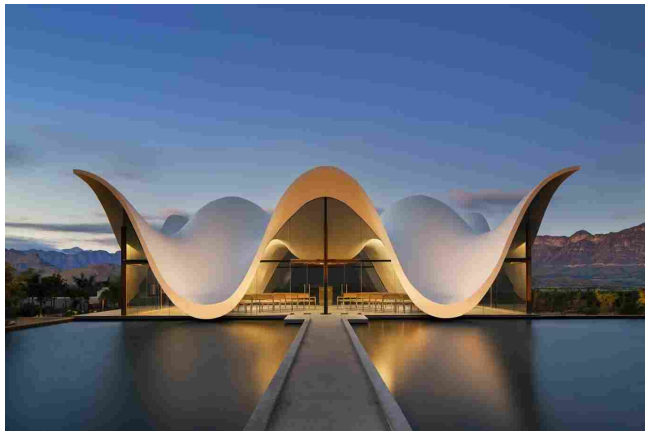
LECTURE 7: DESIGN PATTERNS

ANDREY.BOCHARNIKOV@GMAIL.COM

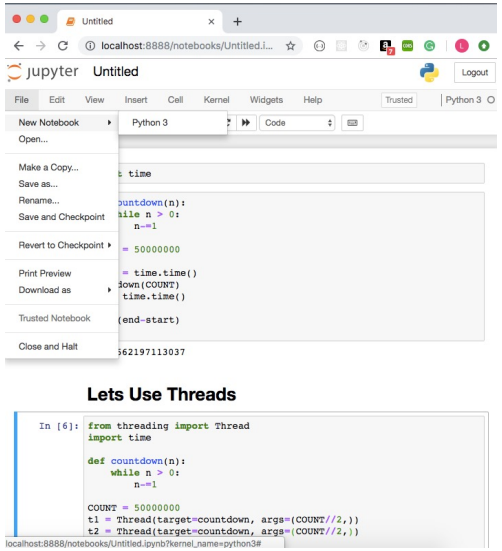
TELEGRAM: @RICKO_X

DESIGN PATTERNS

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Christopher Alexander about patterns in buildings and towns)



PYTHON OOP



- On python you can:
  - Write a simple scripts (or just test something in the terminal/Jupyter Notebook)
  - Or create complex frameworks, applications, libraries
- And for big projects, we need some rules (or patterns)

PATTERNS CLASSIFICATION

- Purpose
  - Creational patterns concern the process of object creation
  - Structural patterns deal with the composition of classes or objects
  - Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility
- Scope
  - Class patterns deal with relationships between classes and their subclasses
  - Object patterns deal with object relationships, which can be changed at runtime and are more dynamic

Creational Design Pattern

For handling Object creation mechanisms

- Constructor
- Factory
- Abstract Factory
- Prototype
- Singleton
- Builder

Structural Design Pattern

For identifying ways to realize relationships between objects

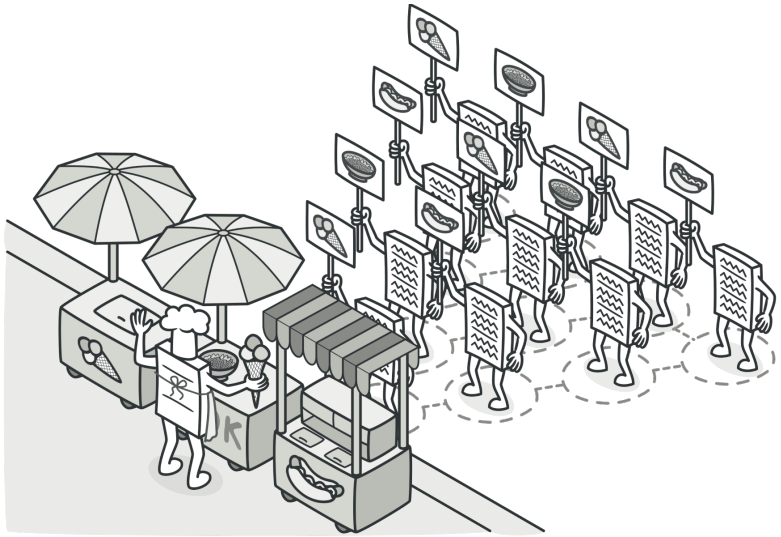
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behavioral Design Pattern

