DePaul University

Design Principles:

S.O.L.I.D.

Object-oriented Software Development
SE 350– Spring 2021

Vahid Alizadeh

Week 6.1
May 4, 2021

Announcements

Future Schedule

Assignment 2 will be graded by Wed

Midterm will be graded by Sun

- ~~**Assignment 1**~~
- ~~**Assignment 2**~~
- ~~**Mid Term Exam**~~
- **Assignment 3:**
  - Release: Week 7
  - Due: Week 8
- **Assignment 4:**
  - Release: Week 8
  - Due: Week 9
- **Bonus Research Project:**
  - Presentation Due: Week 10
  - Report Due: Week 11
- **Final Exam:**
  - Week 11

Bonus Credits: Research Paper & Presentation

- **Research Presentation**
  - **Due** Week 10
  - Max 10 slides - ~7 min talk
  - **Template**: your choice!
- **Research Report**
  - **Due** Week 11
  - Writing requirement 3-4 pages
  - At least 2 external references
    - (Conference paper, articles, journals, books)
  - **Template**: ACM Proceedings (Link)
    - LaTex or Word Template (Also uploaded on D2L)
    - Overleaf Latex template (Link)
- **Research on**
  - Object-oriented programming related concepts/principles
  - Design Patterns topics.
- **Select and Announce Your Topic On:**
  - MS Teams: Research Project Channel

Bonus Credits: Research Topics and Resources

- **Some resources to find research articles and books:**
  - Google Scholar
  - Scopus
  - IEEEXplore
  - DePaul Library
- **A great resource to find related topics/resources:**
  - SemanticScholar
- **Some Topics:**
  - Design Patterns
    - **NO** Singleton, Abstract Factory, Builder, Factory method, Decorator, Adapter, Proxy
  - Object oriented metrics: QMOOD, MOOD, C&K, …
  - Design Antipatterns
  - Impact of Design Principles and Patterns
    - Pros and Cons

- **Some papers:**
  - Olague, Hector M., et al. "**Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes.**" IEEE Transactions on software Engineering 33.6 (2007): 402-419.
  - Maurer, S.. "**Design Patterns Explained A New Perspective On Object Oriented Design**." (2016).
  - Subburaj, R. et al. "**Impact of Object Oriented Design Patterns on Software Development**." (2015).
  - Dong, J. et al. "**A Review of Design Pattern Mining Techniques**." Int. J. Softw. Eng. Knowl. Eng. 19 (2009): 823-855.
  - Jiang, S. and Huaxin Mu. "**Design patterns in object oriented analysis and design**." 2011 IEEE 2nd International Conference on Software Engineering and Service Science (2011): 326-329.
  - Din, Jamilah et al. "**A Review of the Antipatterns Detection Approaches in Object-Oriented Design.**" Journal of Convergence Information Technology 8 (2013): 518-527.
  - Aras, Mehmed Taha and Y. Selçuk. "**Metric and rule based automated detection of antipatterns in object-oriented software systems.**" 2016 7th International Conference on Computer Science and Information Technology (CSIT) (2016): 1-6.
  - Khomh, F.. "**Patterns and quality of object-oriented software systems**." (2010).
  - Abbes, Marwen et al. "**An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, on Program Comprehension**." 2011 15th European Conference on Software Maintenance and Reengineering (2011): 181-190.
  - Plösch, Reinhold et al. "**Measuring, Assessing and Improving Software Quality based on Object-Oriented Design Principles**." Open Computer Science 6 (2016): 187 - 207.

Design Principles

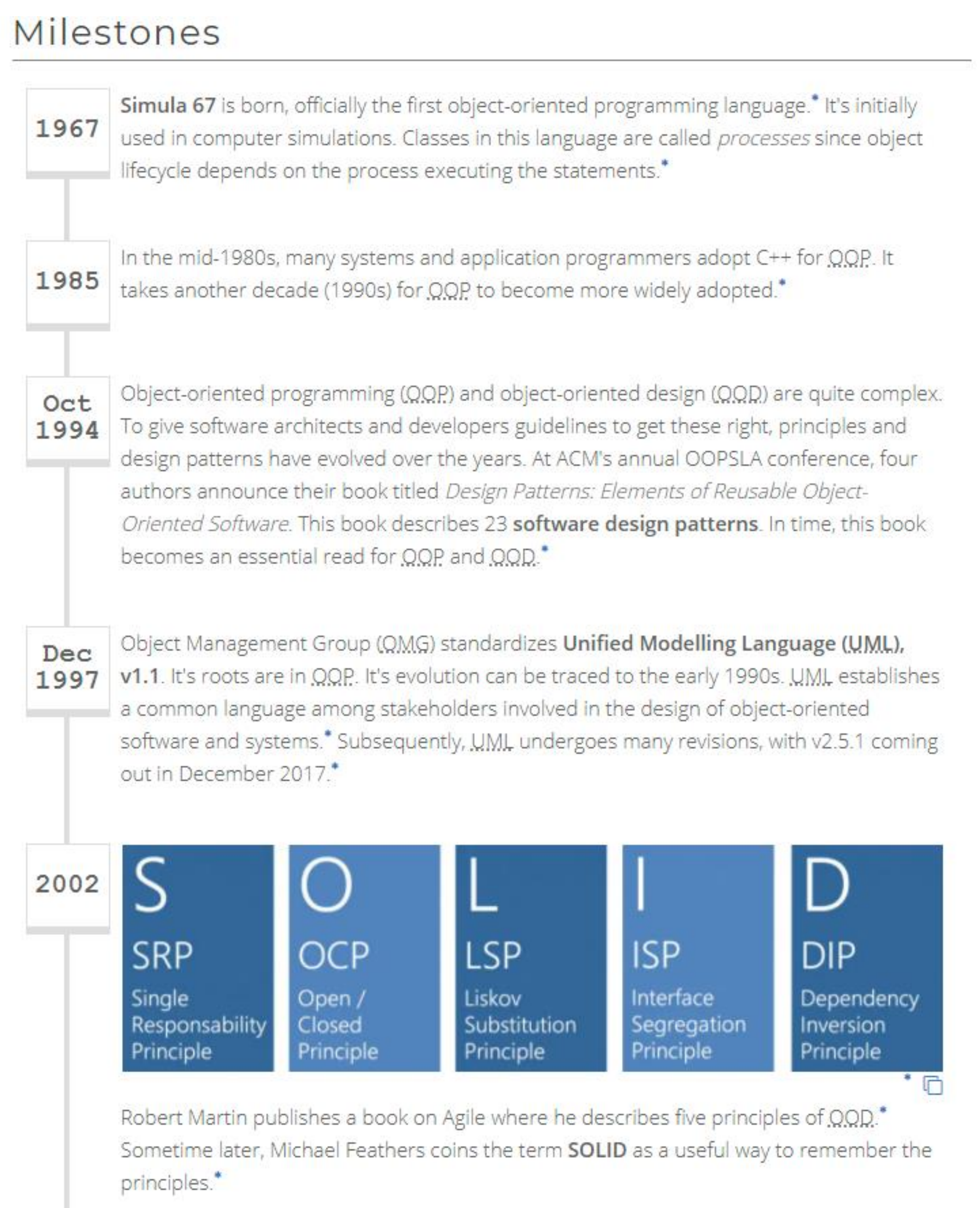
SOLID

Milestones – SOLID Principles

Martin, Robert C. "Design principles and design patterns." Object Mentor 1.34 (2000):

Link to the paper

To create understandable, readable, and testable code that many developers can collaboratively work on.



