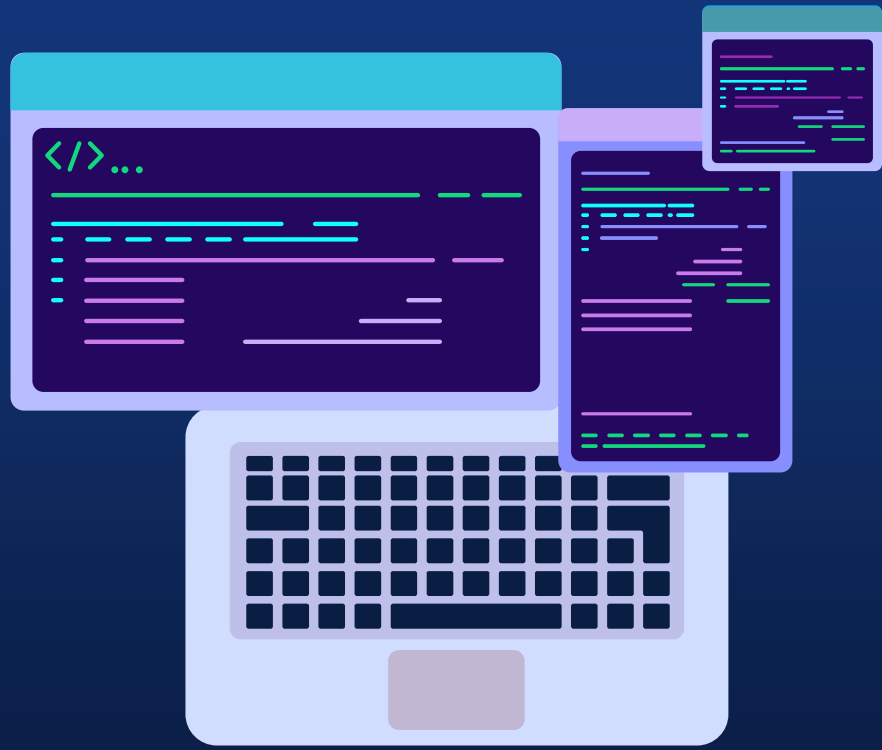
Software Design and Important concepts



Mentor: MSc. Einar Rocha

CONTENT

01
OOP Pillars

Inheritance, Polymorphism
Encapsulation, Abstraction

02
Clean Code

Meaningful Names,
Functions, Unit test
Code Smells...

03
SOLID

Single Responsiblity
Open closed
Liskov Substitution
Interface Segregation
Dependency Inversion

04
Design patterns

Singleton, Factory Method
Strategy, Observer
Builder...

