Inatel
Instituto Nacional de Telecomunicações

Object-oriented Software Design Patterns

Concepts and Examples

Marcelo Vinícius Cysneiros Aragão
marcelovca90@inatel.br

Topics

- What are design patterns?
- Benefits of using design patterns
- Categories and Examples
  - Creational Design Patterns
  - Structural Design Patterns
  - Behavioral Design Patterns
- AntiPatterns
- References

What are design patterns?

- In software engineering, a **design pattern** is a general reusable solution to a commonly occurring problem within a given context in software design.
- A design pattern is not a finished design that can be transformed directly into source or machine code.
- It is a description or template for how to solve a problem that can be used in many different situations.
- Patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system

A DESIGN PATTERN IS A WELL DESCRIBED SOLUTION TO A COMMON SOFTWARE PROBLEM

Benefits of using design patterns

- Design Patterns are already defined and provides **industry standard approach** to solve a recurring problem, so it saves time if we sensibly use the design pattern.
- Using design patterns promotes **reusability** that leads to more **robust** and highly maintainable code. It helps in reducing total cost of ownership (TCO) of the software product.
- Since design patterns are already defined, it makes our code easy to understand and debug. It leads to faster development and new members of team understand it easily.

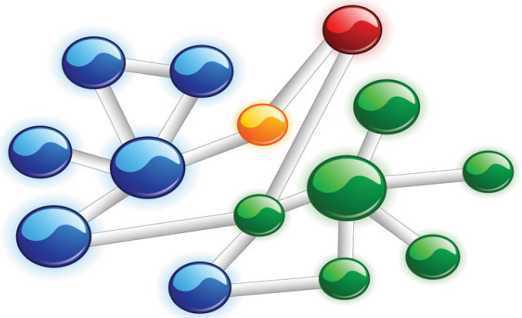
Categories

Creational Design Patterns

Creational design patterns provide solution to instantiate a object in the best possible way for specific situations.

Structural Design Patterns

Structural patterns provide different ways to create a class structure, for example using inheritance and composition to create a large object from small objects.



Behavioral Design Patterns

Behavioral patterns provide solution for the better interaction between objects and how to provide lose coupling and flexibility to extend easily.

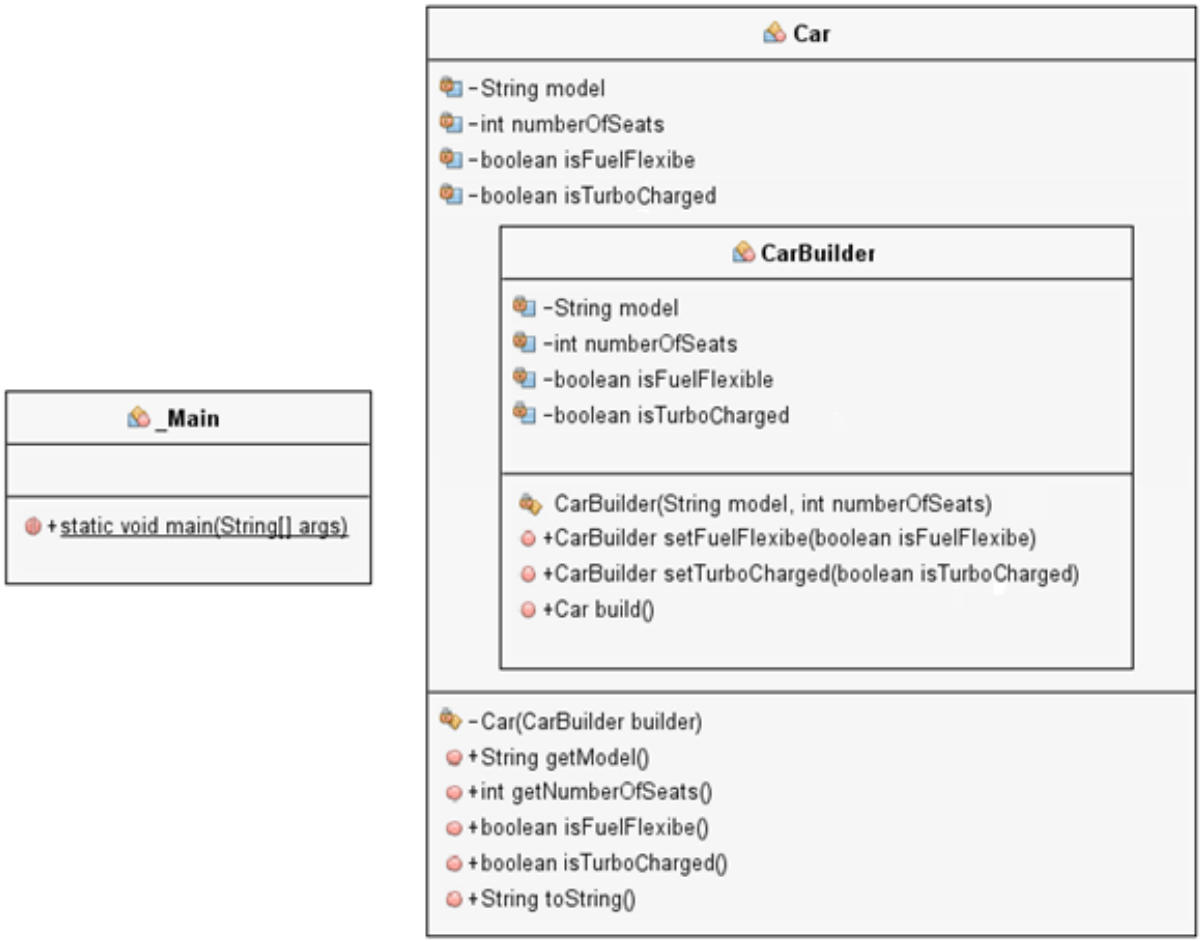
Creational Design Patterns

- These design patterns are all about Class instantiation.
- This pattern can be further divided into class-creation patterns and object-creational patterns.
- While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