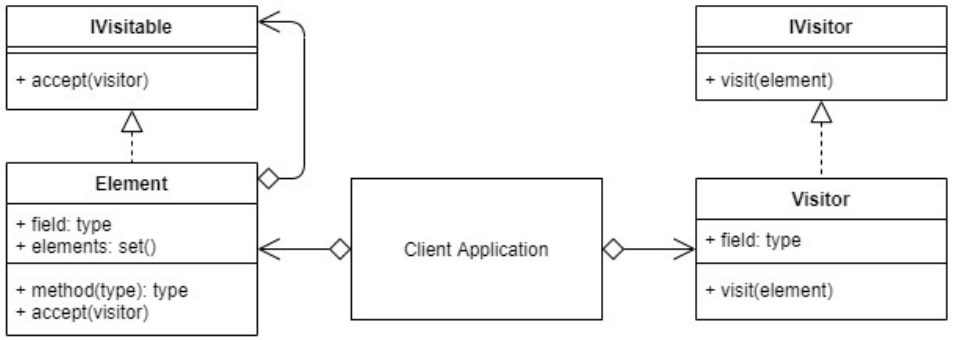
For handling communication between different objects

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template method
- Visitor

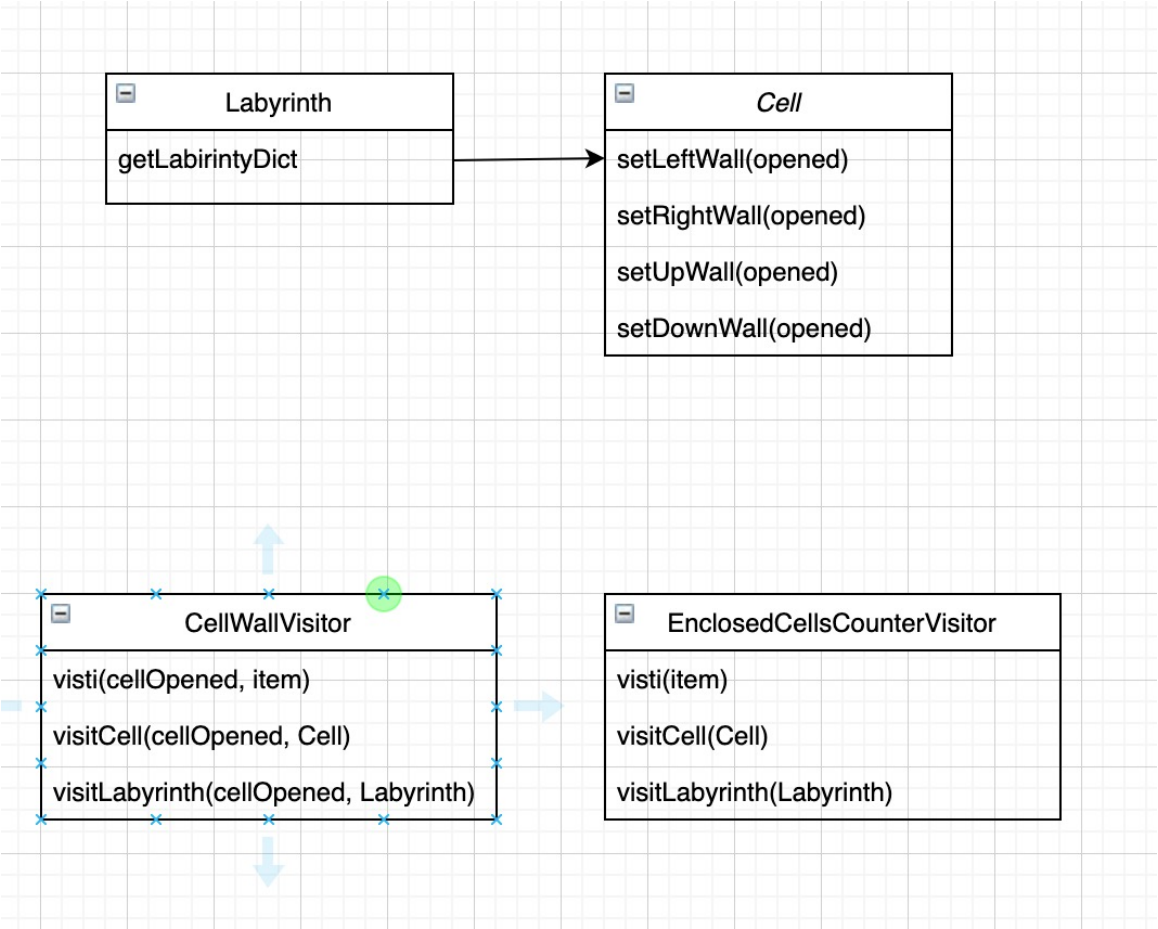
VISITOR



- Intent: Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates
- Participants:
  - Visitor Interface: An interface for the Concrete Visitors.
  - Concrete Visitor: The Concrete Visitor will traverse the hierarchy of elements.