04

Design patterns

Agenda

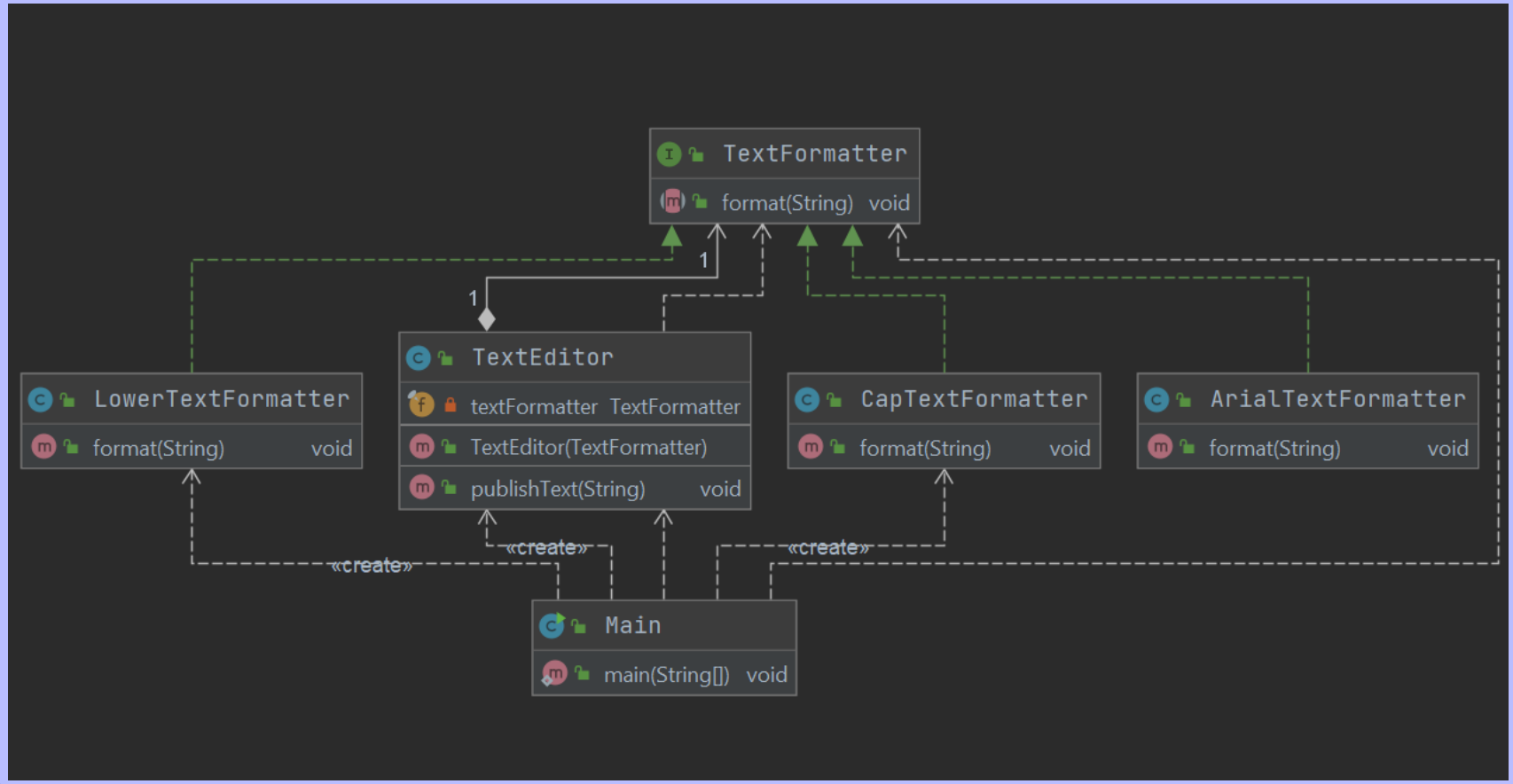
- Strategy — Example...
- Visitor — Example...

Strategy

When to use?

- Many related classes differ only in their behavior.
- You need different variants of an algorithm.
- An algorithm uses data that clients shouldn’t know about.
- A class defines many behaviors, and these appear as multiple conditional statements in its operations.

Strategy


TextFormatter
format(String) void
1
1
TextEditor
textFormatter TextFormatter
TextEditor(TextFormatter)
publishText(String) void
LowerTextFormatter
format(String) void
CapTextFormatter
format(String) void
ArialTextFormatter
format(String) void
«create»
«create»
«create»
Main
main(String[]) void

CONS

If you only have a couple of algorithms and they rarely change, there's no real reason to overcomplicate the program

Clients must be aware of the differences between strategies to be able to select a proper one.

A lot of modern programming languages have functional type support that lets you implement different versions of an algorithm inside a set of anonymous functions

PROS

You can swap algorithms used inside an object at runtime.

You can isolate the implementation details of an algorithm from the code that uses it.

You can replace inheritance with composition.

Open/Closed Principle.