Abstract Factory	Creates an instance of several families of classes
Builder	Separates object construction from its representation
Factory Method	Creates an instance of several derived classes
Object Pool	Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
Prototype	A fully initialized instance to be copied or cloned
Singleton	A class of which only a single instance can exist

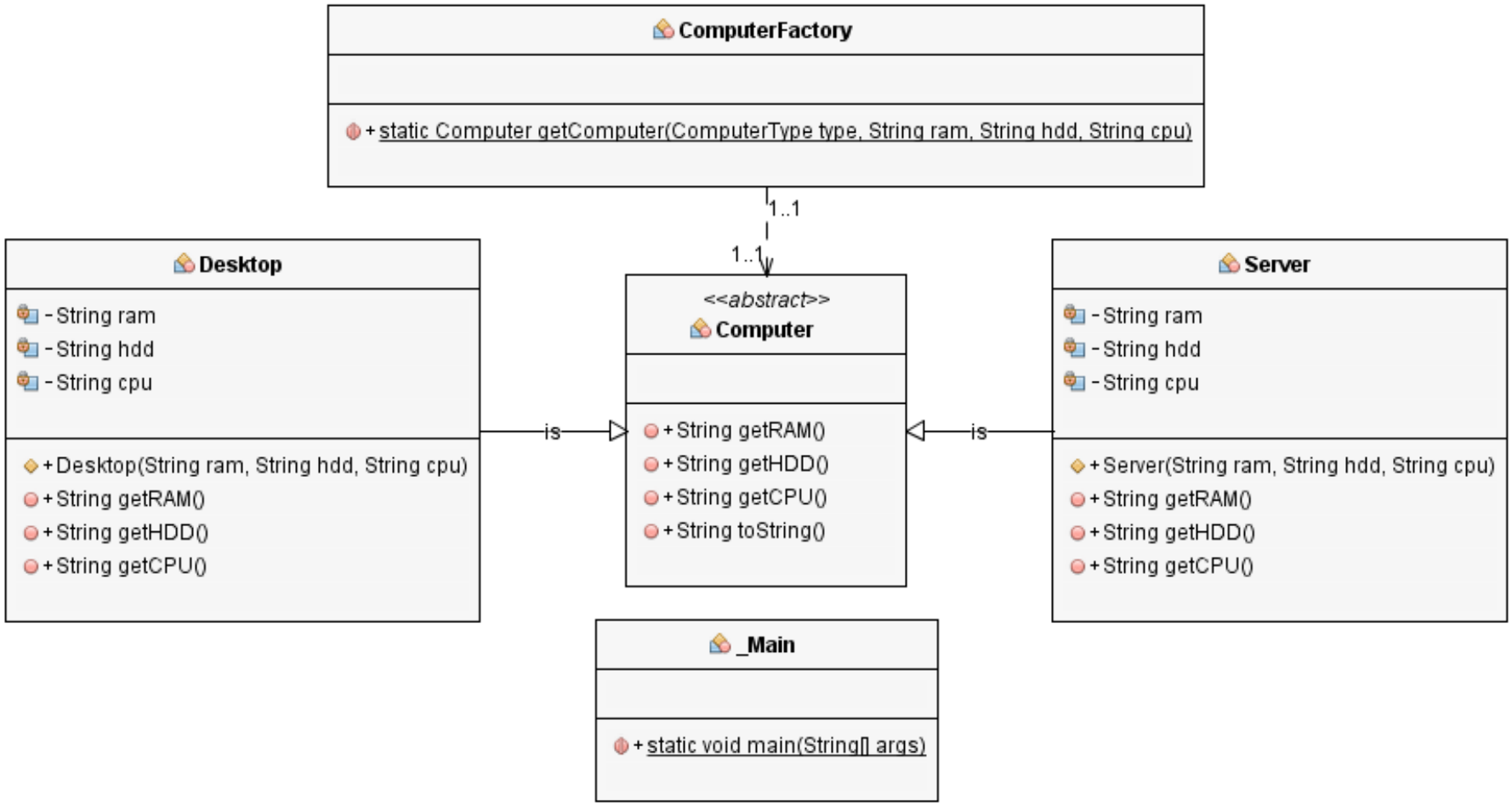
Creational Design Pattern Example: Builder

- Separates object construction from its representation



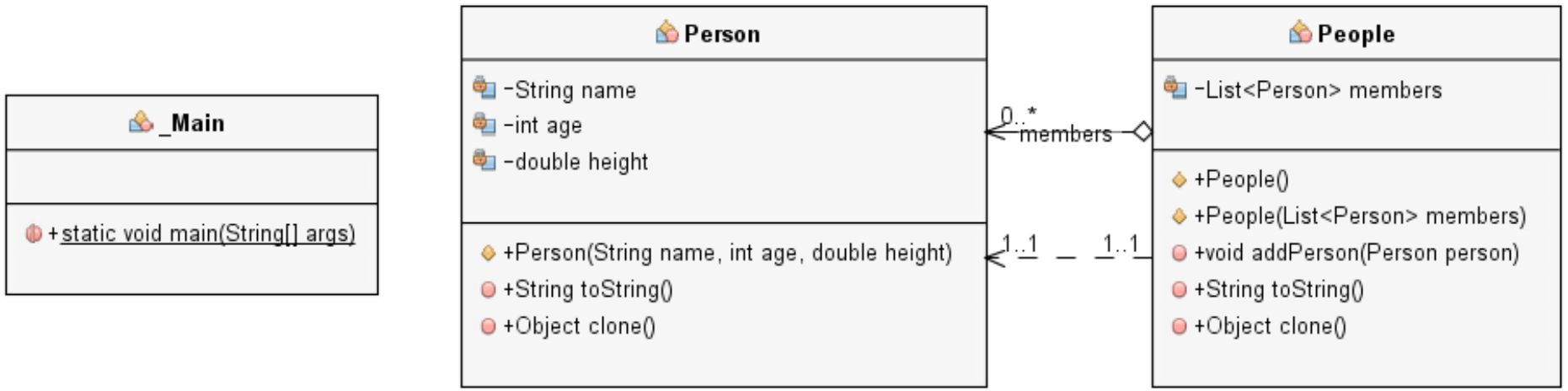
Creational Design Pattern Example: Factory Method