Milestones

1967 — **Simula 67** is born, officially the first object-oriented programming language.* It's initially used in computer simulations. Classes in this language are called *processes* since object lifecycle depends on the process executing the statements.*

1985 — In the mid-1980s, many systems and application programmers adopt C++ for OOP. It takes another decade (1990s) for OOP to become more widely adopted.*

Oct 1994 — Object-oriented programming (OOP) and object-oriented design (OOD) are quite complex. To give software architects and developers guidelines to get these right, principles and design patterns have evolved over the years. At ACM's annual OOPSLA conference, four authors announce their book titled *Design Patterns: Elements of Reusable Object-Oriented Software*. This book describes 23 **software design patterns**. In time, this book becomes an essential read for OOP and OOD.*

Dec 1997 — Object Management Group (OMG) standardizes **Unified Modelling Language (UML), v1.1**. It's roots are in OOP. It's evolution can be traced to the early 1990s. UML establishes a common language among stakeholders involved in the design of object-oriented software and systems.* Subsequently, UML undergoes many revisions, with v2.5.1 coming out in December 2017.*

2002

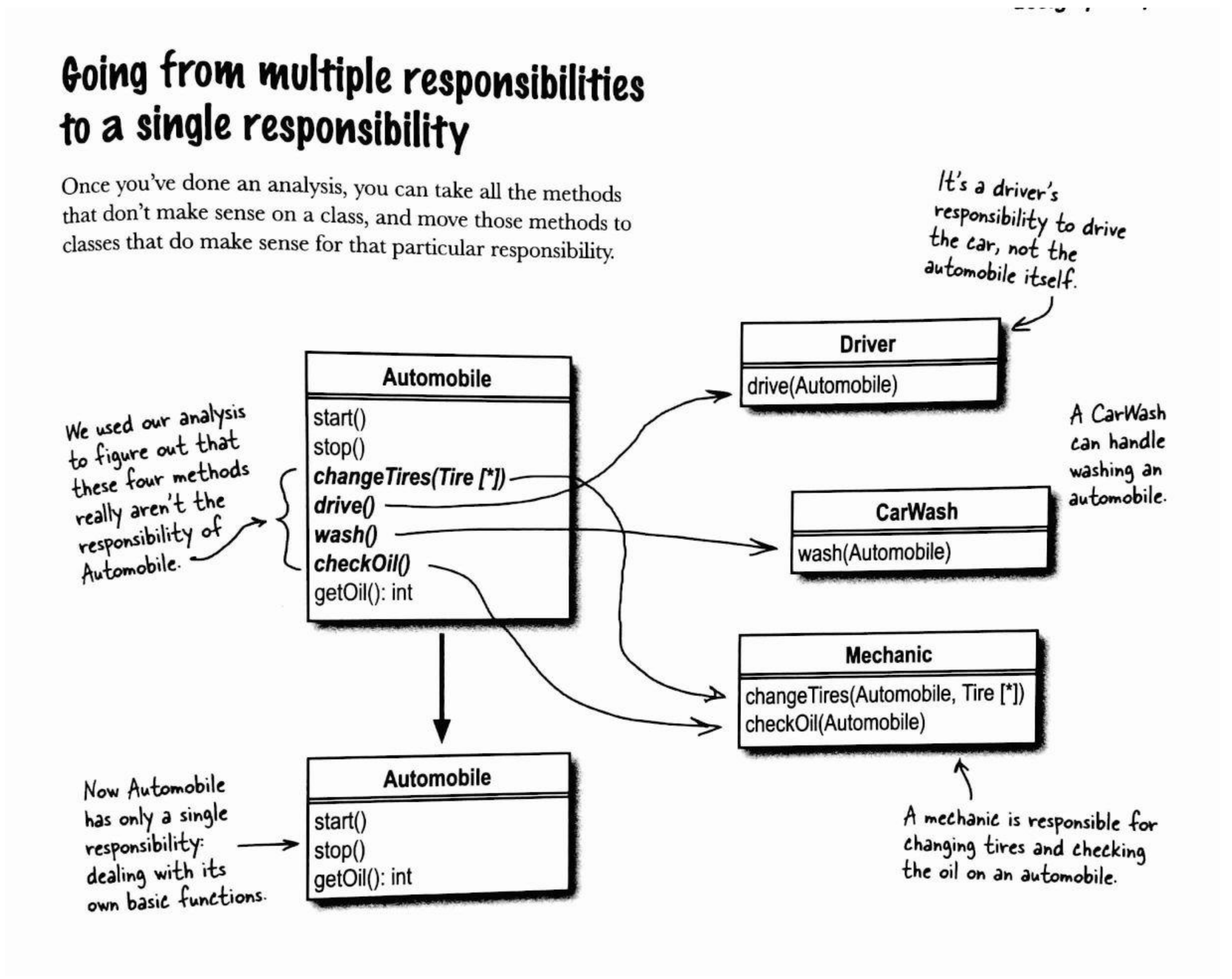
S	O	L	I	D
SRP	OCP	LSP	ISP	DIP
Single Responsability Principle	Open / Closed Principle	Liskov Substitution Principle	Interface Segregation Principle	Dependency Inversion Principle

Robert Martin publishes a book on Agile where he describes five principles of OOD.* Sometime later, Michael Feathers coins the term **SOLID** as a useful way to remember the principles.*

Single Responsibility Principle

A class should do one thing and therefore it should have only a single reason to change.

- **Benefits**
  - Easier testing
  - Less dependencies to other modules/classes
  - Better code organization
- **How to make sure your code follows the SRP?**

