Principles of Object Oriented Class Design

Prof. Jefersson Alex dos Santos

Based and adapted from: http://homepages.dcc.ufmg.br/~fernando/classes/dcc052/ementa/lesson10/

Introduction

- Which factors compromise the maintainability of software?

Introduction

- Which factors compromise the maintainability of software?
  - Rigidity: is the tendency for software to be difficult to change, even in simple ways.

Introduction

- Which factors compromise the maintainability of software?
  - Rigidity: is the tendency for software to be difficult to change, even in simple ways.
  - Fragility: is the tendency of the software to break in many places every time it is changed.

Introduction

- Which factors compromise the maintainability of software?
  - Rigidity: is the tendency for software to be difficult to change, even in simple ways.
  - Fragility: is the tendency of the software to break in many places every time it is changed.
  - Immobility: software that is hard to reuse.

Introduction

- Which factors compromise the maintainability of software?
  - Rigidity: is the tendency for software to be difficult to change, even in simple ways.
  - Fragility: is the tendency of the software to break in many places every time it is changed.
  - Immobility: software that is hard to reuse.
  - Viscosity: simple changes that are hard to maintain.

Introduction

- Which factors compromise the maintainability

These are principles that help to manage the dependencies between modules.

k

  - Immobility: software that is hard to reuse.
  - Viscosity: simple changes that are hard to maintain.

Concepts

1. The Open-Closed Principle (OCP)
2. The Liskov Substitution Principle (LSP)
3. The Dependency Inversion Principle (DIP)
4. The Interface Segregation Principle (ISP)

Concepts

1. **The Open-Closed Principle (OCP)**
2. The Liskov Substitution Principle (LSP)
3. The Dependency Inversion Principle (DIP)
4. The Interface Segregation Principle (ISP)

Open-Closed Principle (OCP)

- We should be able to extend our modules without having to modify them.

Open-Closed Principle (OCP)

```
class Employee {
	private int hours;

	public Employee(int hours) {
		this.hours = hours;
	}

	public int getHours() {
		return hours;
	}
}

class Manager extends Employee {
	private int numSubordinates;

	public Manager(int hours, int numSubs) {
		super(hours);
		numSubordinates = numSubs;
	}

	public int getNumSubs() {
		return numSubordinates;
	}
}

class Guard extends Employee {
	private double dangerFactor;

	public Guard(int hours, double df) {
		super(hours);
		dangerFactor = df;
	}

	public double getDangerFactor() {
		return dangerFactor;
	}
}
```

```
public class Payment {

	public double getPay(Employee e) {
		if (e instanceof Manager) {
			return 1000.0 * ((Manager)
e).getNumSubs() + 200.0 * e.getHours();
		} else if (e instanceof Guard) {
			return 800.0 * ((Guard)
e).getDangerFactor() + 160.0 * e.getHours();
		} else {
			return 600.0 * e.getHours();
		}
	}

	public static void main(String args[]) {
	}
}
```

What are the problems with this payment system?

Open-Closed Principle (OCP)

1. How the system would have to be modified to add a new type of Employee? (ex2)

   SuperEmployee, who makes more money!

Open-Closed Principle (OCP)

1. How the system would have to be modified to add a new type of Employee? (ex2)

   SuperEmployee, who makes more money!

2. Why this program fails to be object-oriented?

Open-Closed Principle (OCP)

1. How the system would have to be modified to add a new type of Employee? (ex2)

   SuperEmployee, who makes more money!

2. Why this program fails to be object-oriented?
3. And how to fix it? (ex3)

Open-Closed Principle (OCP)

1. How the system would have to be modified to add a new type of Employee? (ex2)

   SuperEmployee, who makes more money!

2. Why this program fails to be object-oriented?
3. And how to fix it? (ex3)
4. Now, what if we want to ensure that SuperEmployees get payed first? (ex4)

Open-Closed Principle (OCP)

1. How the system would have to be modified to add a new type of Employee? (ex2)

   SuperEmployee, who makes more money!

2. Why this program fails to be object-oriented?
3. And how to fix it? (ex3)
4. Now, what if we want to ensure that SuperEmployees get payed first? (ex4)
5. Is it possible to close the system against ordering? (ex5)

Open-Closed Principle (OCP)

- Encapsulation is one of the key heuristics behind the open-closed principle. Why?
- Consider a variable that we know that is not likely to change:

```
public class Device {
     public boolean status;
}
```

Is there any problem to set it public?

Open-Closed Principle (OCP)

- What about:

```
public class Time {
  int hours; int minutes; int seconds;
}
```

How do global variables compromise the open-closed principle?

Open-Closed Principle (OCP)

- Run-time type interrogation (RTTI) is another source of violations of the OCP principle. Examples?
- But there are also examples of situations when the use of RTTI is safe, i.e, it does not violate the OCP principle. Which examples?
  - - E.g: want to know how many guards we have in the payroll.

Open-Closed Principle (OCP)

- A good heuristic to support the OCP principle is "programming to the interfaces". How so?
- The OCP principle could be re-phrased as "New features are added by adding new code, instead of changing old code".

