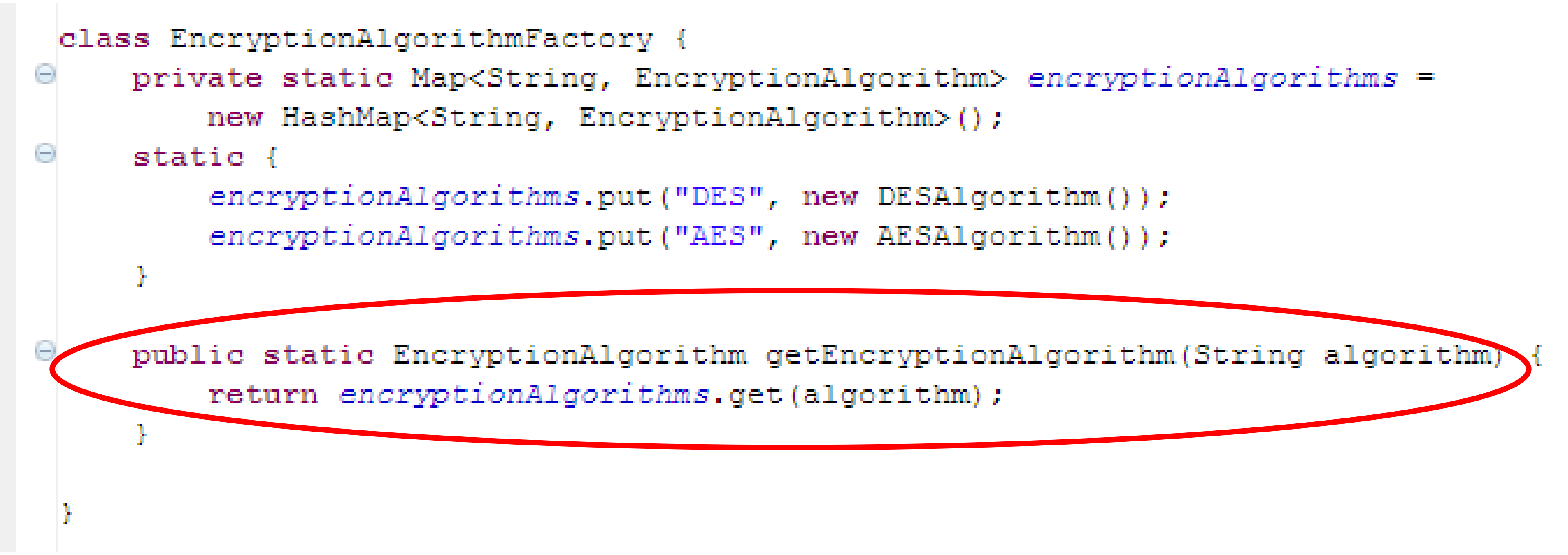


```
class EncryptionService
{
    public void encrypt(String sourceFileName, String targetFileName, EncryptionAlgorithm algorithm)
        throws FileNotFoundException, IOException
    {
        // Read content
        byte[] content;
        File sourceFile = new File(sourceFileName);
        InputStream is = new FileInputStream(sourceFile);
        content = new byte[(int)sourceFile.length()];
        is.read(content, 0, (int)sourceFile.length());

        // encrypt
        byte[] encryptedContent = algorithm.doEncryption(content);

        // write encrypted content
        File targetFile = new File(targetFileName);
        OutputStream os = new FileOutputStream(targetFile);
        os.write(encryptedContent);
    }
}
```

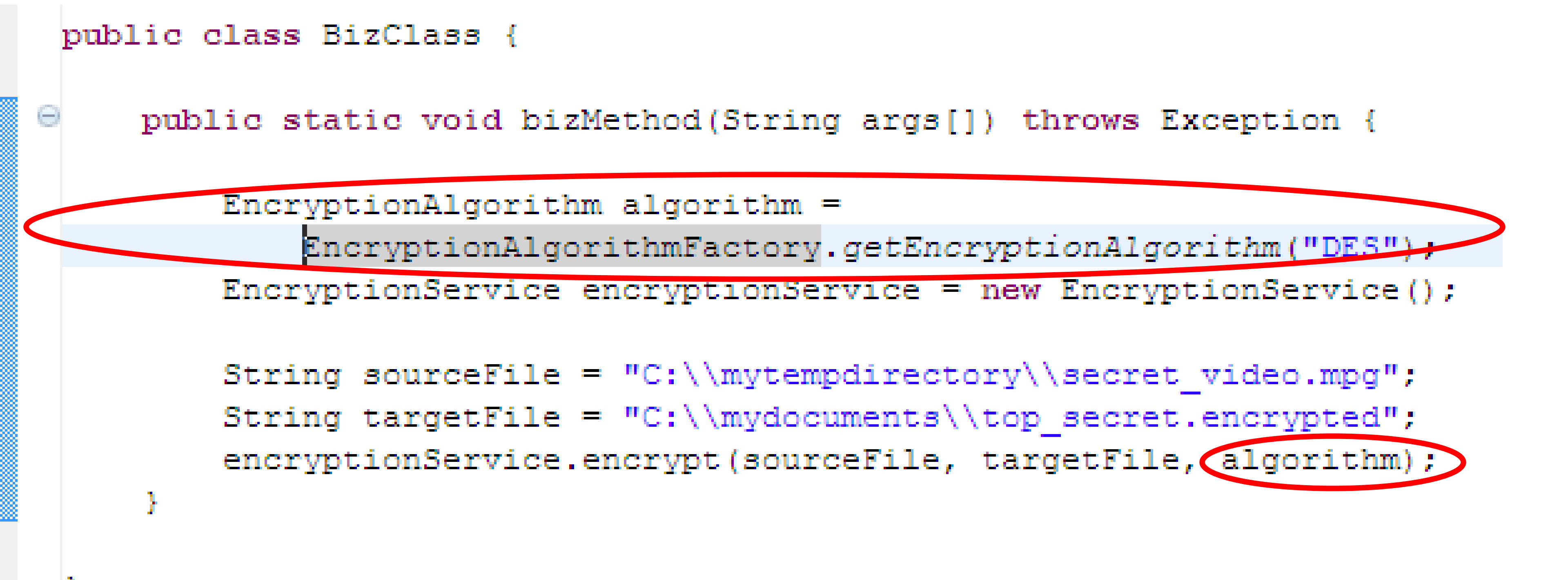
SRP partially resolved



```java
abstract class EncryptionAlgorithm {
    public abstract byte[] doEncryption(byte[] content);
}

class DESAlgorithm extends EncryptionAlgorithm {
    public byte[] doEncryption(byte[] content) {
        // TODO encrypt according to DES algorithm here
        return null;
    }
}

class AESAlgorithm extends EncryptionAlgorithm {
    public byte[] doEncryption(byte[] content) {
        // TODO encrypt according to AES algorithm here
        return null;
    }
}
```