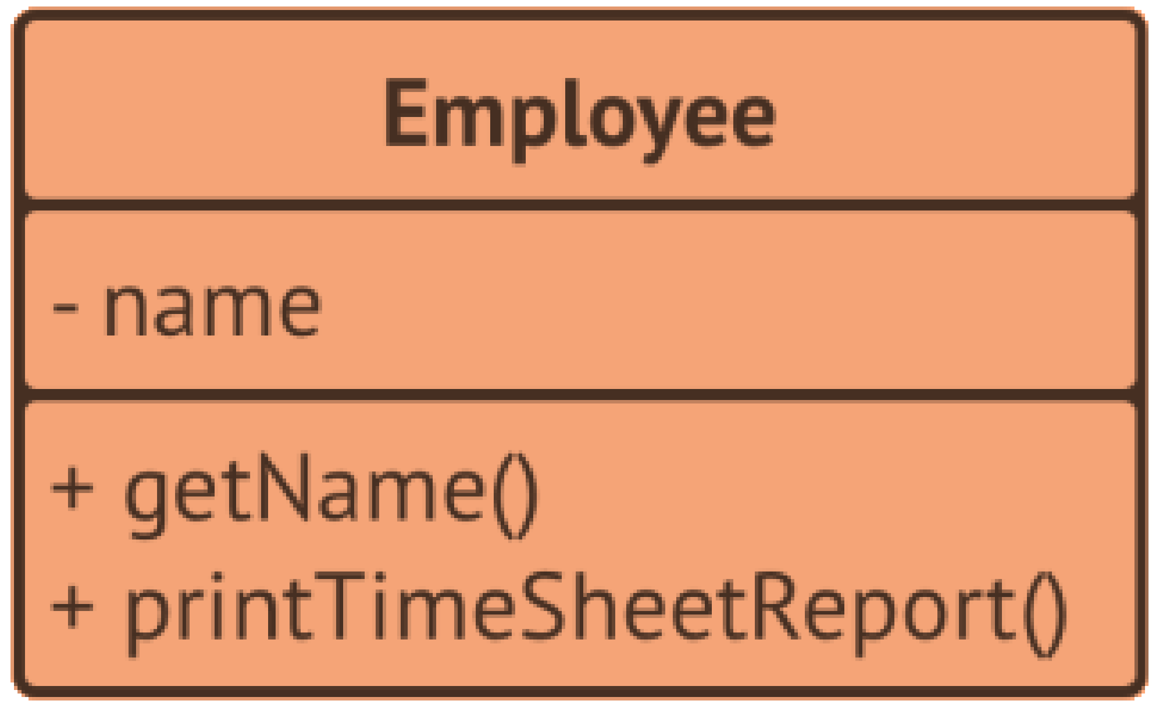


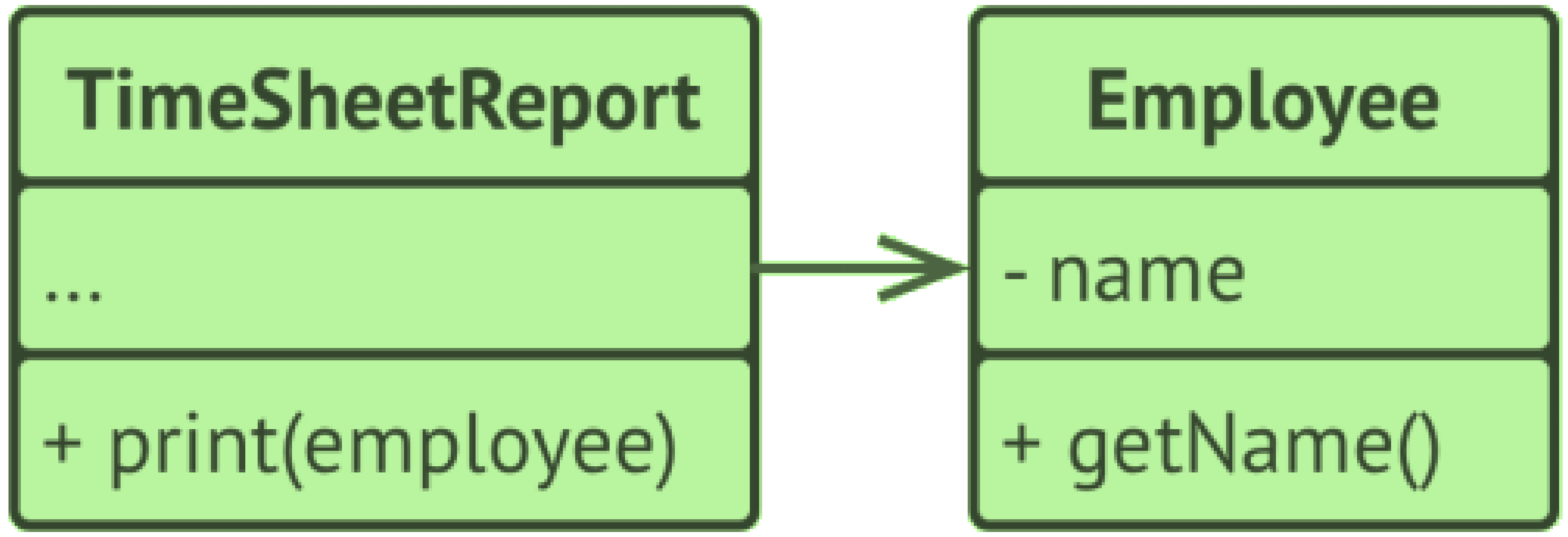
Going from multiple responsibilities to a single responsibility

Once you've done an analysis, you can take all the methods that don't make sense on a class, and move those methods to classes that do make sense for that particular responsibility.

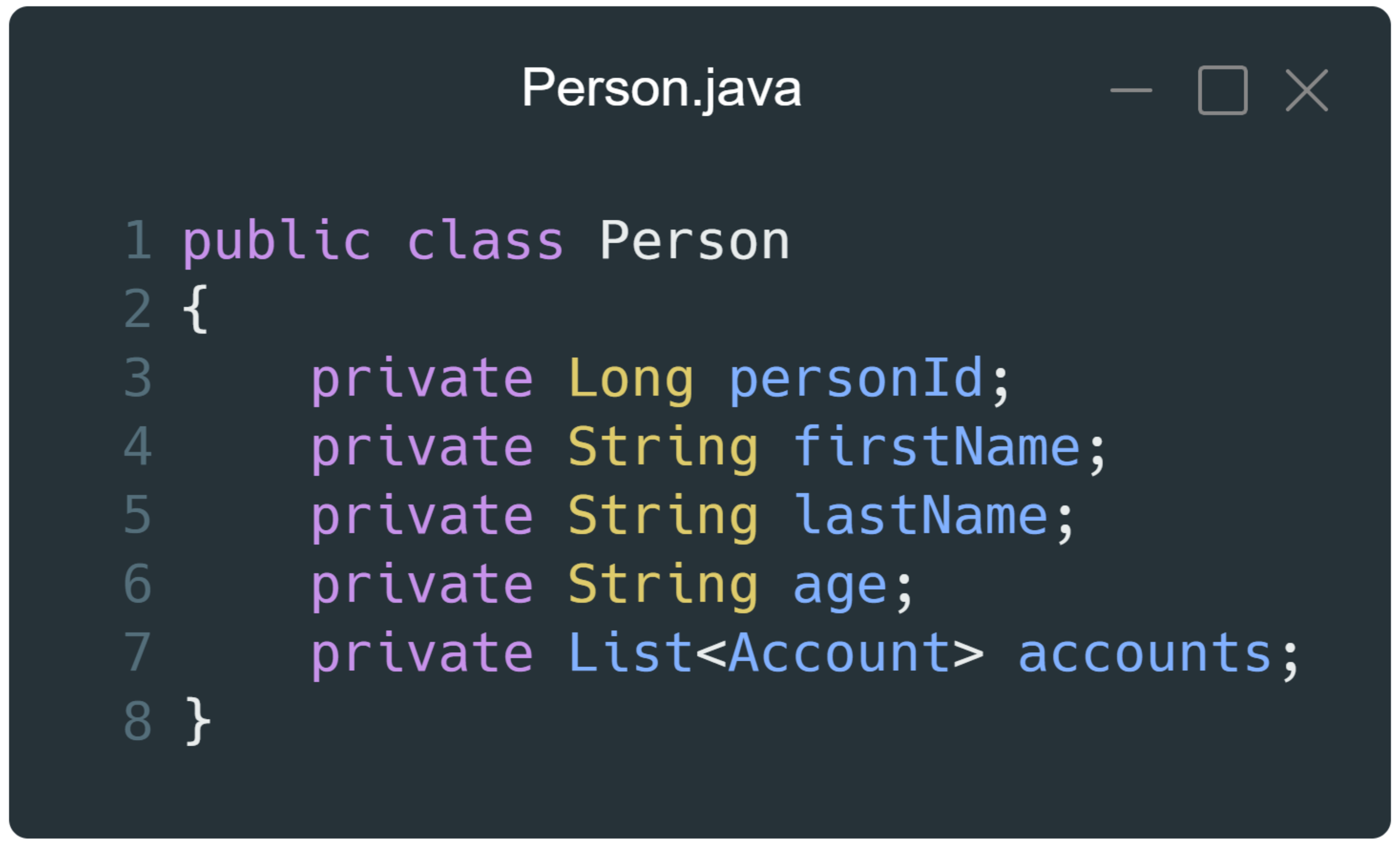
Automobile
start()
stop()
changeTires(Tire [*])
drive()
wash()
checkOil()
getOil(): int
We used our analysis to figure out that these four methods really aren't the responsibility of Automobile.
Driver
drive(Automobile)
It's a driver's responsibility to drive the car, not the automobile itself.
CarWash
wash(Automobile)
A CarWash can handle washing an automobile.
Mechanic
changeTires(Automobile, Tire [*])
checkOil(Automobile)
A mechanic is responsible for changing tires and checking the oil on an automobile.
Automobile
start()
stop()
getOil(): int
Now Automobile has only a single responsibility: dealing with its own basic functions.