- Creates an instance of several derived classes



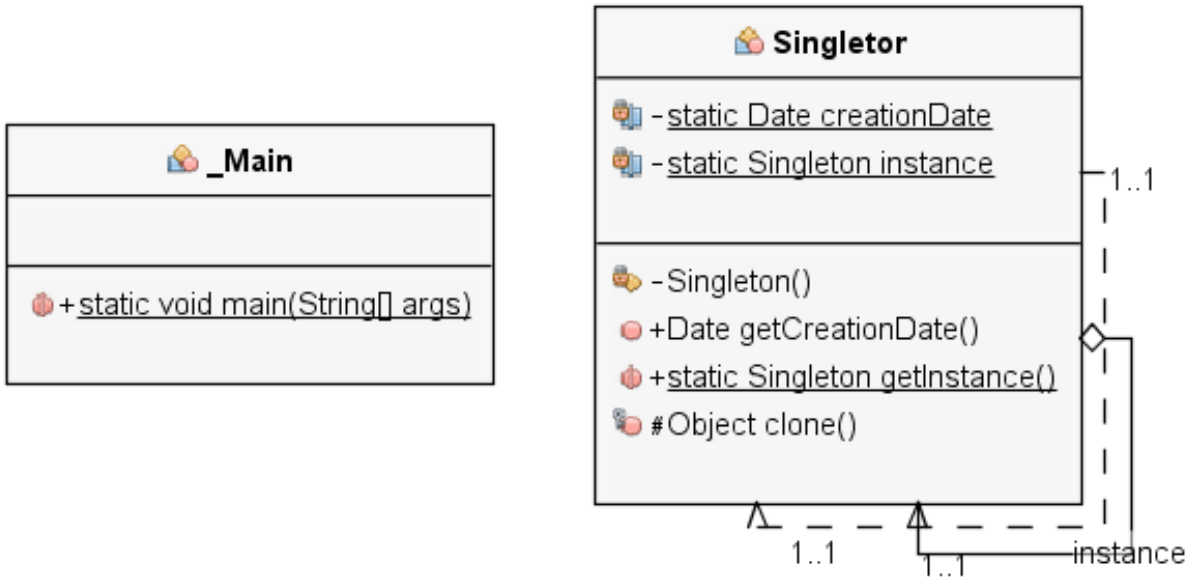
Creational Design Pattern Example: Prototype