Concepts

1. The Open-Closed Principle (OCP)
2. **The Liskov Substitution Principle (LSP)**
3. The Dependency Inversion Principle (DIP)
4. The Interface Segregation Principle (ISP)

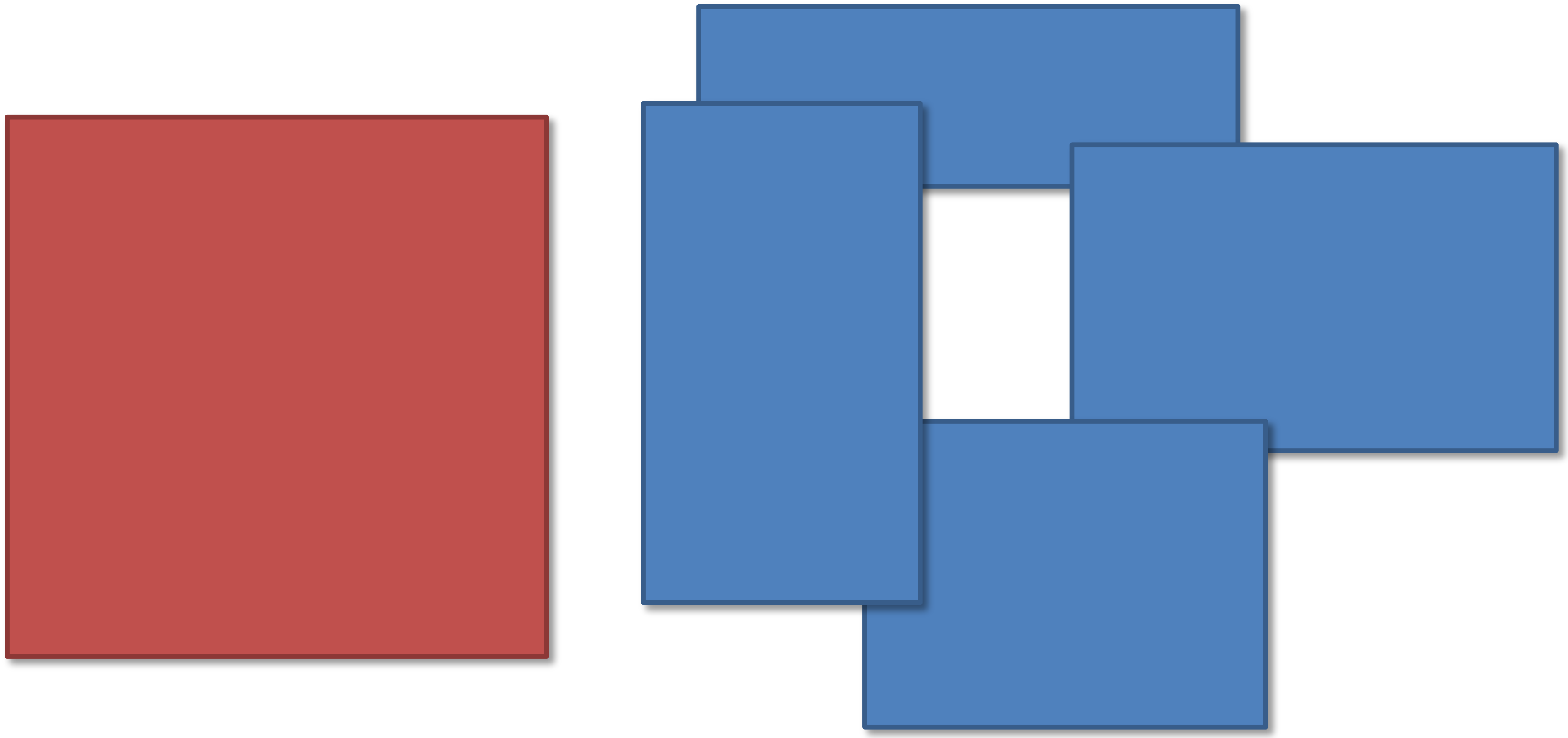
The Liskov Substitution Principle

- Subclasses should be substitutable for their base classes

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

The Liskov Substitution Principle

- See Rectangle.java
- How to reuse this class to implement a Square?



The Liskov Substitution Principle

- How to handle the test in TestRect.java ?
- What is the post-condition of, say, setWidth in Rectangle?

```
public void setWidth(int newW) {
   int oldH = h;
   w = newW;
   assert(w == newW && oldH == h);
}
```

- Is this true for Square? What we need to do?