SRP Example: Employee Timesheet

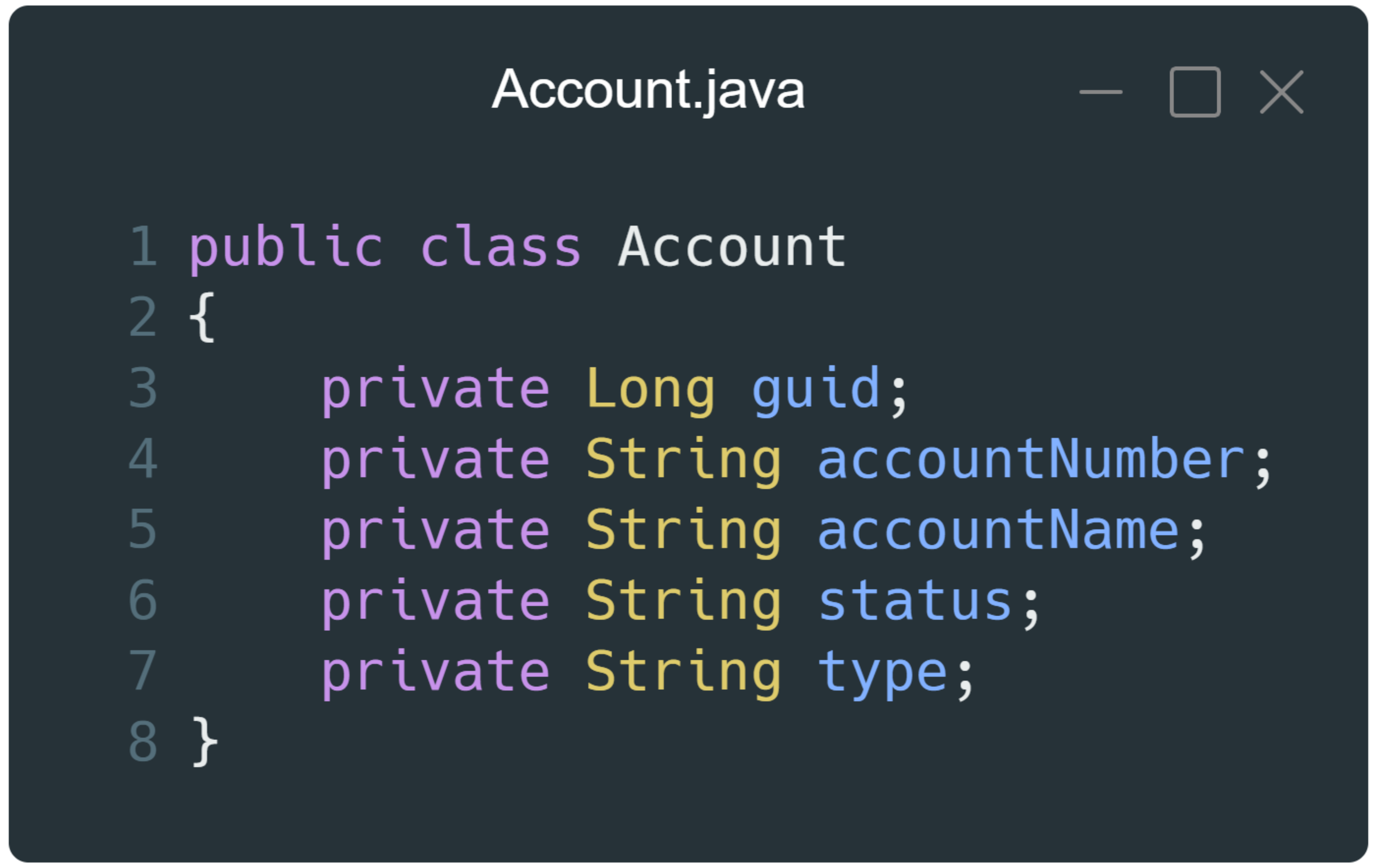




SRP Example 1

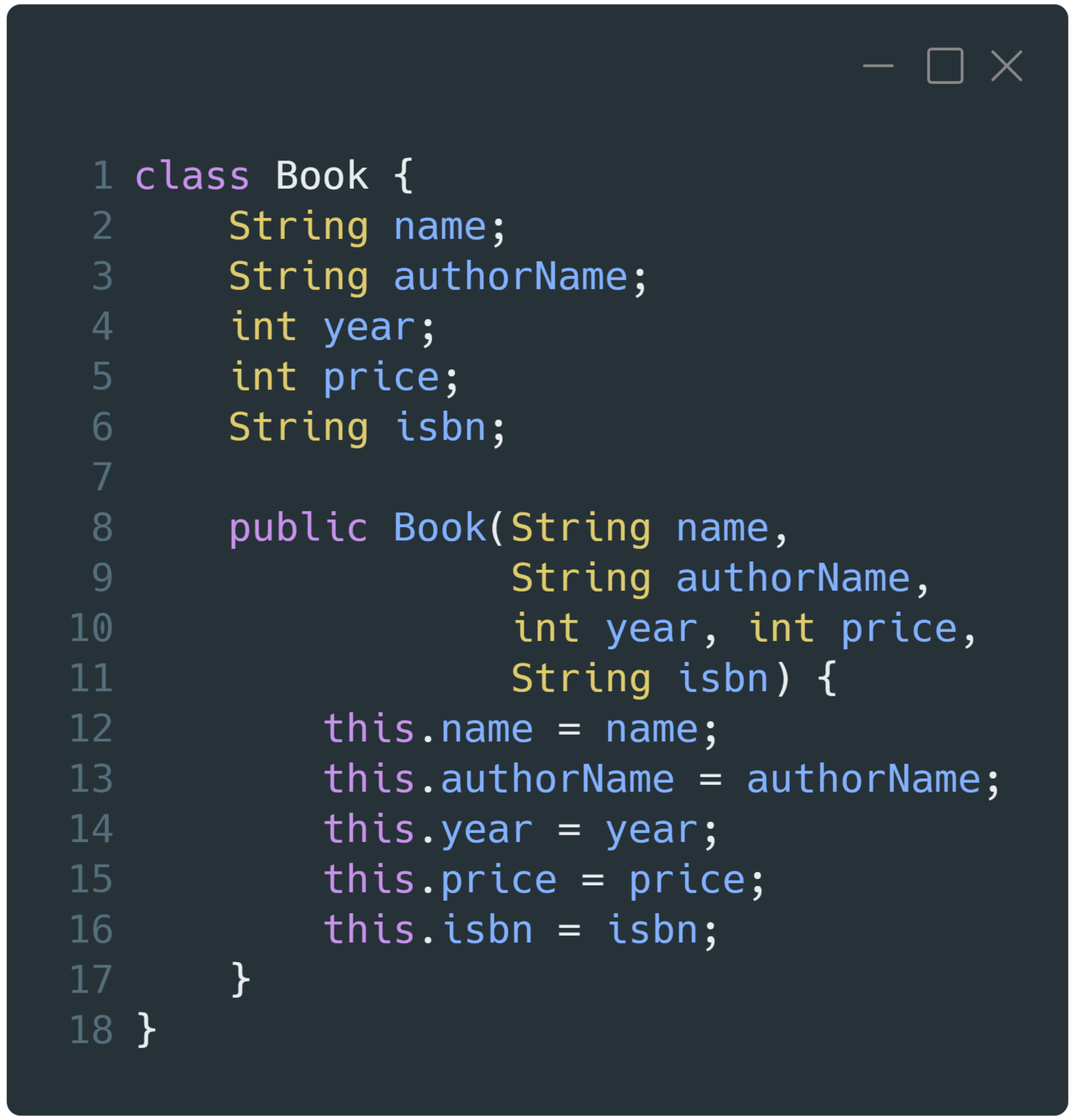

Person.java

```java
public class Person
{
    private Long personId;
    private String firstName;
    private String lastName;
    private String age;