- A fully initialized instance to be copied or cloned



Creational Design Pattern Example: Singleton

- A class of which only a single instance can exist



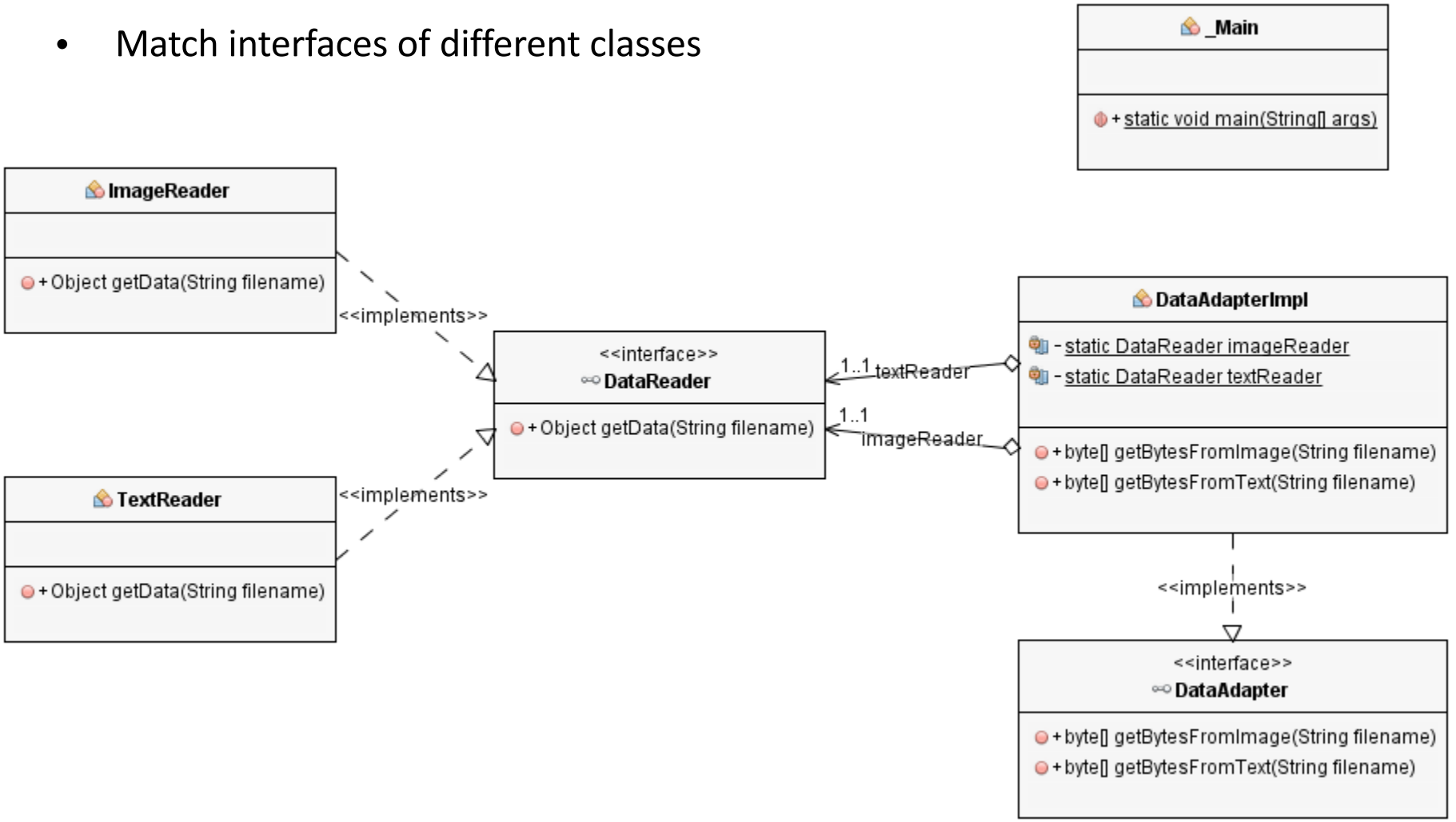
Structural Design Patterns

- These design patterns are all about Class and Object composition.
- Structural class-creation patterns use inheritance to compose interfaces.
- Structural object-patterns define ways to compose objects to obtain new functionality.

Adapter	Match interfaces of different classes
Bridge	Separates an object's interface from its implementation
Composite	A tree structure of simple and composite objects
Decorator	Add responsibilities to objects dynamically
Facade	A single class that represents an entire subsystem
Flyweight	A fine-grained instance used for efficient sharing
Private Class Data	Restricts accessor/mutator access
Proxy	Provide a surrogate or placeholder for another object to control access to it.

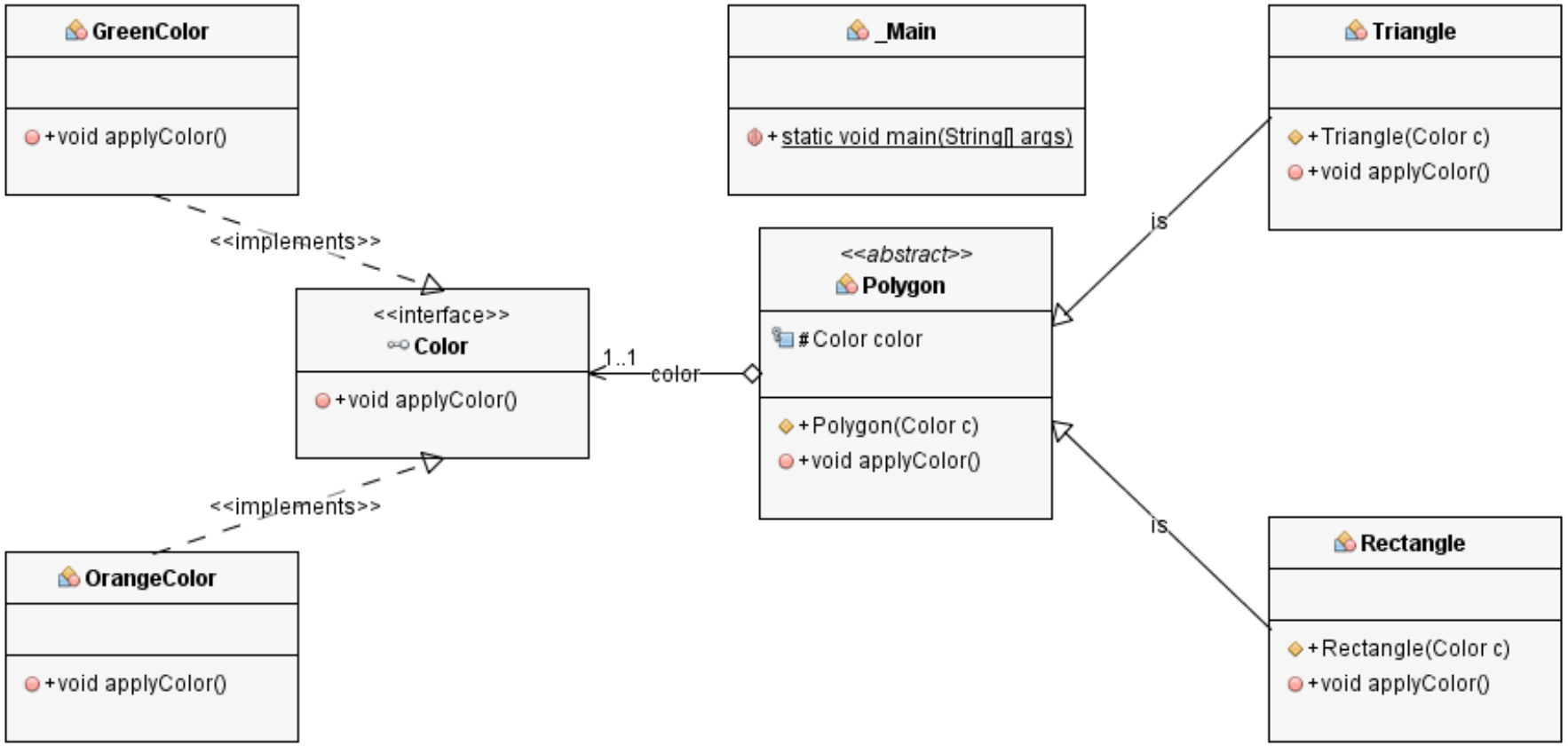
Structural Design Pattern Example: Adapter