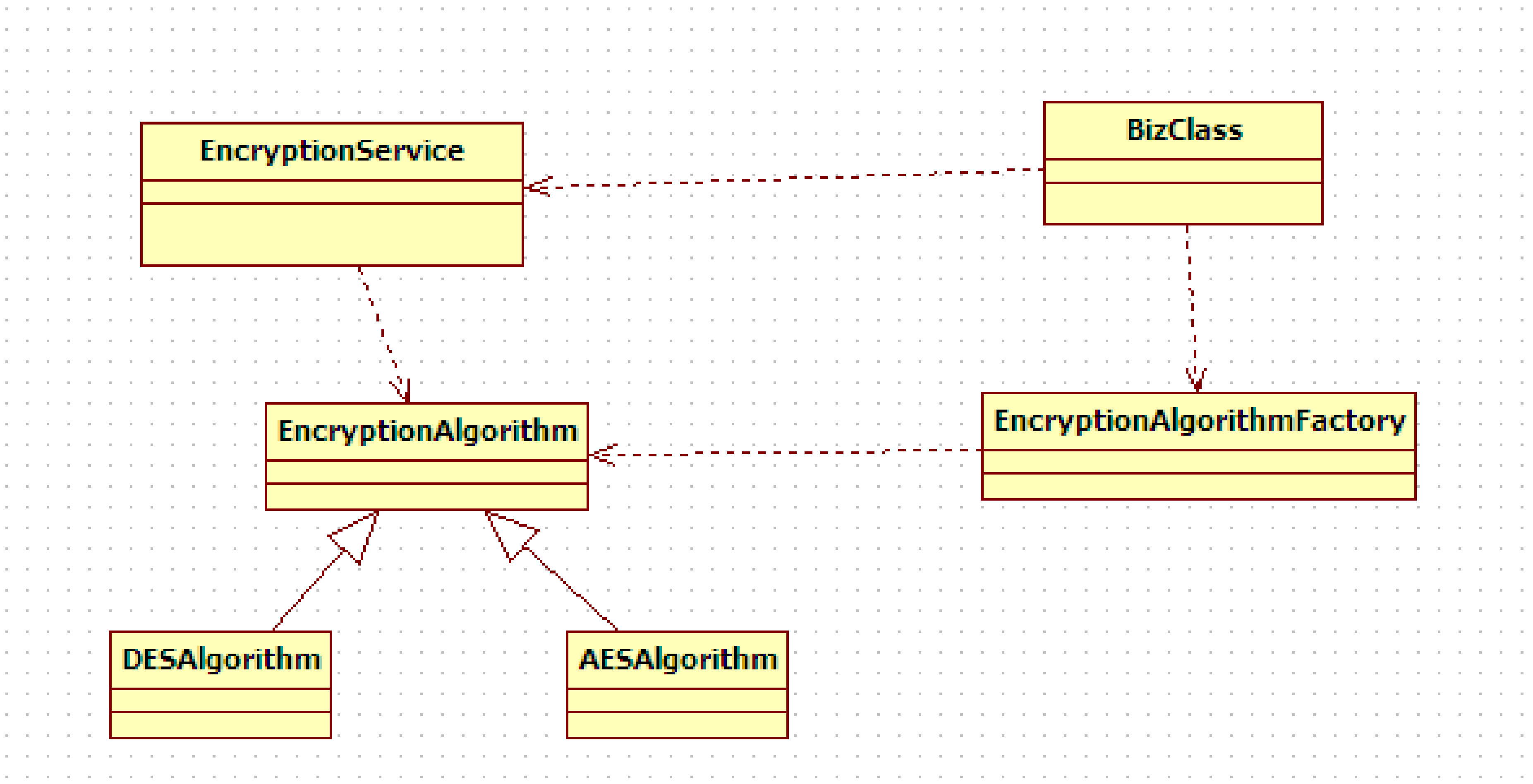


```
class EncryptionAlgorithmFactory {
    private static Map<String, EncryptionAlgorithm> encryptionAlgorithms =
        new HashMap<String, EncryptionAlgorithm>();
    static {
        encryptionAlgorithms.put("DES", new DESAlgorithm());
        encryptionAlgorithms.put("AES", new AESAlgorithm());
    }

    public static EncryptionAlgorithm getEncryptionAlgorithm(String algorithm) {
        return encryptionAlgorithms.get(algorithm);
    }

}
```



```
public class BizClass {

    public static void bizMethod(String args[]) throws Exception {

        EncryptionAlgorithm algorithm =
            EncryptionAlgorithmFactory.getEncryptionAlgorithm("DES");
        EncryptionService encryptionService = new EncryptionService();

        String sourceFile = "C:\\mytempdirectory\\secret_video.mpg";
        String targetFile = "C:\\mydocuments\\top_secret.encrypted";
        encryptionService.encrypt(sourceFile, targetFile, algorithm);
    }
```