    private List<Account> accounts;
}
```

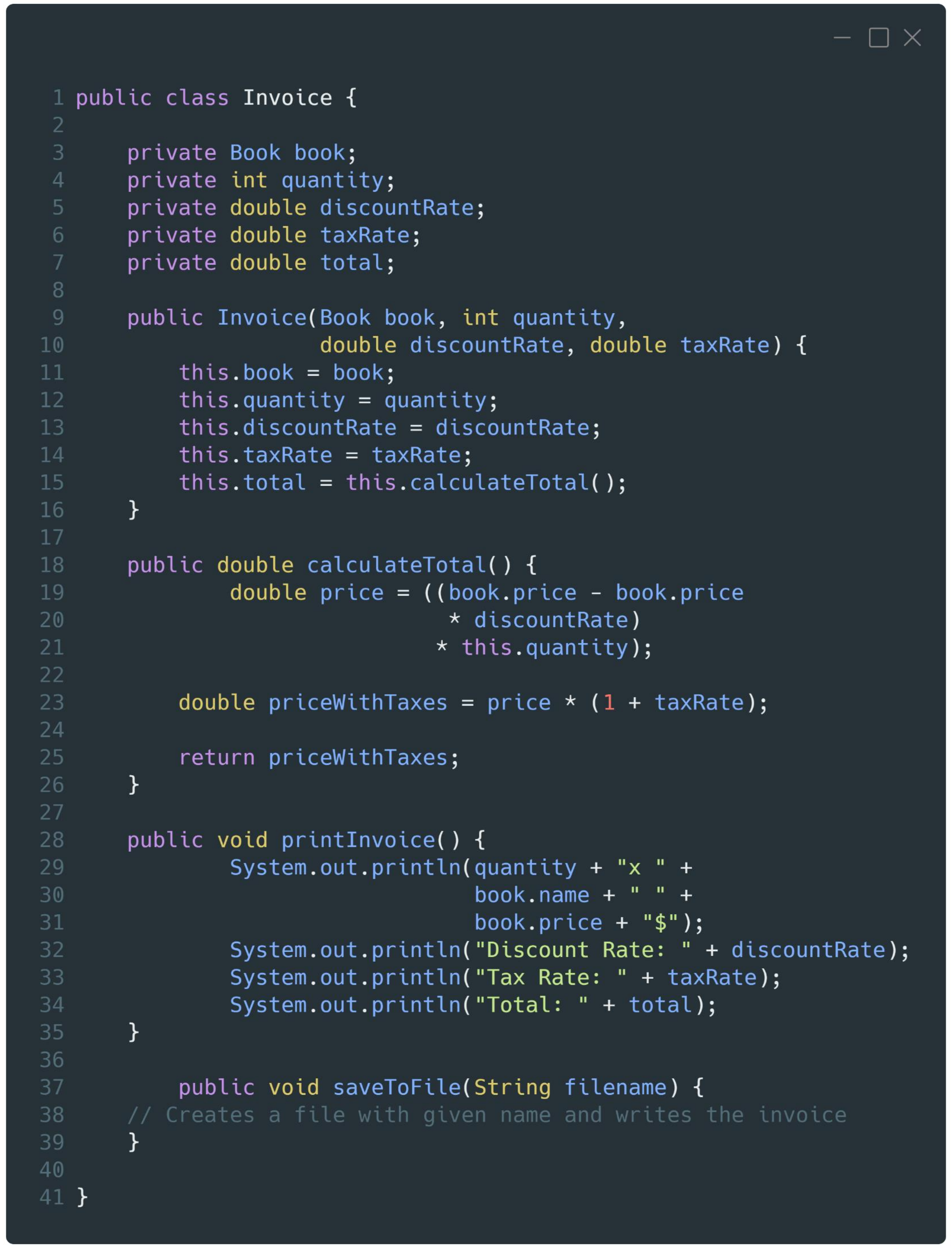

Account.java

```java
public class Account
{
    private Long guid;
    private String accountNumber;
    private String accountName;
    private String status;
    private String type;
}
```