- Match interfaces of different classes



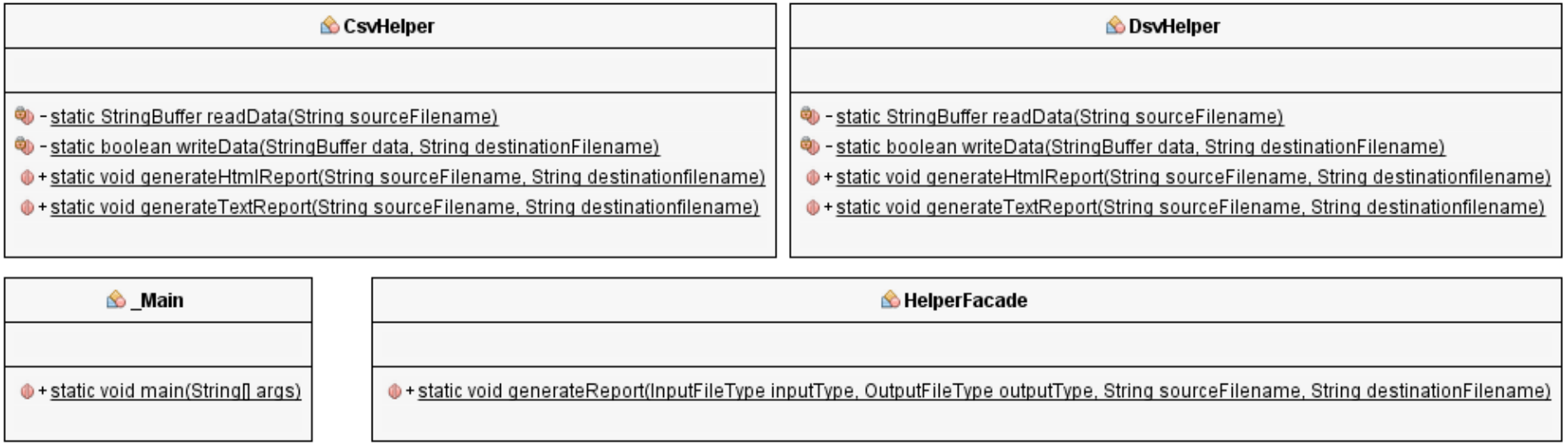
Structural Design Pattern Example: Bridge

- Separates an object's interface from its implementation



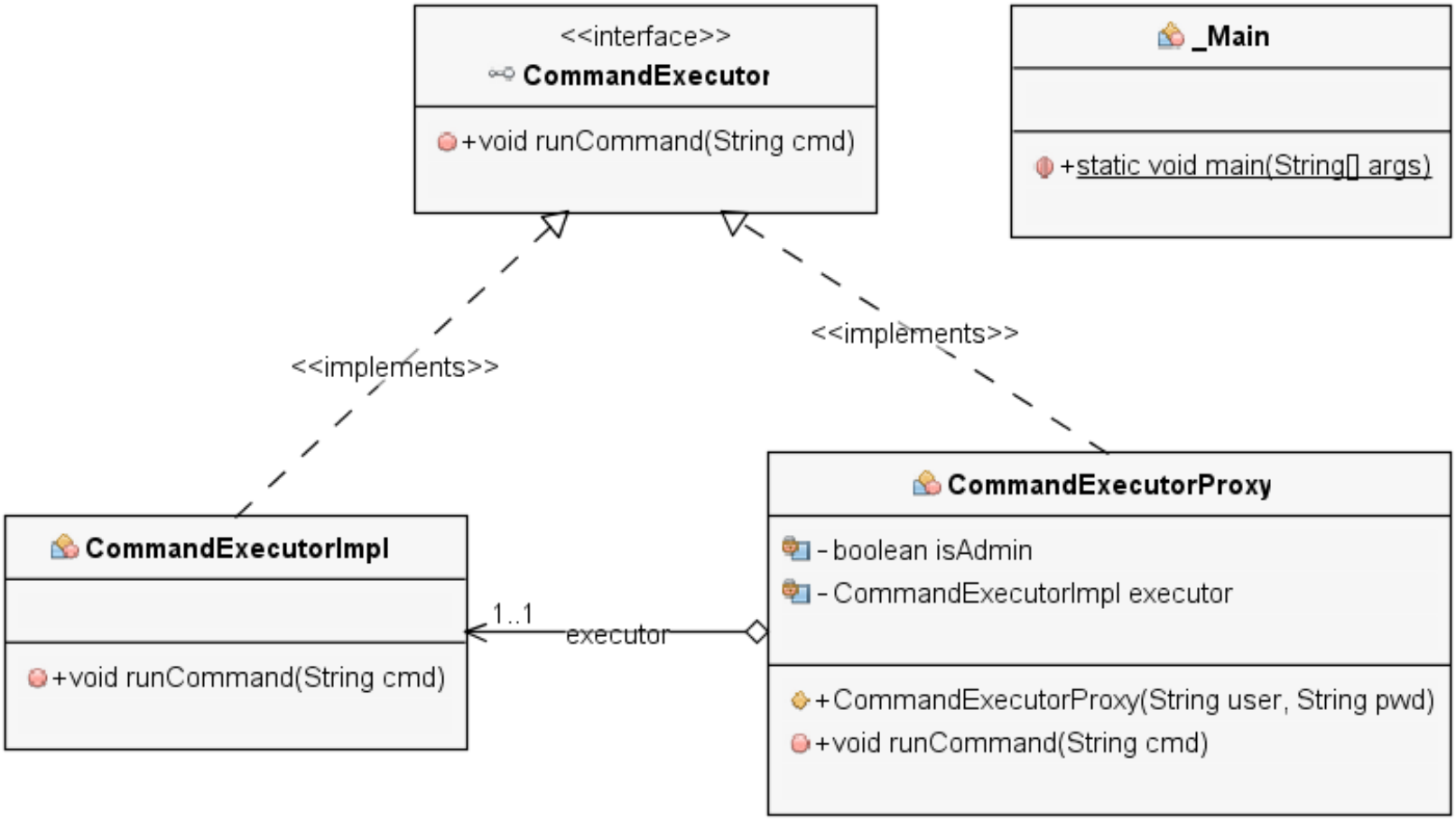
Structural Design Pattern Example: Facade

- A single class that represents an entire subsystem
- Comes from the french word “façade” (feminine noun – Collins Dictionary)
- Usually written in english without ‘ç’ but still pronounced with the /s/ sound



Structural Design Pattern Example: Proxy

- Provide a surrogate or placeholder for another object to control access to it.