Revised Design



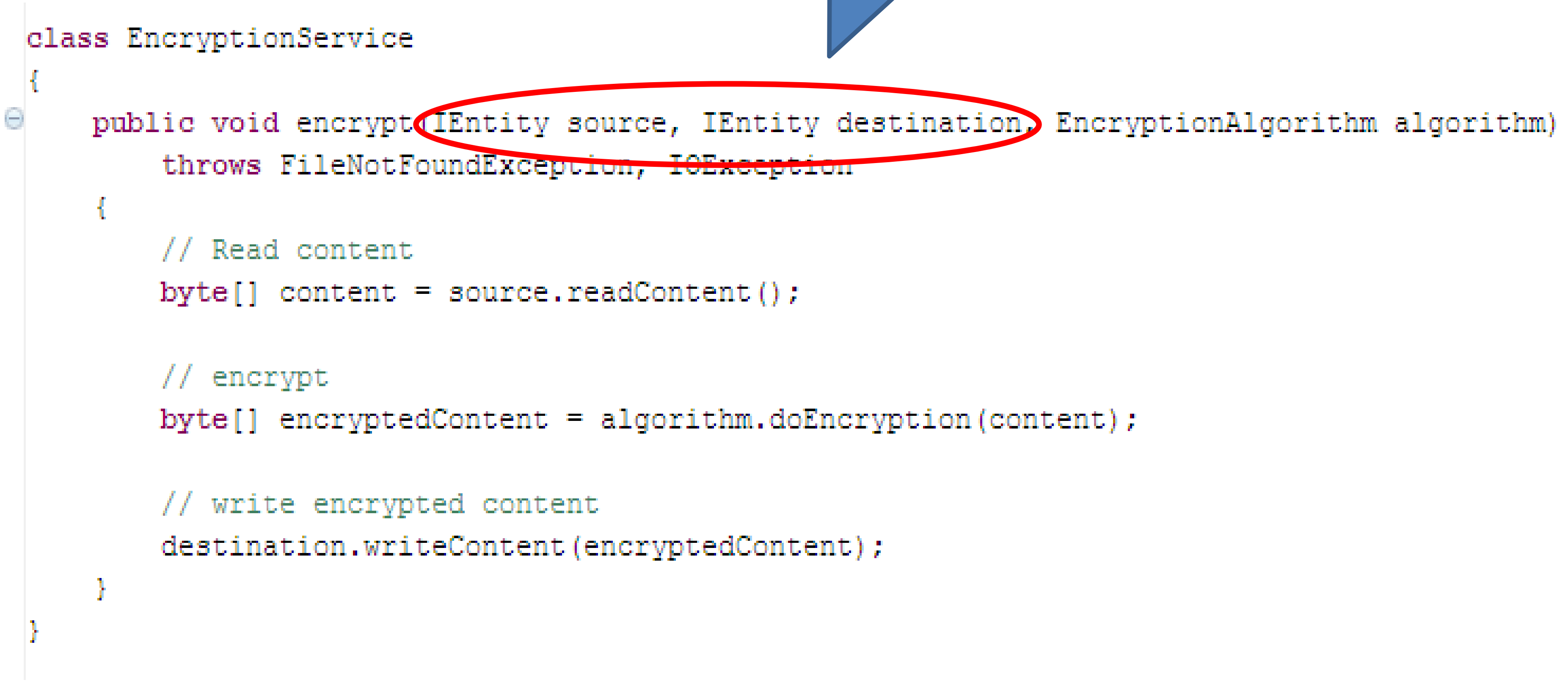
OCP violation
is resolved

SRP violation
is partially
resolved

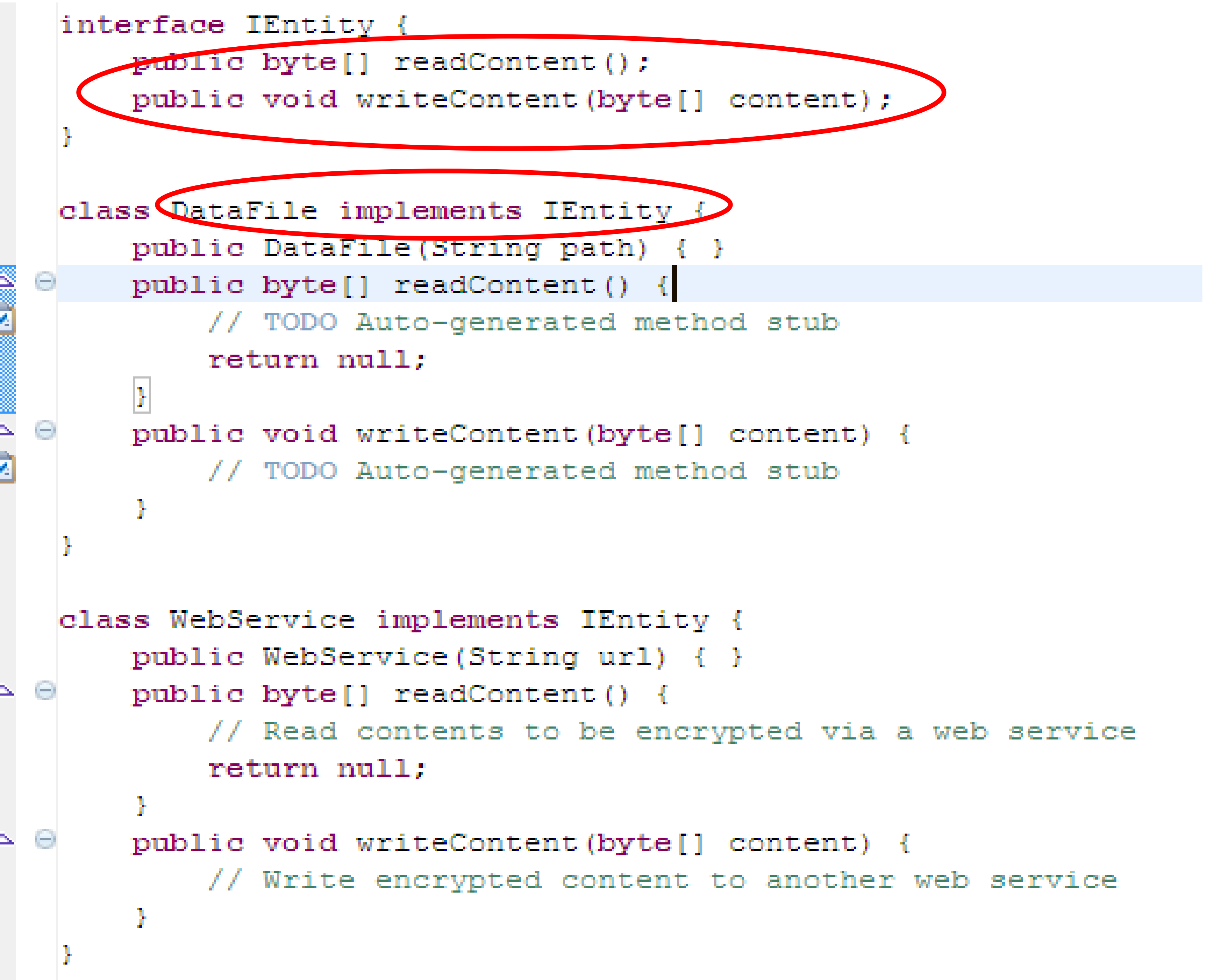
Project “eMystery”

- Sprint 3:
  - The **sources** or **targets** in the encryption service can either be files in the local hard drive, or a data set remotely accessible via a webservice.