The Liskov Substitution Principle

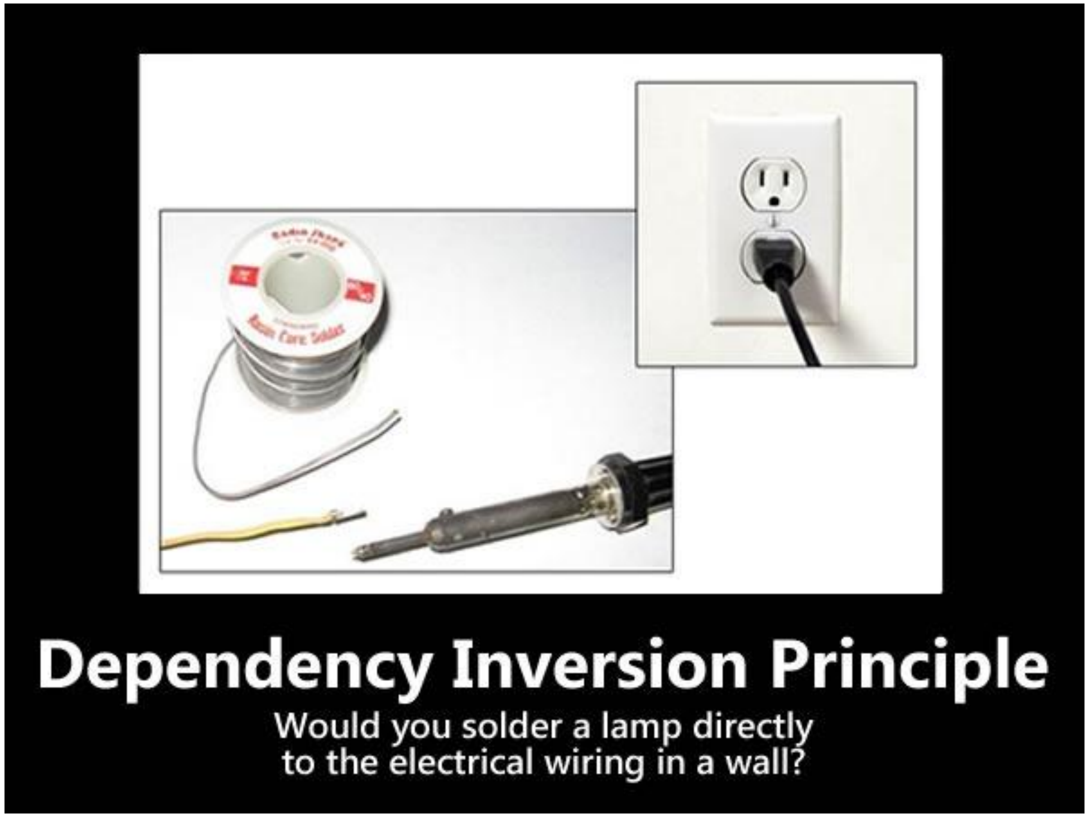
- How does inheritance impact on pre and post conditions?
- When redefining a routine in a subclass:
  - replace a pre-condition by a weaker pre-condition,
  - replace a post-condition by a stronger one.

Concepts

1. The Open-Closed Principle (OCP)
2. The Liskov Substitution Principle (LSP)
3. **The Dependency Inversion Principle (DIP)**
4. The Interface Segregation Principle (ISP)

The Dependency Inversion Principle (DIP)

- Depend on abstractions, not on implementations.


Dependency Inversion Principle
Would you solder a lamp directly
to the electrical wiring in a wall?

The Dependency Inversion Principle (DIP)

- Consider the 'cp' Unix command.

The Dependency Inversion Principle (DIP)

- Consider the 'cp' Unix command.
- What is a good test case for this problem?

The Dependency Inversion Principle (DIP)

- Consider the 'cp' Unix command.
- What is a good test case for this problem?
- What is a reasonable solution?

The Dependency Inversion Principle (DIP)

- Consider the 'cp' Unix command.
- What is a good test case for this problem?
- What is a reasonable solution?
- Is this a good solution?
  - Is this fragile?
  - Is this rigid?
  - Is this hard to reuse?

The Dependency Inversion Principle (DIP)

- Implement a 'wc' UNIX like utility.
- Can you reuse anything from the 'cp' application?
- What is the algorithm implemented by FileCopier?
  - while has data: read data from input; send data to output.

The Dependency Inversion Principle (DIP)

- What is the interface of the output?

```
public interface OutChannel {
   void write(int data) throws Exception;
   void done() throws Exception;
}
```

- How to implement this interface to do a 'cp'?
  - See dip.ex2.FileWriter.java

The Dependency Inversion Principle (DIP)

- Why the finalize is important?
- How is the 'wc' app divided? Which tasks does it execute?
- How is an OutChannel that counts the number of characters in the input file?
  - See dip.ex2. WordCounter.java
- How is an OutChannel that counts the number of words in the input file?
  - See dip.ex2.CharacterCounter.java

The Dependency Inversion Principle (DIP)

- What are good heuristics to see if our programs adhere to the DIP?
  - Few classes on the left side of expressions.
  - Few classes as formal parameters.

The Interface Segregation Principle (ISP)

- Give to each client only the interface that the client needs

Interface Segregation Principle
Tailor your Interfaces to the Client's Specific Requirements

References

1. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf
2. http://www.objectmentor.com/resources/articles/ocp.pdf
3. http://www.objectmentor.com/resources/articles/lsp.pdf
4. http://www.objectmentor.com/resources/articles/isp.pdf
5. http://www.objectmentor.com/resources/articles/dip.pdf