SRP Example 2: Bookstore Invoice Problem

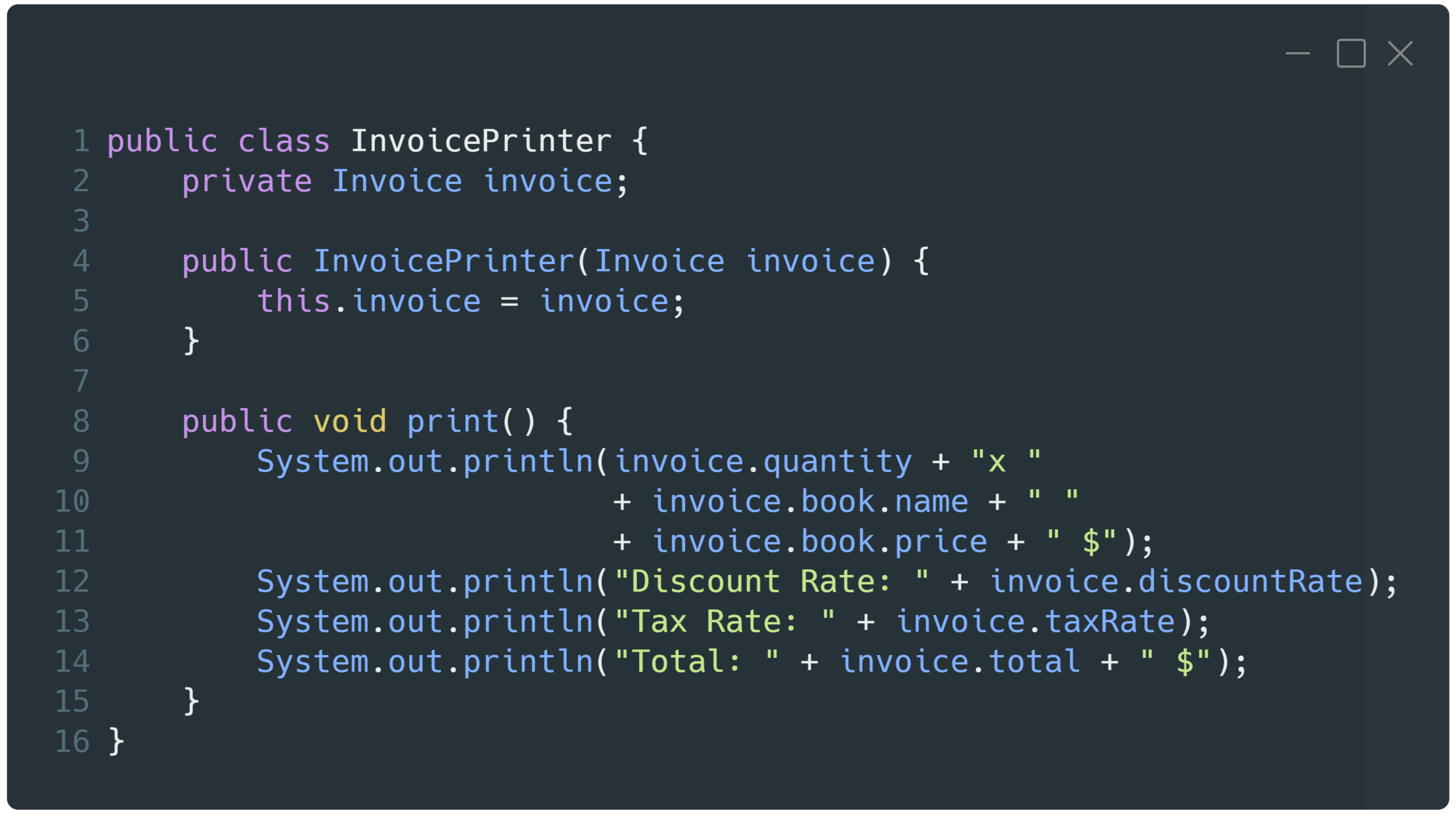


```java
class Book {
    String name;
    String authorName;
    int year;
    int price;
    String isbn;

    public Book(String name,
                String authorName,
                int year, int price,
                String isbn) {
        this.name = name;
        this.authorName = authorName;
        this.year = year;
        this.price = price;
        this.isbn = isbn;
    }
}
```

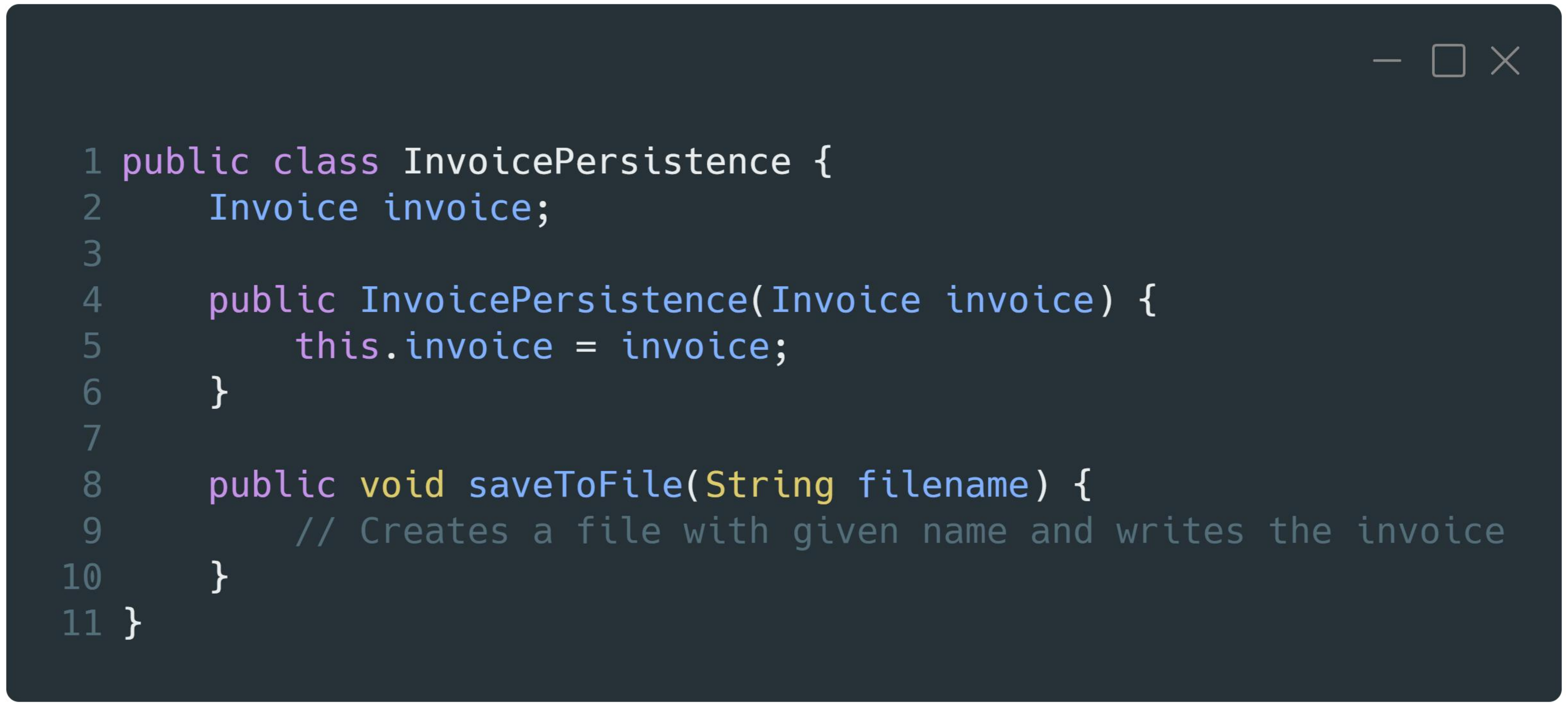


```java
public class Invoice {

    private Book book;
    private int quantity;
    private double discountRate;
    private double taxRate;
    private double total;

    public Invoice(Book book, int quantity,
                   double discountRate, double taxRate) {
        this.book = book;
        this.quantity = quantity;
        this.discountRate = discountRate;
        this.taxRate = taxRate;
        this.total = this.calculateTotal();
    }

    public double calculateTotal() {
            double price = ((book.price - book.price
                             * discountRate)
                            * this.quantity);

        double priceWithTaxes = price * (1 + taxRate);

        return priceWithTaxes;
    }

    public void printInvoice() {
            System.out.println(quantity + "x " +
                               book.name + " " +
                               book.price + "$");
            System.out.println("Discount Rate: " + discountRate);
            System.out.println("Tax Rate: " + taxRate);
            System.out.println("Total: " + total);
    }

        public void saveToFile(String filename) {
    // Creates a file with given name and writes the invoice
    }

}
```