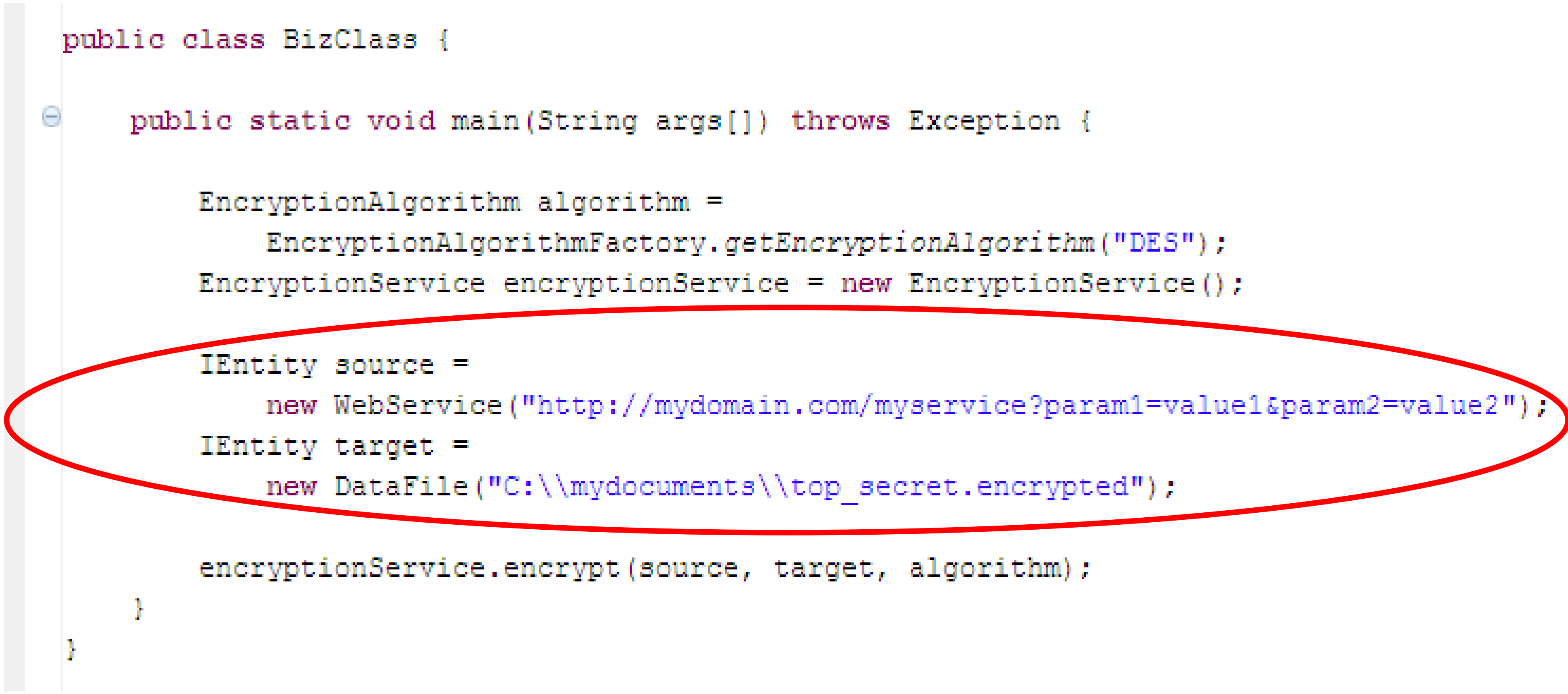
Solution with DIP and SRP



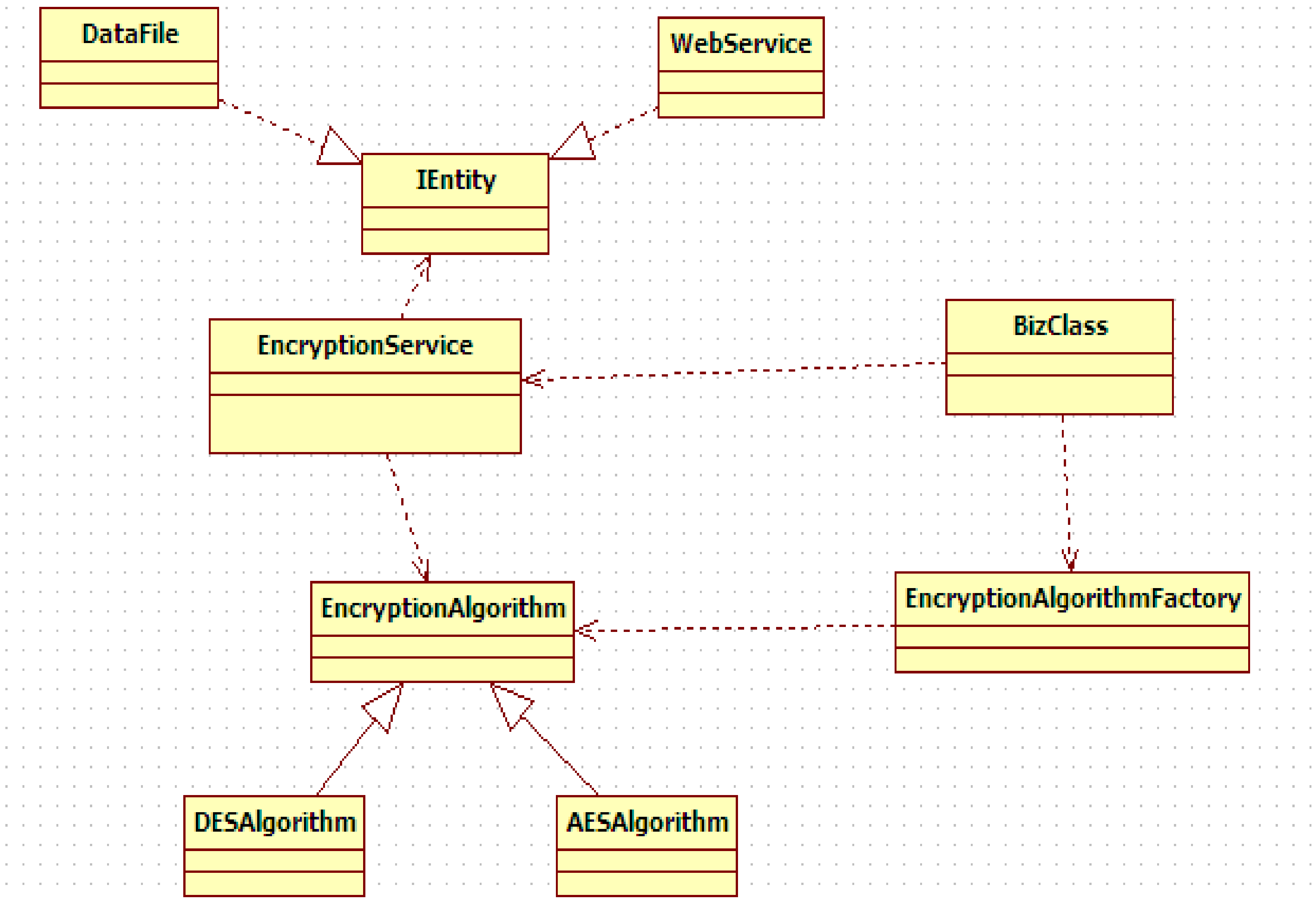
```java
class EncryptionService
{
    public void encrypt(IEntity source, IEntity destination, EncryptionAlgorithm algorithm)
        throws FileNotFoundException, IOException
    {
        // Read content
        byte[] content = source.readContent();

        // encrypt
        byte[] encryptedContent = algorithm.doEncryption(content);

        // write encrypted content
        destination.writeContent(encryptedContent);
    }
}
```



```
interface IEntity {
    public byte[] readContent();
    public void writeContent(byte[] content);
}

class DataFile implements IEntity {
    public DataFile(String path) { }
    public byte[] readContent() {
        // TODO Auto-generated method stub
        return null;
    }
    public void writeContent(byte[] content) {
        // TODO Auto-generated method stub
    }
}

class WebService implements IEntity {
    public WebService(String url) { }
    public byte[] readContent() {
        // Read contents to be encrypted via a web service
        return null;
    }
    public void writeContent(byte[] content) {
        // Write encrypted content to another web service
    }
}
```

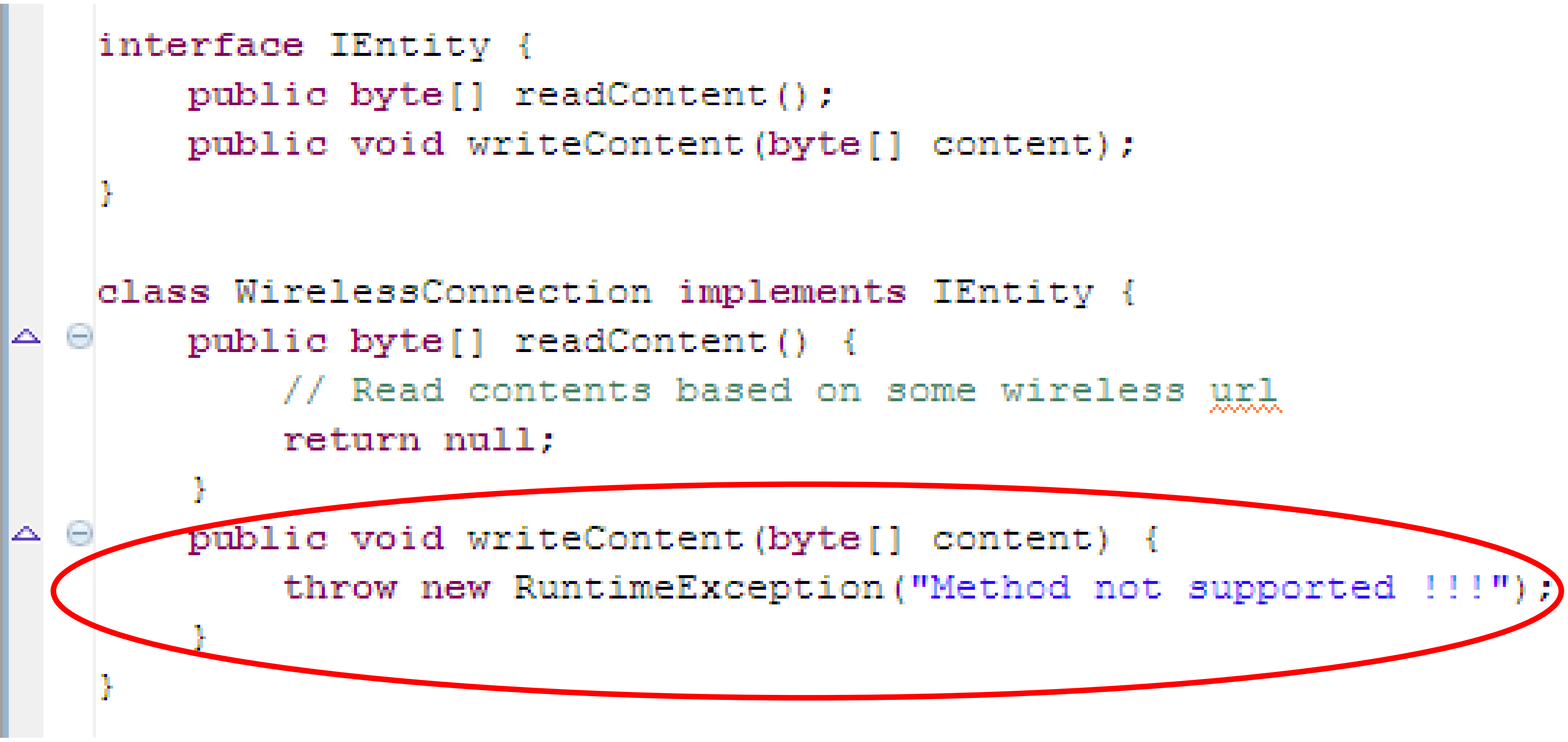


```
public class BizClass {

    public static void main(String args[]) throws Exception {

        EncryptionAlgorithm algorithm =
            EncryptionAlgorithmFactory.getEncryptionAlgorithm("DES");
        EncryptionService encryptionService = new EncryptionService();

        IEntity source =
            new WebService("http://mydomain.com/myservice?param1=value1&param2=value2");
        IEntity target =
            new DataFile("C:\\mydocuments\\top_secret.encrypted");

        encryptionService.encrypt(source, target, algorithm);
    }
}
```


