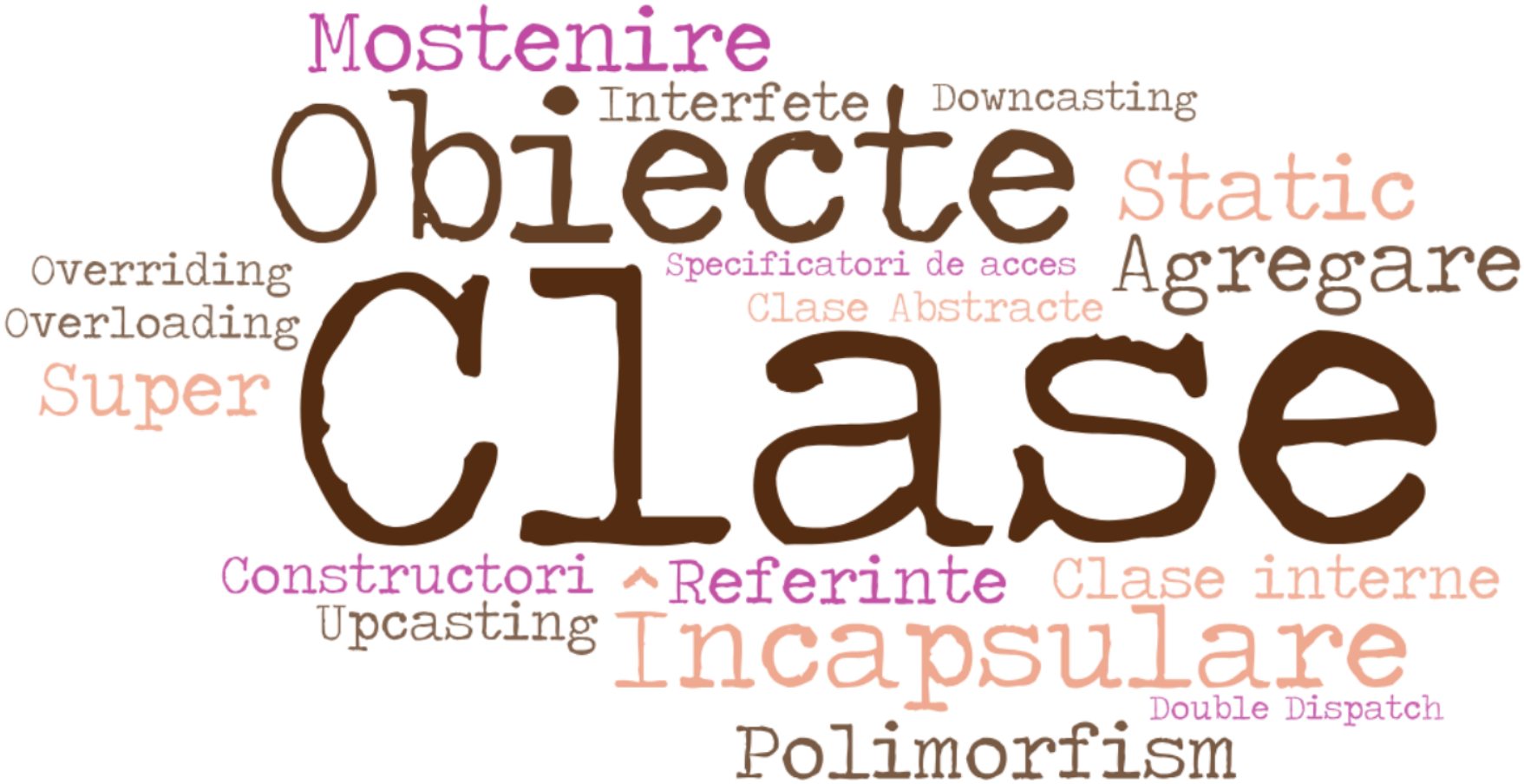
Object-Oriented Design Principles

Alexandru Olteanu

Universitatea Politehnica Bucuresti
Facultatea de Automatică si Calculatoare, Departamentul Calculatoare
alexandru.olteanu@upb.ro

OOP, 2020

Partea 1: Bazele teoretice ale OOP



SOLID

SOLID

Five design principles synthesized by Robert C. Martin (Uncle Bob):

- SRP: Single Responsibility Principle
- OCP: Open-Closed Principle
- LSP: Liskov Substitution Principle
- ISP: Interface Segregation Principle
- DIP: Dependency Inversion Principle

Single-responsiblity principle

Definition

A class should have only one responsibility, meaning that a class should have one and only one reason to change.