SRP Example 2: Solution



```java
public class InvoicePrinter {
    private Invoice invoice;

    public InvoicePrinter(Invoice invoice) {
        this.invoice = invoice;
    }

    public void print() {
        System.out.println(invoice.quantity + "x "
                           + invoice.book.name + " "
                           + invoice.book.price + " $");
        System.out.println("Discount Rate: " + invoice.discountRate);
        System.out.println("Tax Rate: " + invoice.taxRate);
        System.out.println("Total: " + invoice.total + " $");
    }
}
```



```java
public class InvoicePersistence {
    Invoice invoice;

    public InvoicePersistence(Invoice invoice) {
        this.invoice = invoice;
    }

    public void saveToFile(String filename) {
        // Creates a file with given name and writes the invoice
    }
}
```

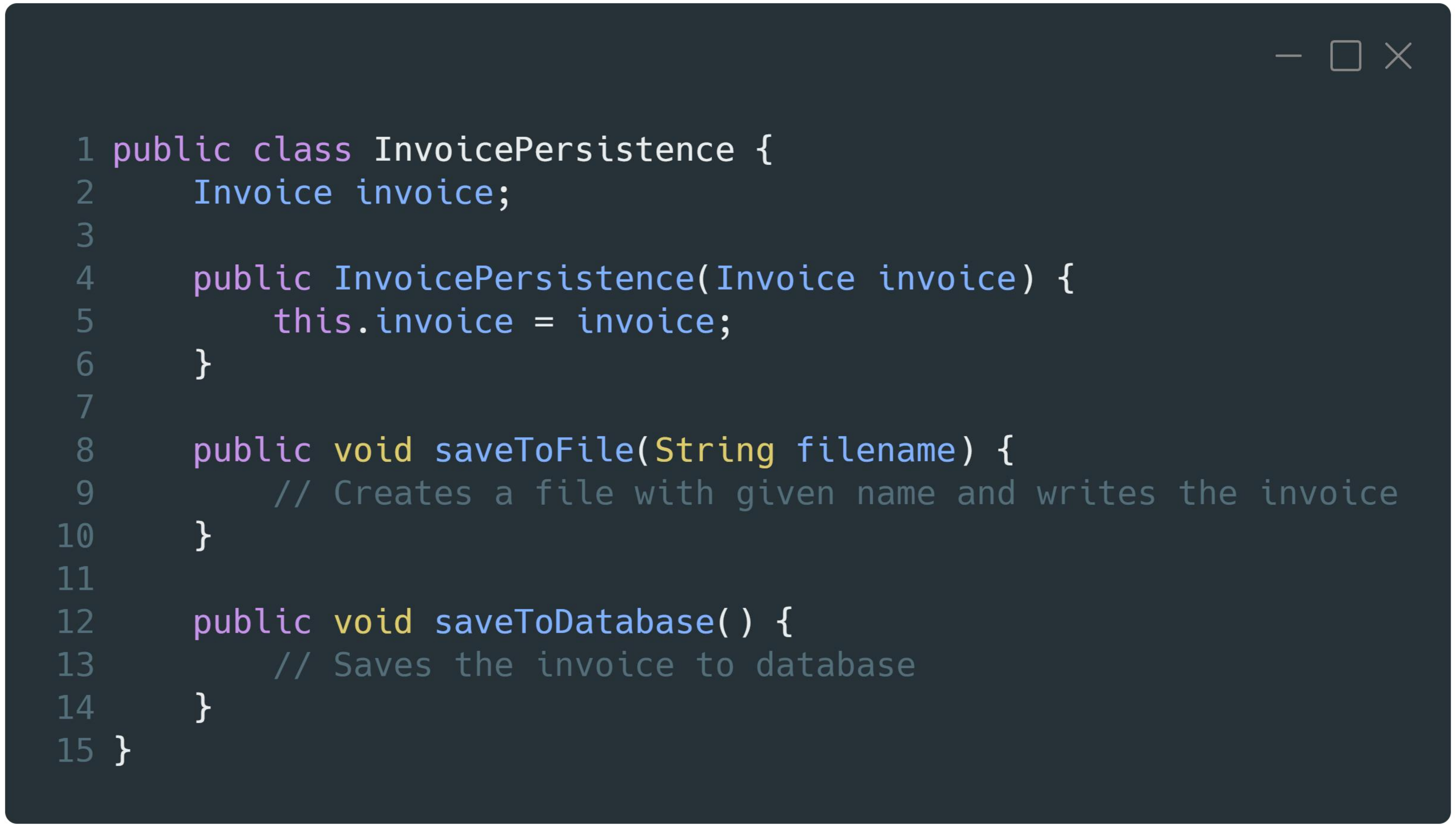
Open-Closed Principle

- **Component**
  - Can be anything from a single class to an entire program
- **Modification:**
  - Changing the code
- **Extension:**
  - Adding new functionality
- **OCP is usually done with the help of interfaces and abstract classes.**
- **How to make sure your code follows the Open/Closed Design Principle?**
  - Implementation inheritance
  - Interface inheritance

"Software components should be open for extension, but closed for modification"