DataFile
WebService
IEntity
EncryptionService
BizClass
EncryptionAlgorithm
EncryptionAlgorithmFactory
DESAlgorithm
AESAlgorithm

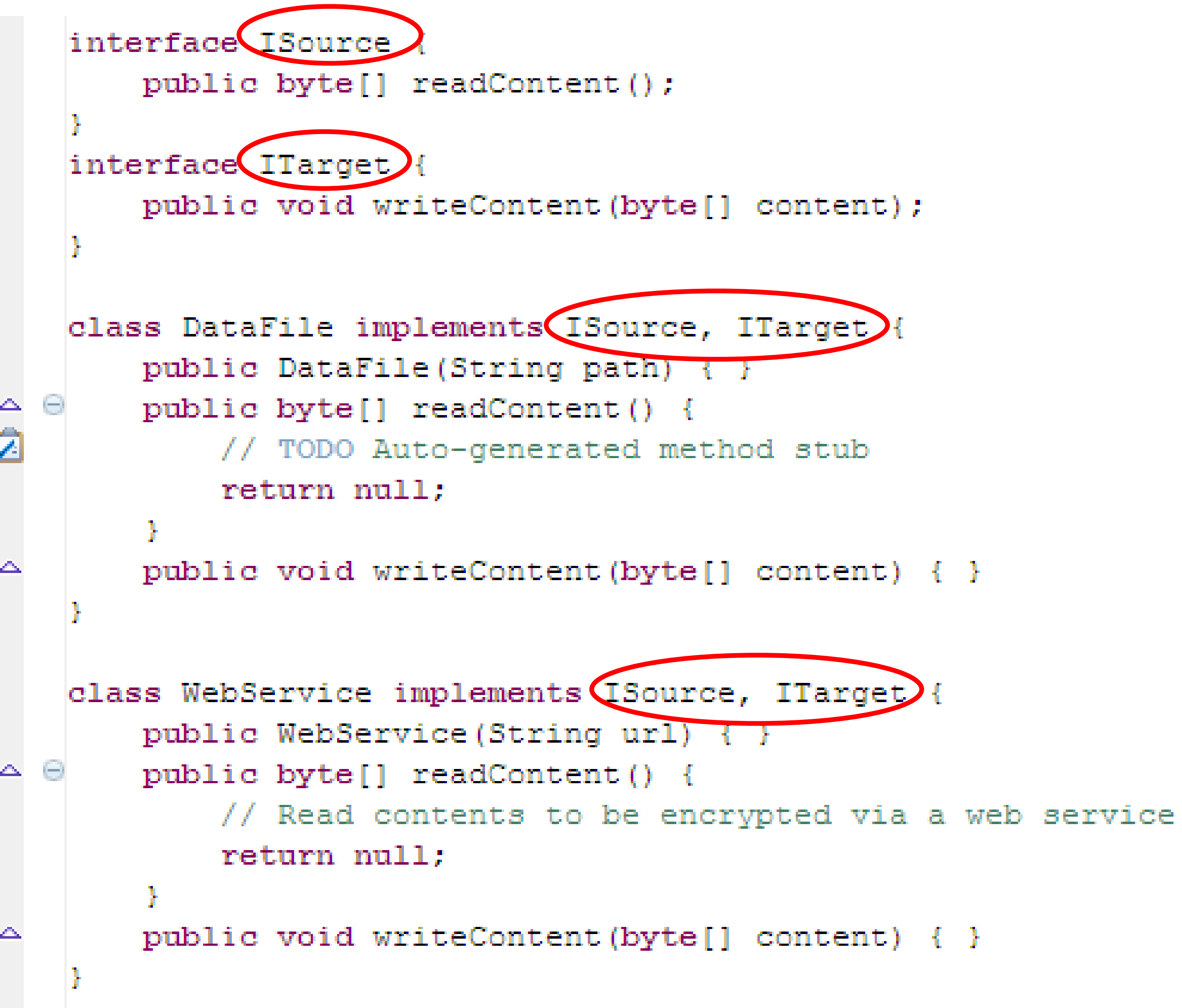
Project “eMystery”

- Sprint 4:
  - **Wireless Connections** will **ONLY** act as **sources** for the encryption service