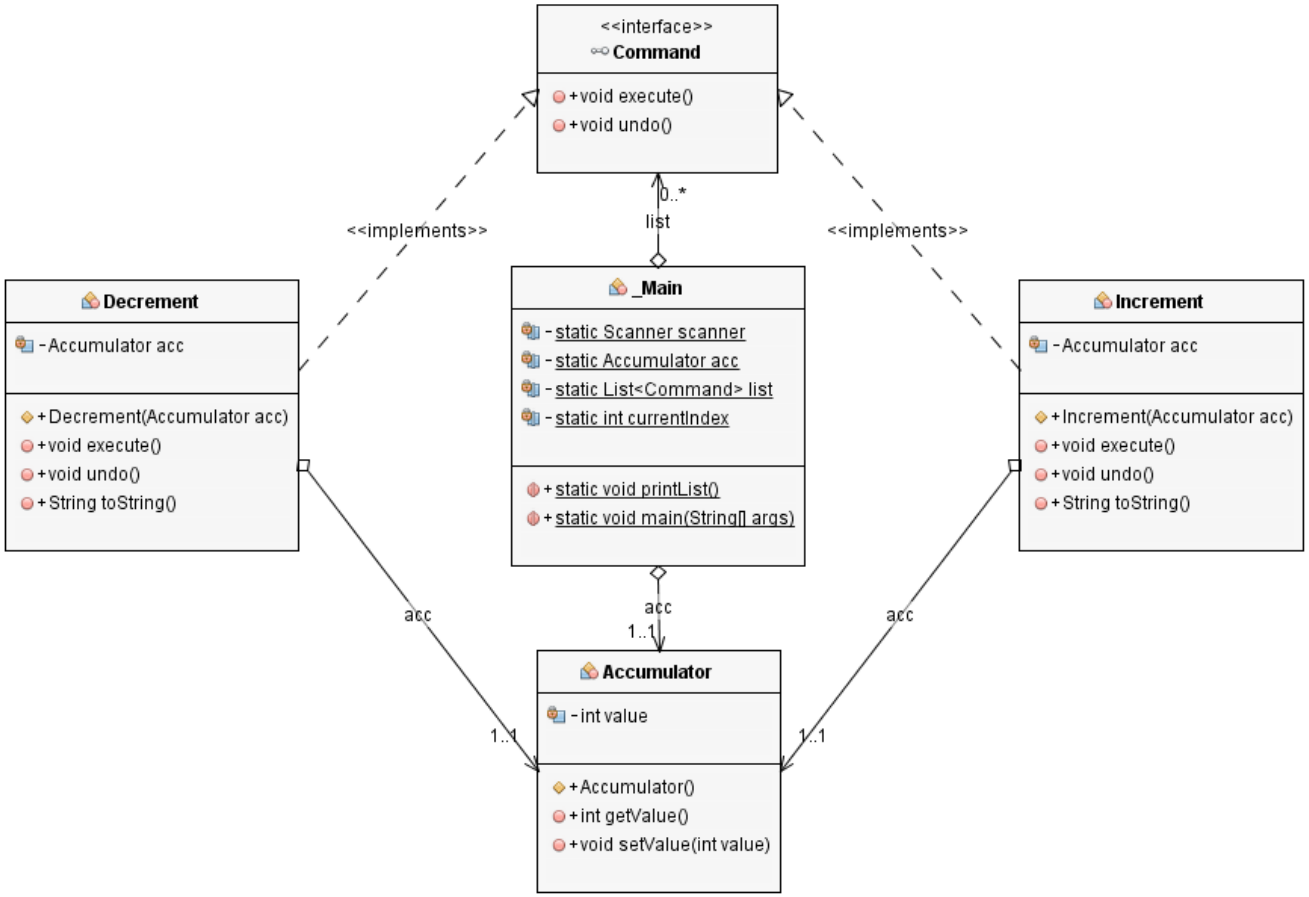
Behavioral Design Patterns

- These design patterns are all about Class's objects communication.
- Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

Chain of responsibility	A way of passing a request between a chain of objects
Command	Encapsulate a command request as an object
Interpreter	A way to include language elements in a program
Iterator	Sequentially access the elements of a collection
Mediator	Defines simplified communication between classes
Memento	Capture and restore an object's internal state
Null Object	Designed to act as a default value of an object
Observer	A way of notifying change to a number of classes
State	Alter an object's behavior when its state changes
Strategy	Encapsulates an algorithm inside a class
Template method	Defer the exact steps of an algorithm to a subclass
Visitor	Defines a new operation to a class without change