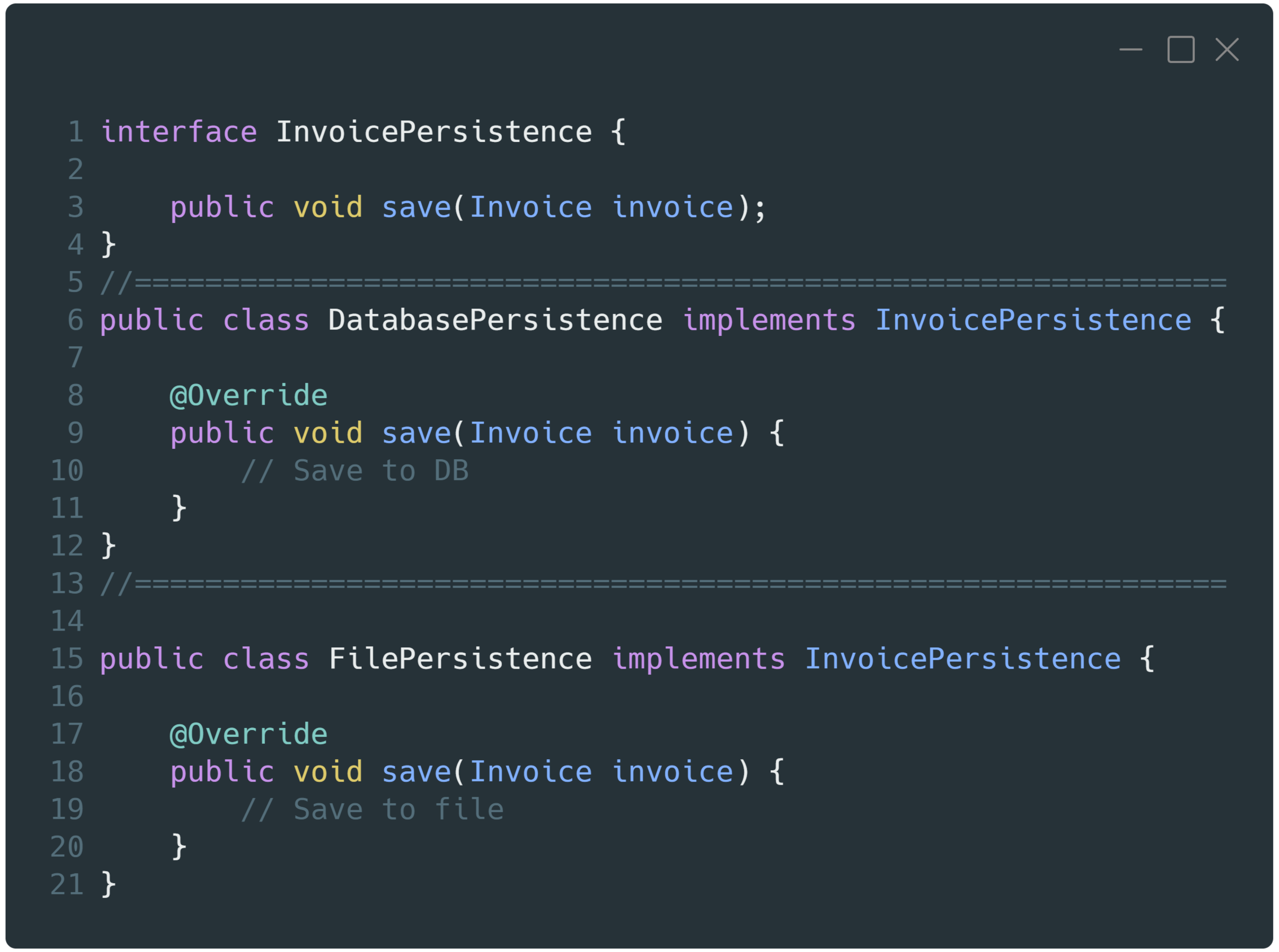
OCP Example: Bookstore Invoice Problem

- Save the invoices to a database
- So we can search them easily.

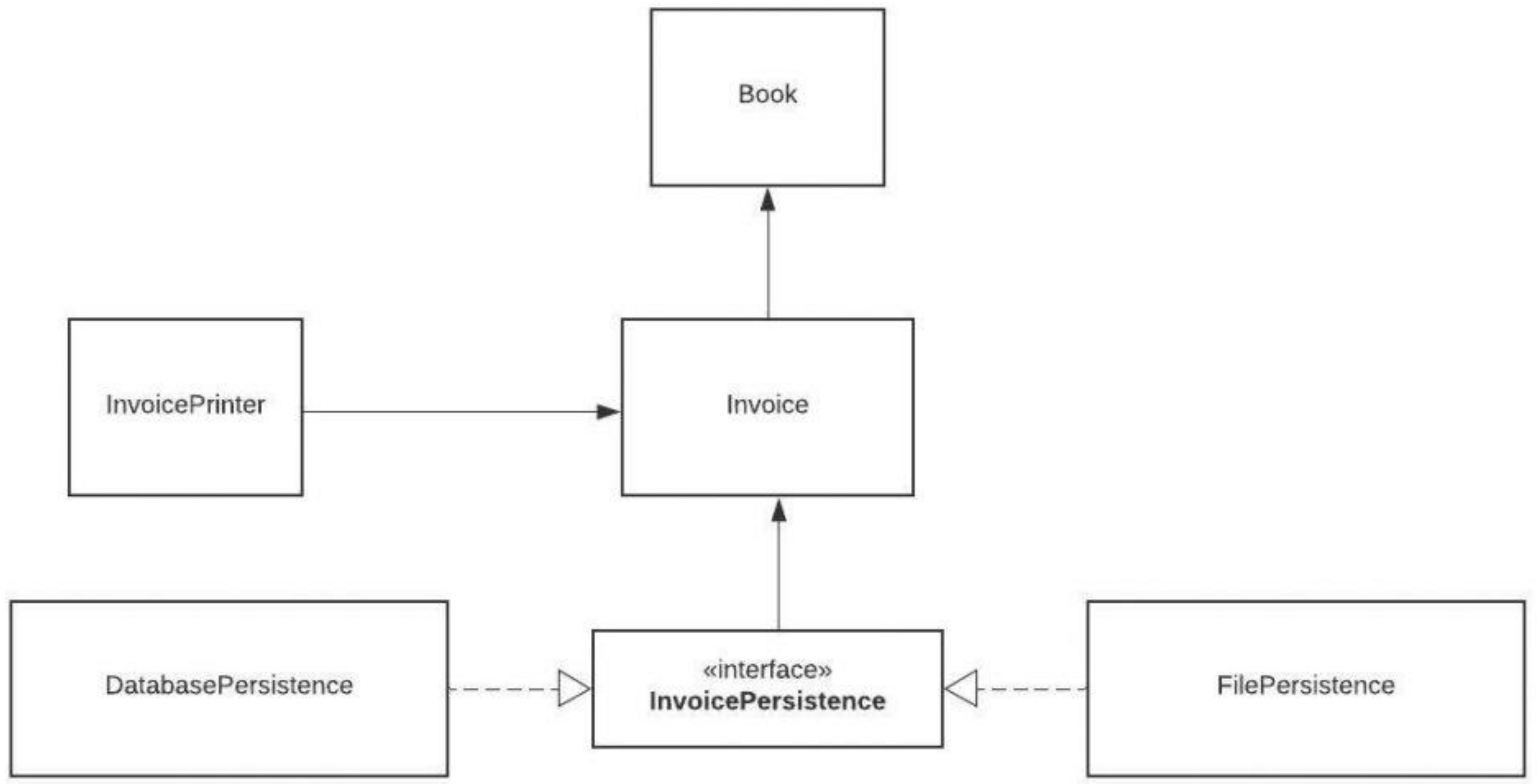


```java
public class InvoicePersistence {
    Invoice invoice;

    public InvoicePersistence(Invoice invoice) {
        this.invoice = invoice;
    }

    public void saveToFile(String filename) {
        // Creates a file with given name and writes the invoice
    }

    public void saveToDatabase() {
        // Saves the invoice to database
    }
}
```

OCP Example: Bookstore Invoice Solution



```java
interface InvoicePersistence {

    public void save(Invoice invoice);
}
//===============================================================
public class DatabasePersistence implements InvoicePersistence {

    @Override
    public void save(Invoice invoice) {
        // Save to DB
    }
}
//===============================================================

public class FilePersistence implements InvoicePersistence {

    @Override
    public void save(Invoice invoice) {
        // Save to file
    }
}
```



Any Question

????????????????

How do you feel about the course?

Please Send Your Question or Feedback...

Top

New

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**