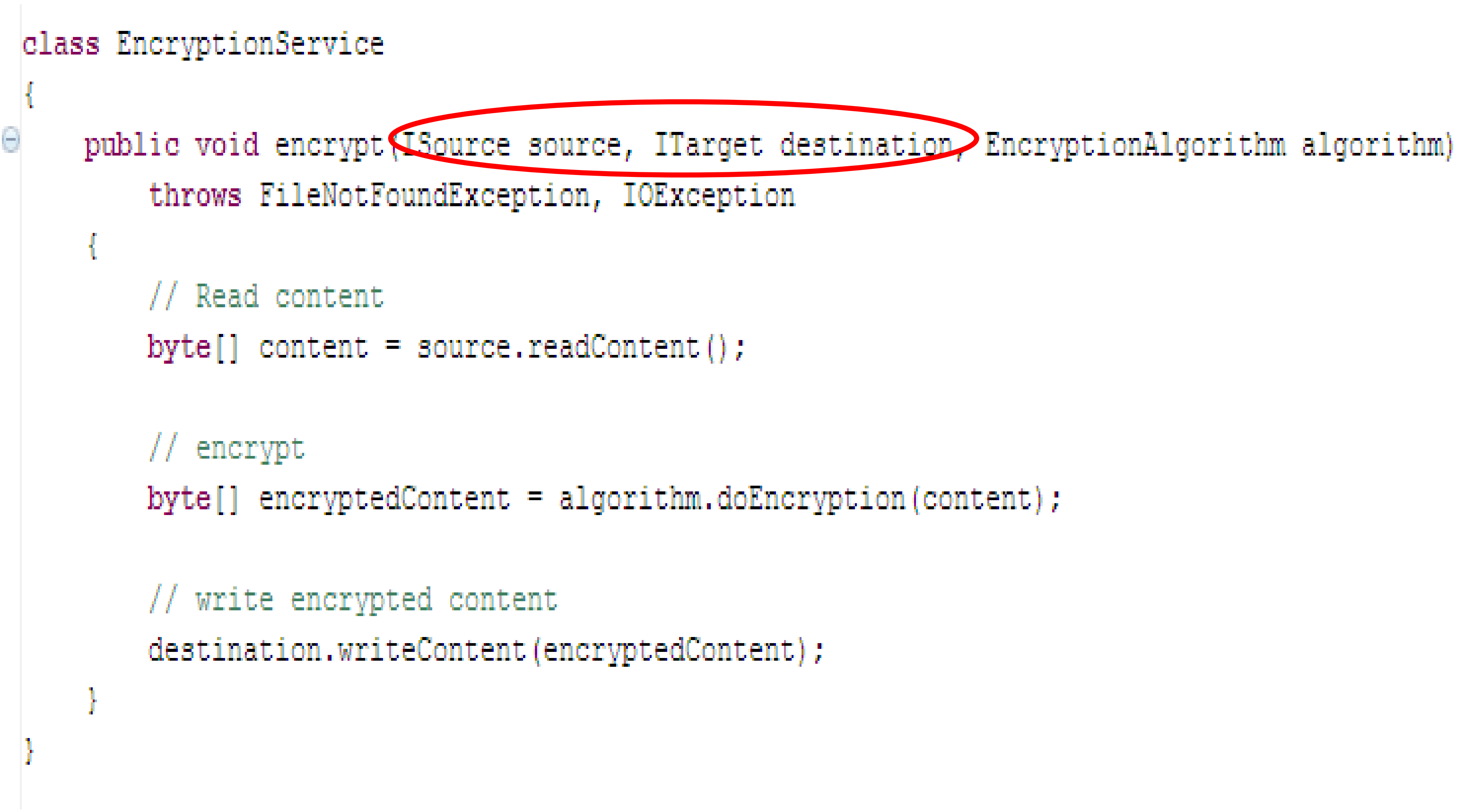


```
interface IEntity {
    public byte[] readContent();
    public void writeContent(byte[] content);
}

class WirelessConnection implements IEntity {
    public byte[] readContent() {
        // Read contents based on some wireless url
        return null;
    }
    public void writeContent(byte[] content) {
        throw new RuntimeException("Method not supported !!!");
    }
}
```

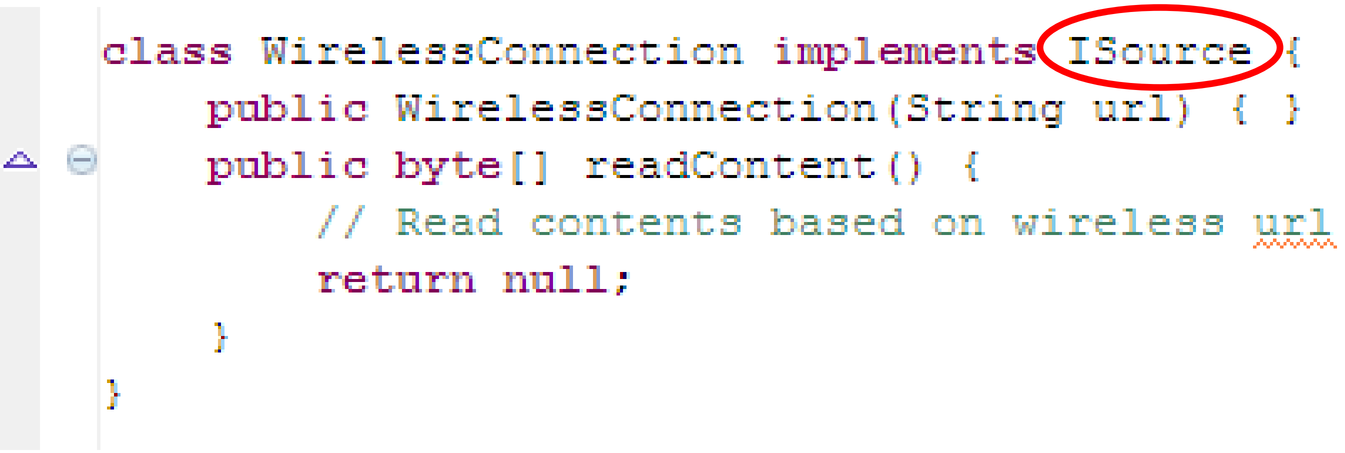
The previous solution violated ISP



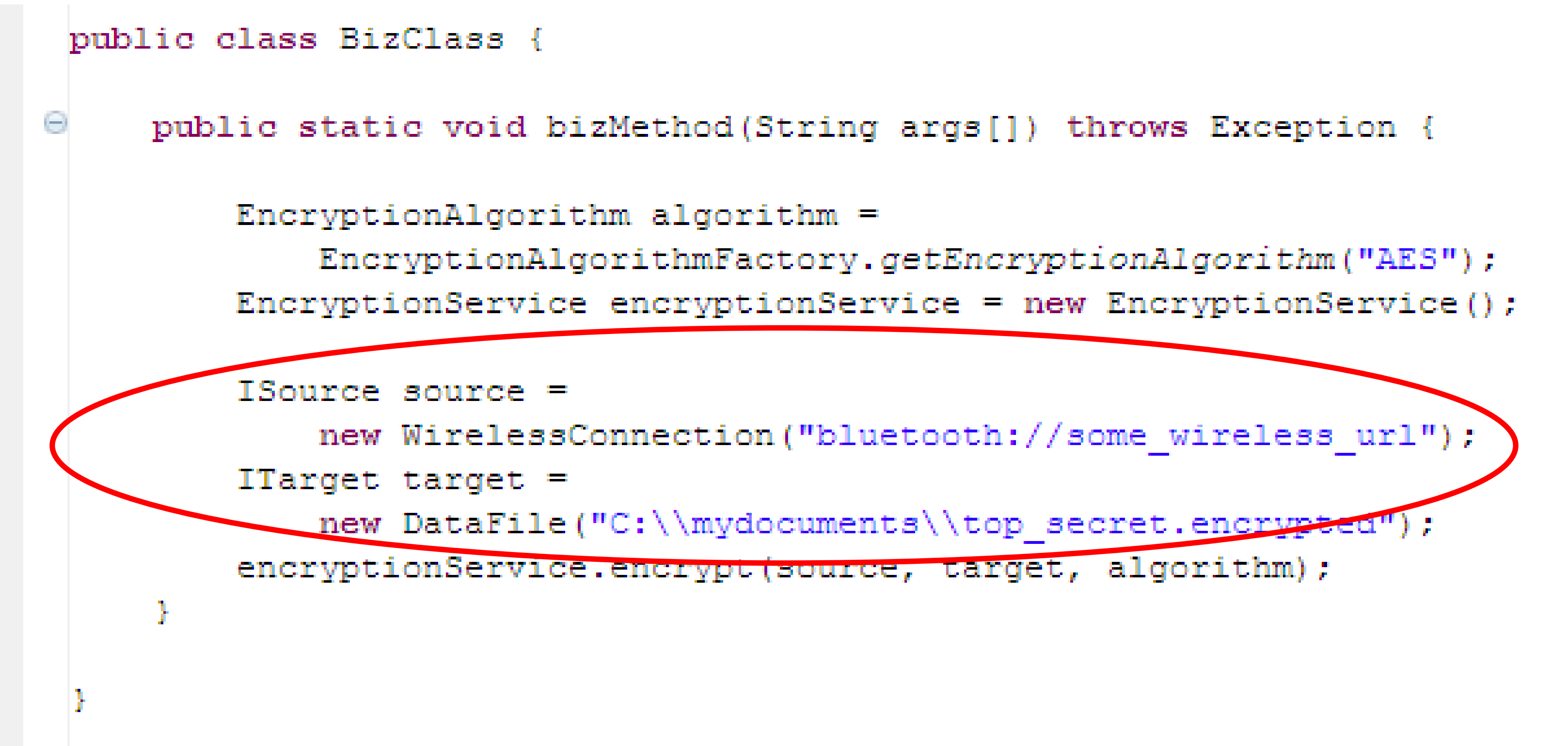
```java
interface ISource {
    public byte[] readContent();
}
interface ITarget {
    public void writeContent(byte[] content);
}

class DataFile implements ISource, ITarget {
    public DataFile(String path) { }
    public byte[] readContent() {
        // TODO Auto-generated method stub
        return null;
    }
    public void writeContent(byte[] content) { }
}

class WebService implements ISource, ITarget {
    public WebService(String url) { }
    public byte[] readContent() {
        // Read contents to be encrypted via a web service
        return null;
    }
    public void writeContent(byte[] content) { }
}
```



```
class EncryptionService
{
    public void encrypt(ISource source, ITarget destination, EncryptionAlgorithm algorithm)
        throws FileNotFoundException, IOException
    {
        // Read content
        byte[] content = source.readContent();

        // encrypt
        byte[] encryptedContent = algorithm.doEncryption(content);

        // write encrypted content
        destination.writeContent(encryptedContent);
    }
}
```