▸ Robert C. Martin "The Single-Responsibility Principle"

e.g. a Rectangle class should not contain both draw and area functions

Single-responsiblity principle

▸ SOLID principles explained with motivational posters

Open-closed principle

Definition

You should be able to extend a classes behavior, without modifying it.

▸ Robert C. Martin "The Open-Closed Principle", C++ Report, January 1996

e.g. having an AbstractShape, with a generic draw method, allows us to add new shapes

Do not confuse the "extend" mentioned by the principle with extends in Java

Open-closed principle

Definition

Software entities should be open for extension, but closed for modification.
Bertrand Mayer "Object-Oriented Software Construction", 1988

extending the software entities can be done if they use inheritance

Open-closed principle

▸ SOLID principles explained with motivational posters

Liskov substitution principle

Definition

If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T.

- Liskov, Barbara. "Keynote address-data abstraction and hierarchy." ACM Sigplan Notices 23.5 (1988): 17-34

essentially polymorphism

Liskov substitution principle

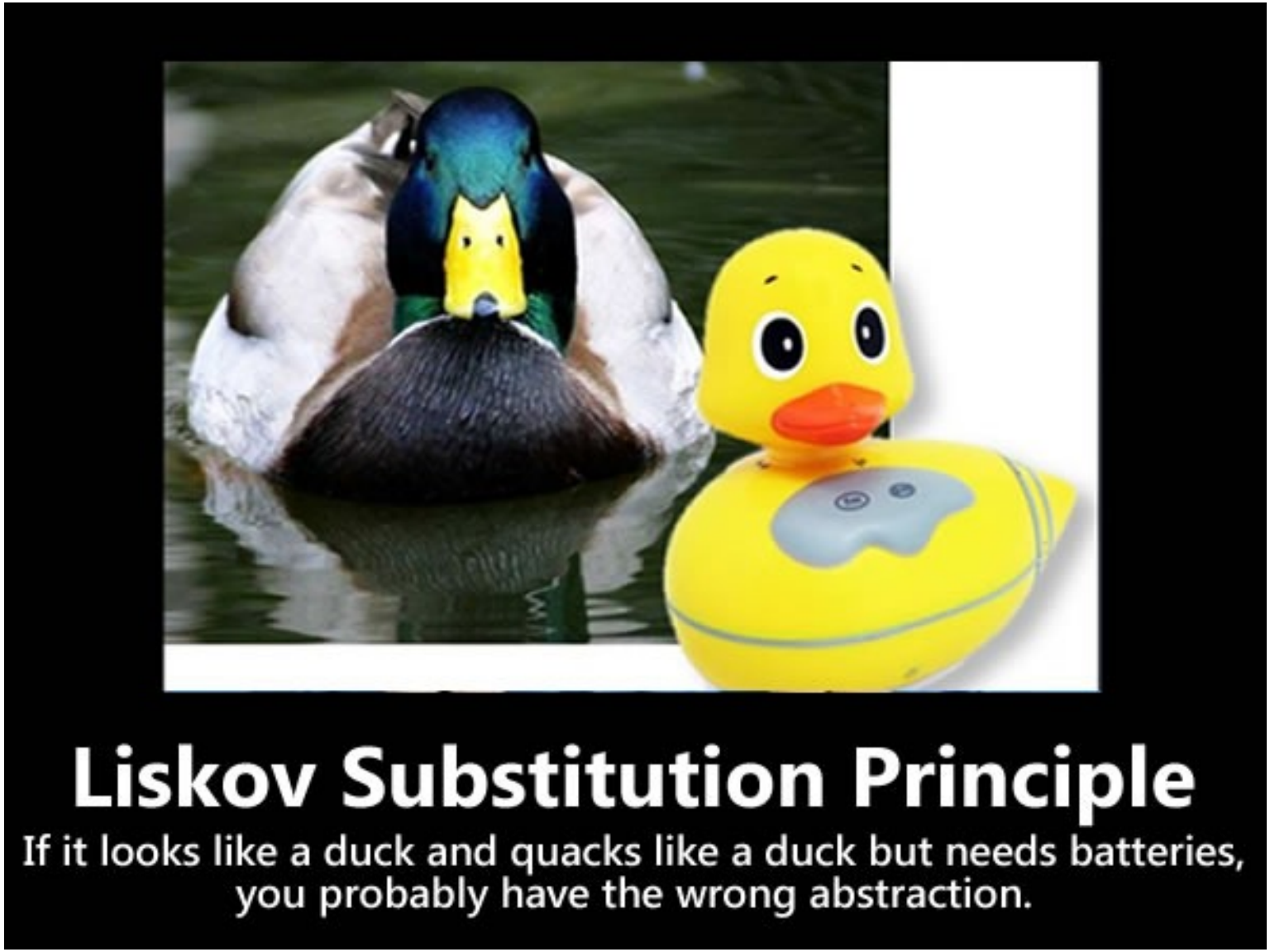
Definition

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

▸ Robert C. Martin "The Liskov substitution principle", C++ Report

e.g. a square might be a rectangle, but the behavior of a Square object is not consistent with the behavior of a Rectangle object

Liskov substitution principle



▸ SOLID principles explained with motivational posters

LSP vs OCP

Follows OCP, but not LSP:

```
public interface IPerson {}

public class Boss implements IPerson {
    public void doBossStuff() { ... }
}

public class Peon implements IPerson {
    public void doPeonStuff() { ... }
}

public class Context {
    public Collection<IPerson> getPersons() { ... }
}
```

▸ LSP vs OCP on StackExchange

LSP vs OCP

Follows OCP, but not LSP - problem:

```
// in some routine that needs to do stuff with
// a collection of IPerson:
Collection<IPerson> persons = context.getPersons();
for (IPerson person : persons) {
    // now we have to check the type... :-P
    if (person instanceof Boss) {
        ((Boss) person).doBossStuff();
    }
    else if (person instanceof Peon) {
        ((Peon) person).doPeonStuff();
    }
}
```

▸ LSP vs OCP on StackExchange

LSP vs OCP

Follows OCP, but not LSP - fix:

```
public class Boss implements IPerson {
    // we're adding this general method
    public void doStuff() {
        // that does the call instead
        this.doBossStuff();
    }
    public void doBossStuff() { ... }
}


public interface IPerson {
    // pulled up method from Boss
    public void doStuff();
}

// do the same for Peon
```

▸ LSP vs OCP on StackExchange

LSP vs OCP

Follows LSP, but not OCP:

```
public class LiskovBase {
    public void doStuff() {
        System.out.println("My name is Liskov");
    }
}
public class LiskovSub extends LiskovBase {
    public void doStuff() {
        System.out.println("I'm a sub Liskov!");
    }
}

public class Context {
    private LiskovBase base;
    public void doLiskovyStuff() {
        base.doStuff();
    }
    public void setBase(LiskovBase base) {
      this.base = base
    }
}
```

LSP vs OCP

Follows LSP, but not OCP - fix:

```
public class LiskovBase {
    // the code that was duplicated is now a template method
    public final void doStuff() {
        System.out.println(getStuffString());
    }

    // the code that "varies" in LiskovBase and it's
    // subclasses called by the template method above
    // we expect it to be virtual and overridden
    public string getStuffString() {
        return "My name is Liskov";
    }
}

public class LiskovSub extends LiskovBase {
    // the actual code that varied
    public string getStuffString() {
        return "I'm sub Liskov!";
    }
}
```

Interface segregation principle

Definition

many client-specific interfaces are better than one general-purpose interface.

▸ Robert C. Martin "The Interface segregation principle"