Visitor

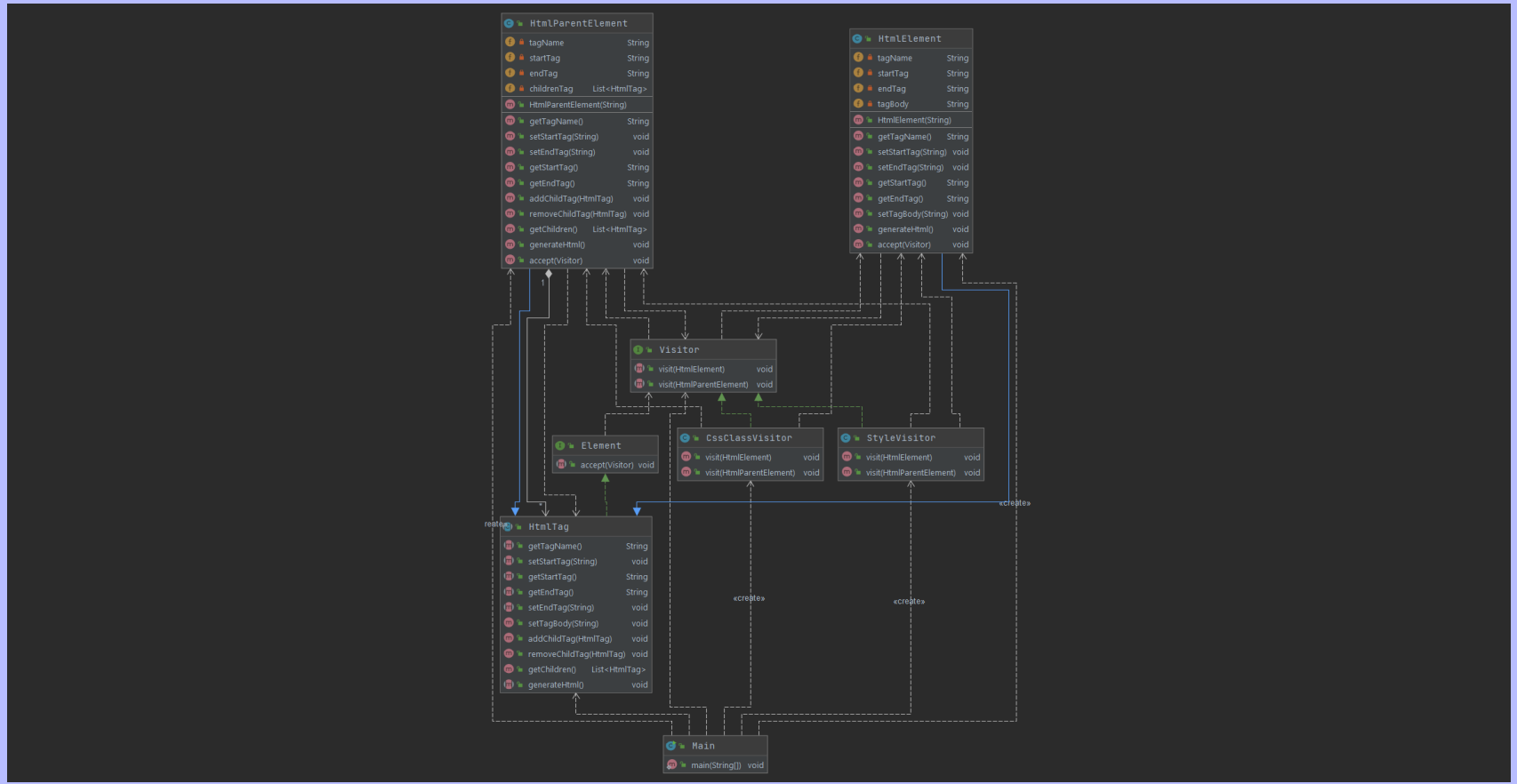
When to use?

- An object structure contains many classes of objects with differing interfaces, and you want to perform operations on these objects that depend on their concrete classes.

- Many distinct and unrelated operations need to be performed on objects in an object structure, and you want to avoid "polluting" their classes with these operations.

- The classes defining the object structure rarely change, but you often want to define new operations over the structure.

Visitor


HtmlParentElement
tagName String
startTag String
endTag String
childrenTag List<HtmlTag>
HtmlParentElement(String)
getTagName() String
setStartTag(String) void
setEndTag(String) void
getStartTag() String
getEndTag() String
addChildTag(HtmlTag) void
removeChildTag(HtmlTag) void
getChildren() List<HtmlTag>
generateHtml() void
accept(Visitor) void
HtmlElement
tagName String
startTag String
endTag String
tagBody String
HtmlElement(String)
getTagName() String
setStartTag(String) void
setEndTag(String) void
getStartTag() String
getEndTag() String
setTagBody(String) void
generateHtml() void
accept(Visitor) void
Visitor
visit(HtmlElement) void
visit(HtmlParentElement) void
Element
accept(Visitor) void
CssClassVisitor
visit(HtmlElement) void
visit(HtmlParentElement) void
StyleVisitor
visit(HtmlElement) void
visit(HtmlParentElement) void
«create»
«create»
«create»
HtmlTag
getTagName() String
setStartTag(String) void
getStartTag() String
getEndTag() String
setEndTag(String) void
setTagBody(String) void
addChildTag(HtmlTag) void
removeChildTag(HtmlTag) void
getChildren() List<HtmlTag>
generateHtml() void
Main
main(String[]) void

CONS

You need to update all visitors each time a class gets added to or removed from the element hierarchy.

Visitors might lack the necessary access to the private fields and methods of the elements that they're supposed to work with.

PROS

Open/Closed Principle.

Single Responsibility Principle.

A visitor object can accumulate some useful information while working with various objects.

THANKS

Do you have any questions?
rochaeinar@outlook.com