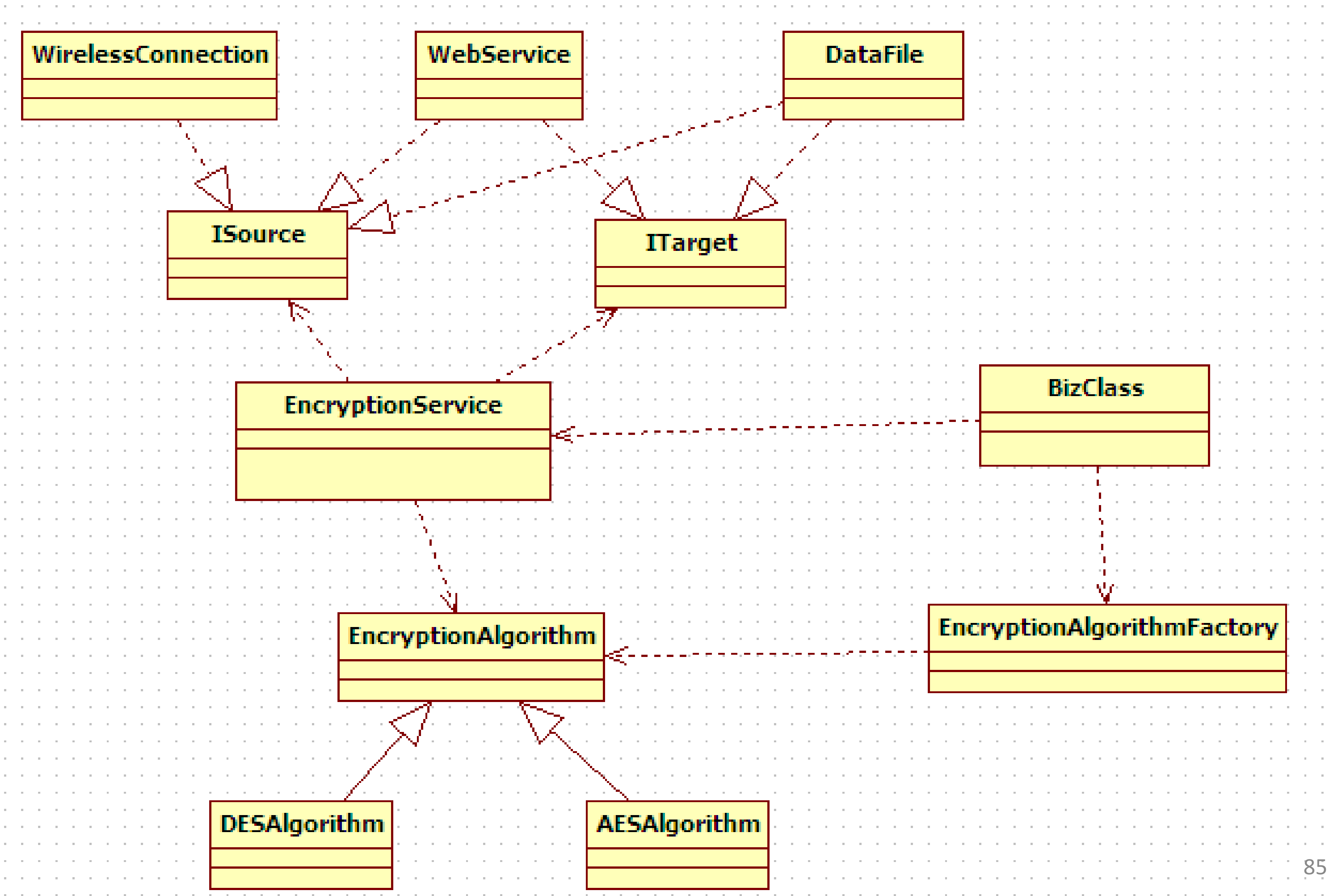


```
class WirelessConnection implements ISource {
    public WirelessConnection(String url) { }
    public byte[] readContent() {
        // Read contents based on wireless url
        return null;
    }
}
```