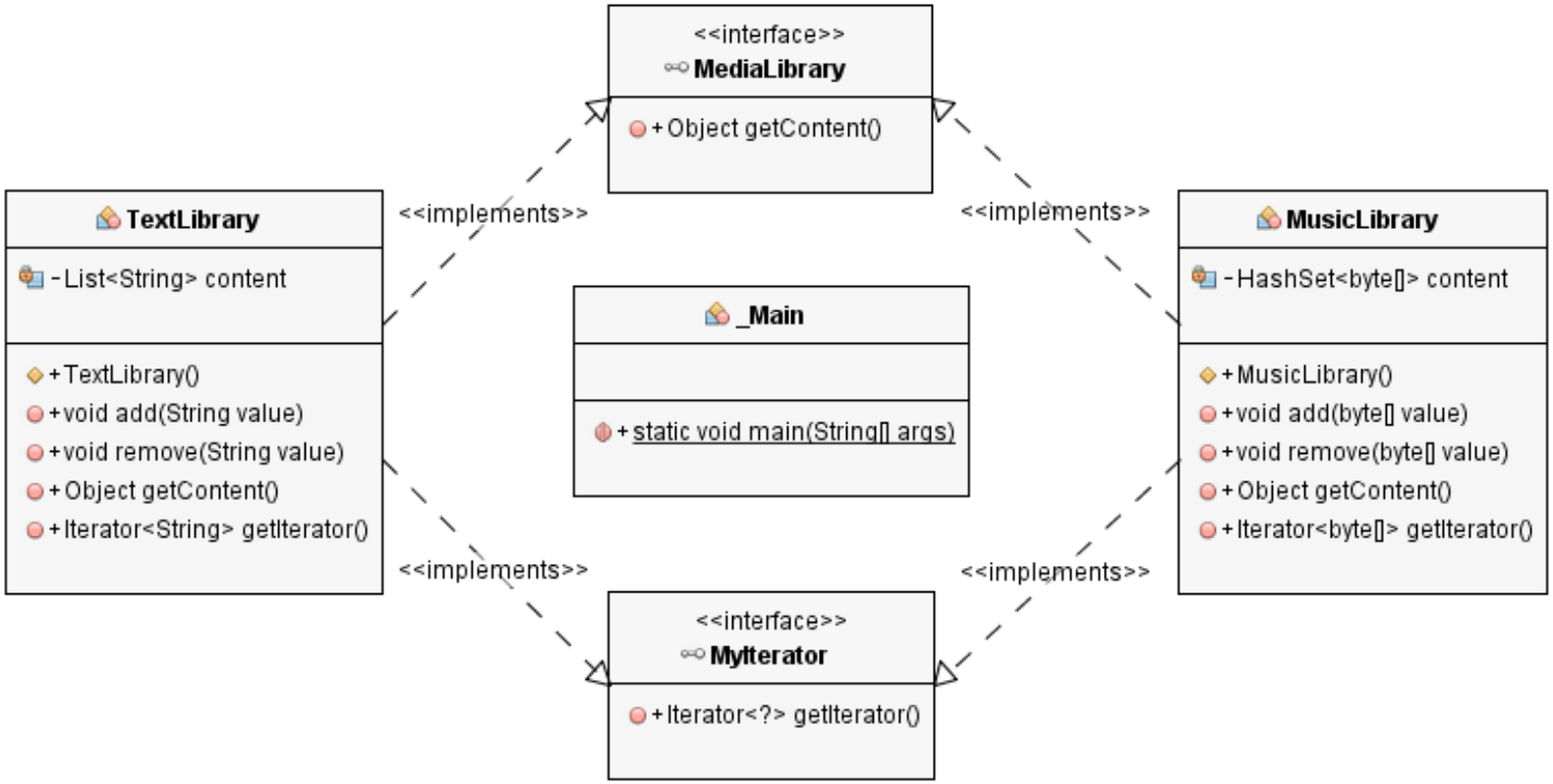
Behavioral Design Pattern Example: Command

- Encapsulate a command request as an object



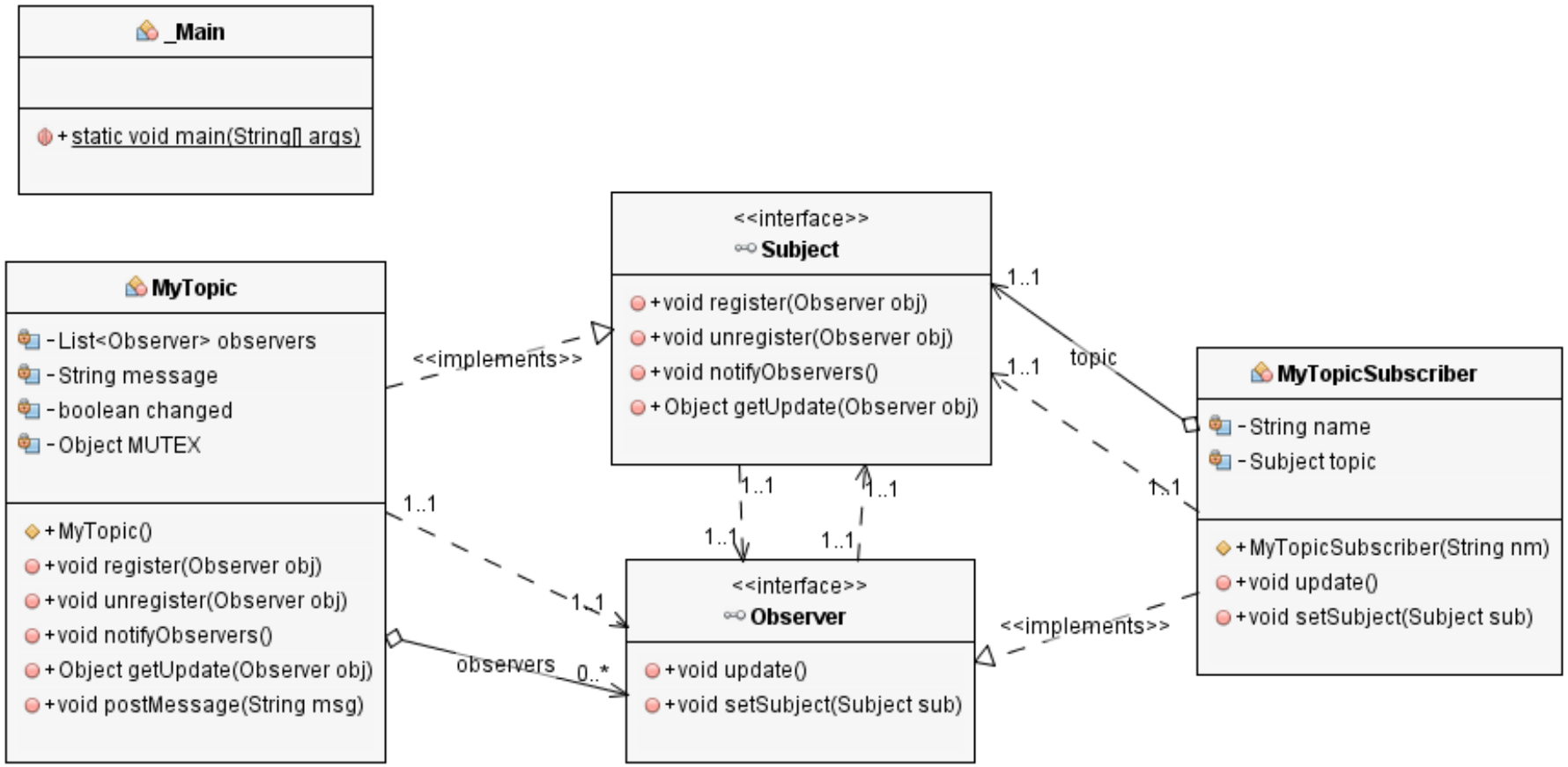
Behavioral Design Pattern Example: Iterator