e.g. having a TimedDoor class, shouldn't require all Doors to have a Timer

Interface segregation principle

▸ SOLID principles explained with motivational posters

Dependency Inversion Principle

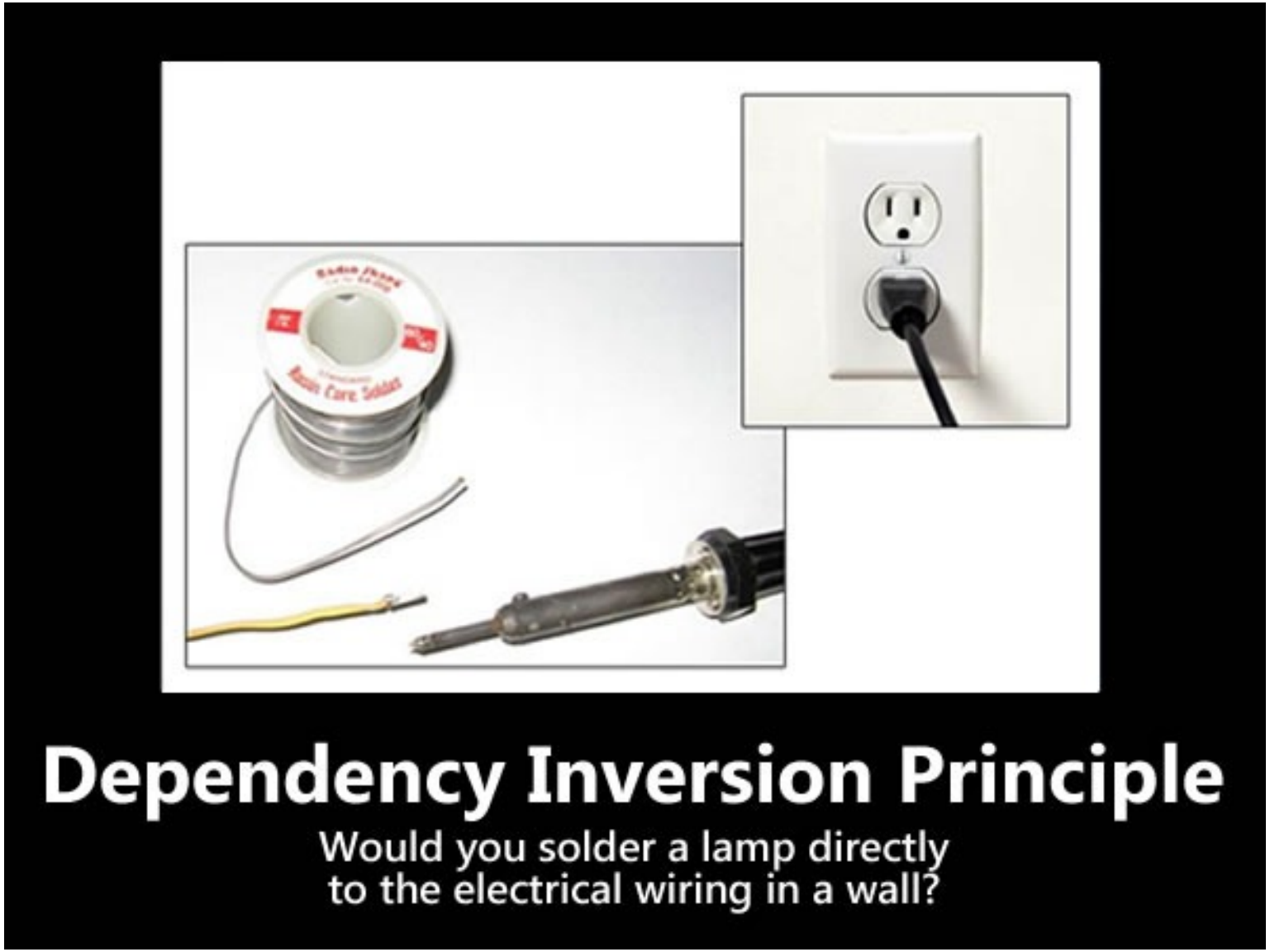
Definition

High level modules should not depend upon low level modules. Both should depend upon abstractions.

▸ Robert C. Martin "The Dependency Inversion Principle", C++ Report

e.g. a program that copies from inputs to outputs, should not depend on the input being a keyboard and the output being a screen
e.g. a PasswordRecovery should not depend on a MySQLConnection, but on a generic DBConnectionInterface

Dependency Inversion Principle



▸ SOLID principles explained with motivational posters

SOLID Tutorials

Acelasi cod, dar in limbaje diferite:

- SOLID in PHP (class inheritance, ca in Java)
- SOLID in JavaScript (prototype-based language)

De citit

Despre SOLID si alte principii de Design Orientat Obiect

- Uncle Bob's Principles of OOD
- Principles Of Object Oriented Design
- KISS, YAGNI, DRY: 3 Principles to Simplify Your Life as a Developer