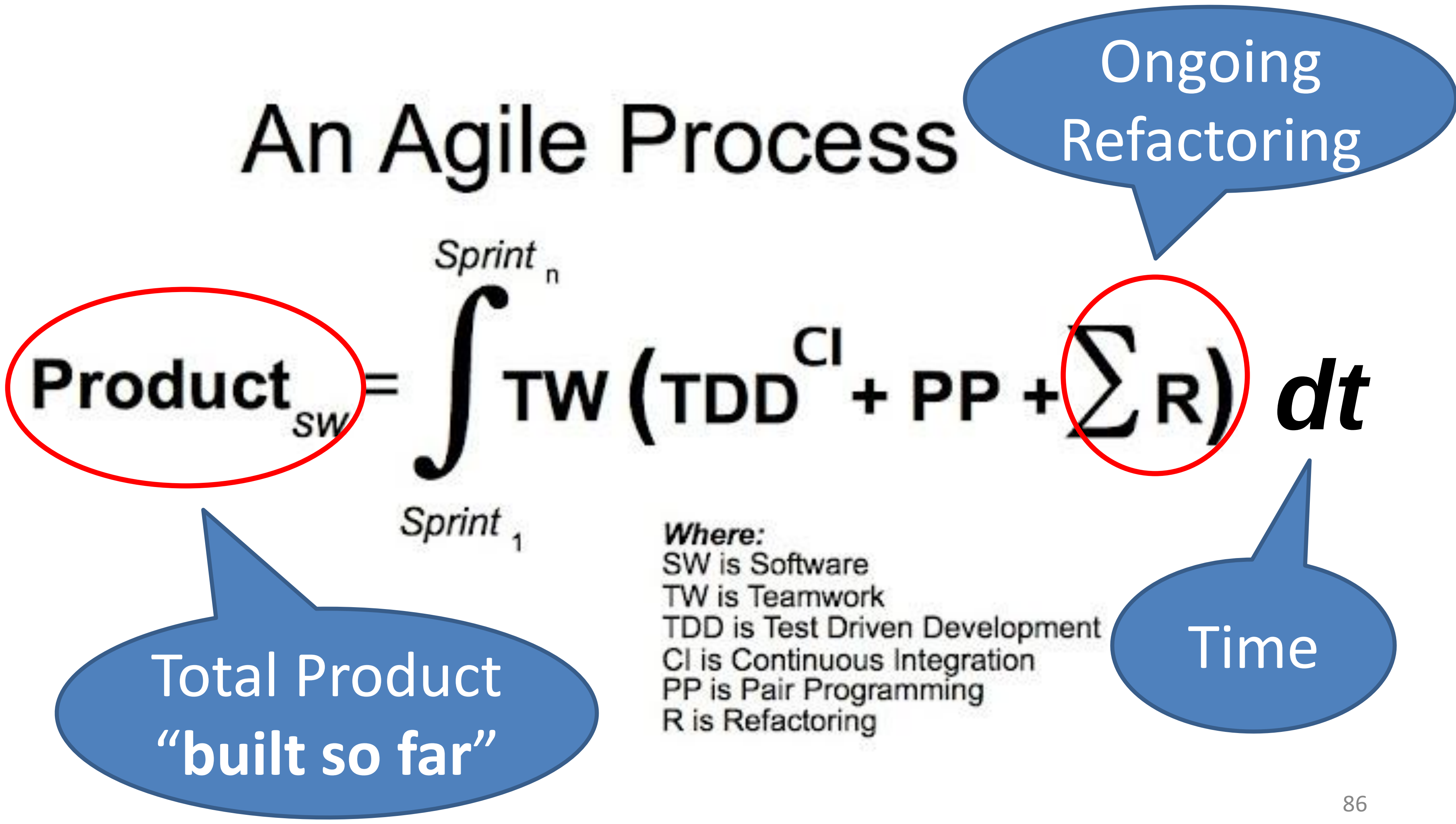


```
public class BizClass {

    public static void bizMethod(String args[]) throws Exception {

        EncryptionAlgorithm algorithm =
            EncryptionAlgorithmFactory.getEncryptionAlgorithm("AES");
        EncryptionService encryptionService = new EncryptionService();

        ISource source =
            new WirelessConnection("bluetooth://some_wireless_url");
        ITarget target =
            new DataFile("C:\\mydocuments\\top_secret.encrypted");
        encryptionService.encrypt(source, target, algorithm);
    }

}
```

Revised Design



A Developer's Recipe across multiple Agile Sprints



An Agile Process

$$\text{Product}_{SW} = \int_{Sprint_1}^{Sprint_n} TW\left(TDD^{CI} + PP + \sum R\right) dt$$

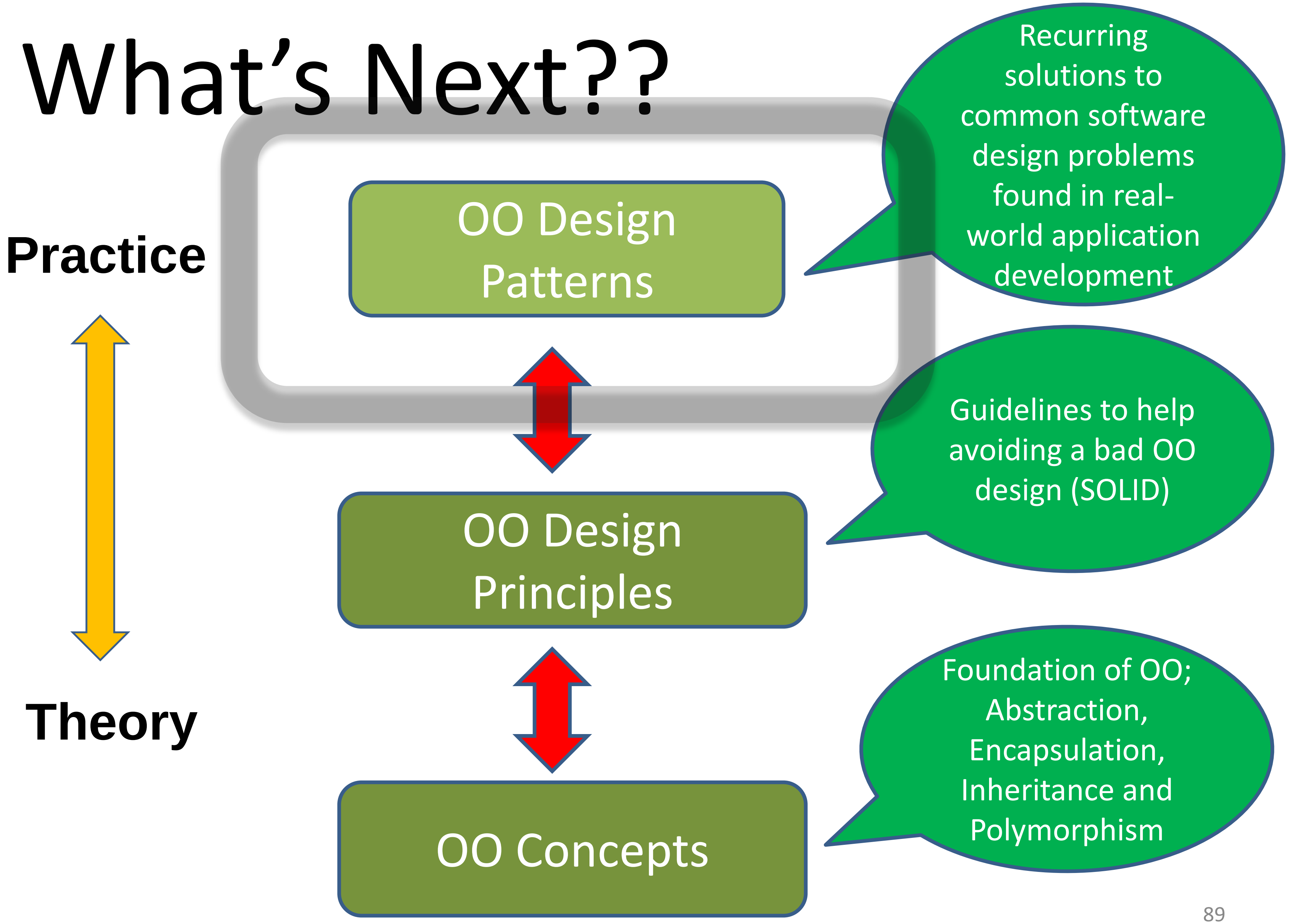
Where:
SW is Software
TW is Teamwork
TDD is Test Driven Development
CI is Continuous Integration
PP is Pair Programming
R is Refactoring

Ongoing Refactoring
Total Product "**built so far**"
Time

What's the reason behind "**EncryptionAlgorithm**" being an abstract class? Why not use an **interface**???

In Java JDBC;
<u>Connection, Statement, PreparedStatement, ResultSet</u> are *interfaces*. None of them are *classes*. Then where's the implementation? How java is connected with databases ???

What’s Next??


Practice
Theory
OO Design Patterns
OO Design Principles
OO Concepts
Recurring solutions to common software design problems found in real-world application development
Guidelines to help avoiding a bad OO design (SOLID)
Foundation of OO; Abstraction, Encapsulation, Inheritance and Polymorphism

Q & A