- Sequentially access the elements of a collection



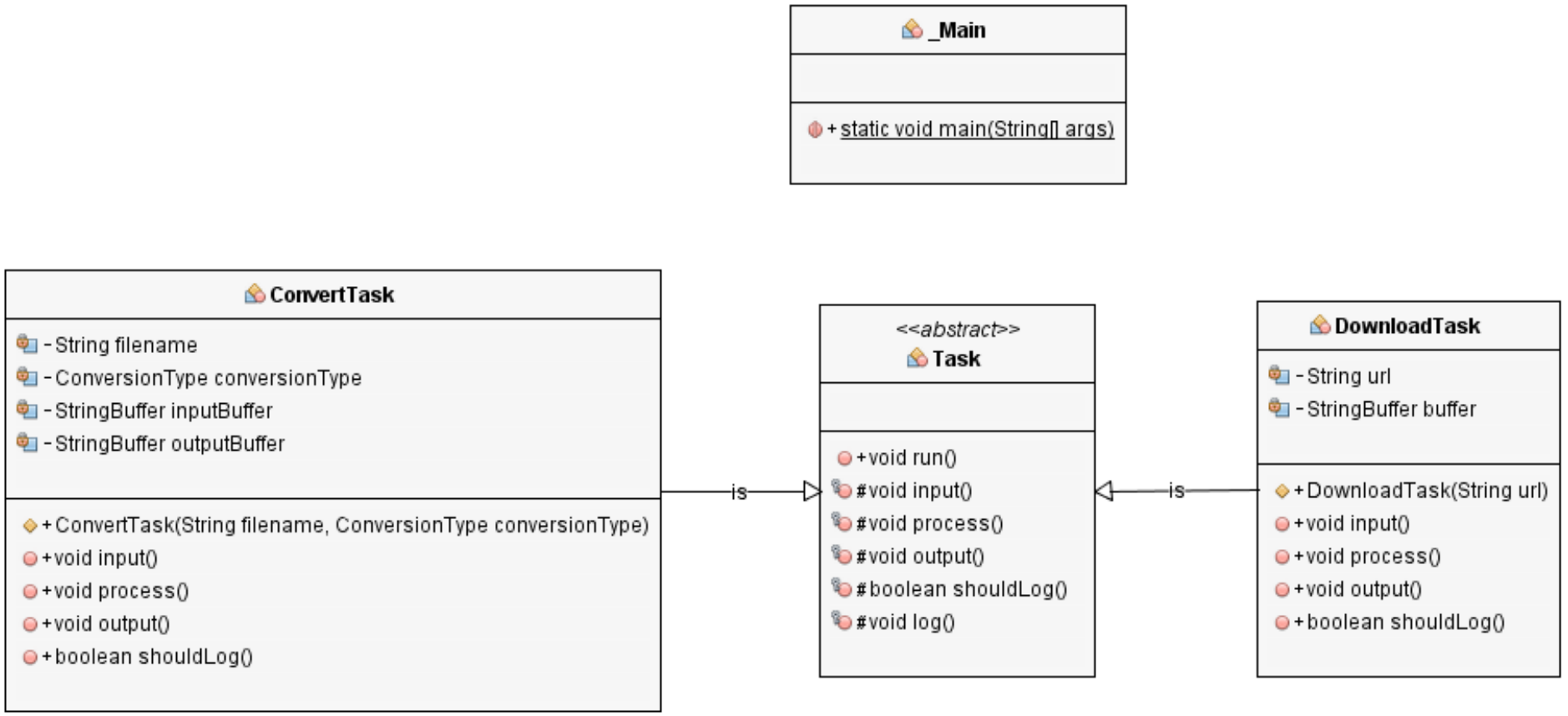
Behavioral Design Pattern Example: Observer

- A way of notifying change to a number of classes



Behavioral Design Pattern Example: Template

- Defer the exact steps of an algorithm to a subclass



AntiPatterns

- AntiPatterns provide real-world experience in recognizing recurring problems in the software industry and provide a detailed remedy for the most common predicaments.
- AntiPatterns highlight the most common problems that face the software industry and provide the tools to enable you to recognize these problems and to determine their underlying causes.
- Furthermore, AntiPatterns present a detailed plan for reversing these underlying causes and implementing productive solutions.

- The AntiPattern may be the result of a manager or developer:
  - not knowing any better
  - not having sufficient knowledge/experience in solving a particular type of problem
  - having applied a perfectly good pattern in the wrong context

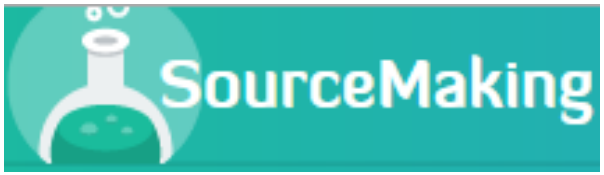
References

JournalDev - Java Design Patterns

http://www.journaldev.com/1827/

SourceMaking – Design Patterns

https://sourcemaking.com/design_patterns



SourceMaking – AntiPatterns

https://sourcemaking.com/antipatterns

Object Oriented Design - Design Patterns

http://www.oodesign.com/