  - Visitable Interface: The interface that elements should implement, that describes the accept() method that will allow them to be visited (traversed).
  - Concrete Element: An object that will be visited. An application will contain a variable number of Elements than can be structured in any particular hierarchy.

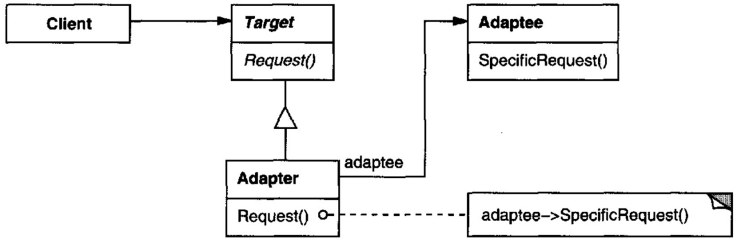


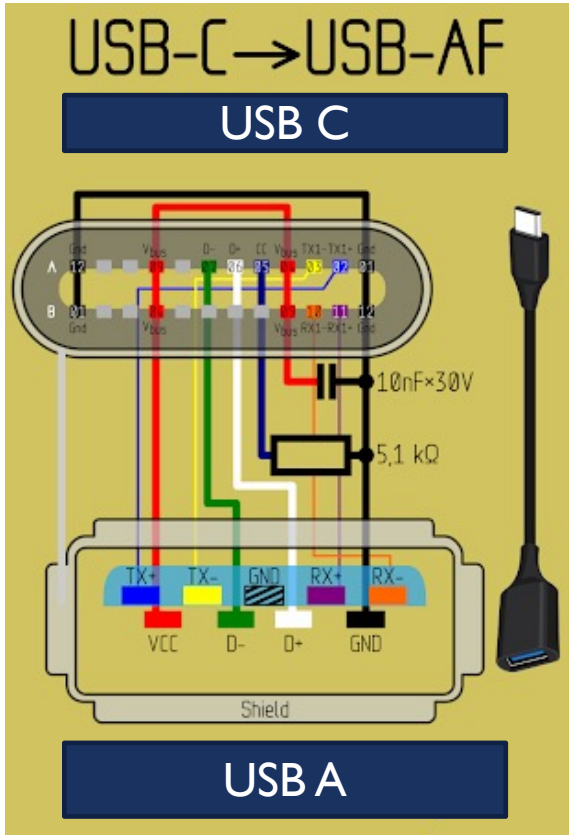
VISITOR EXAMPLE



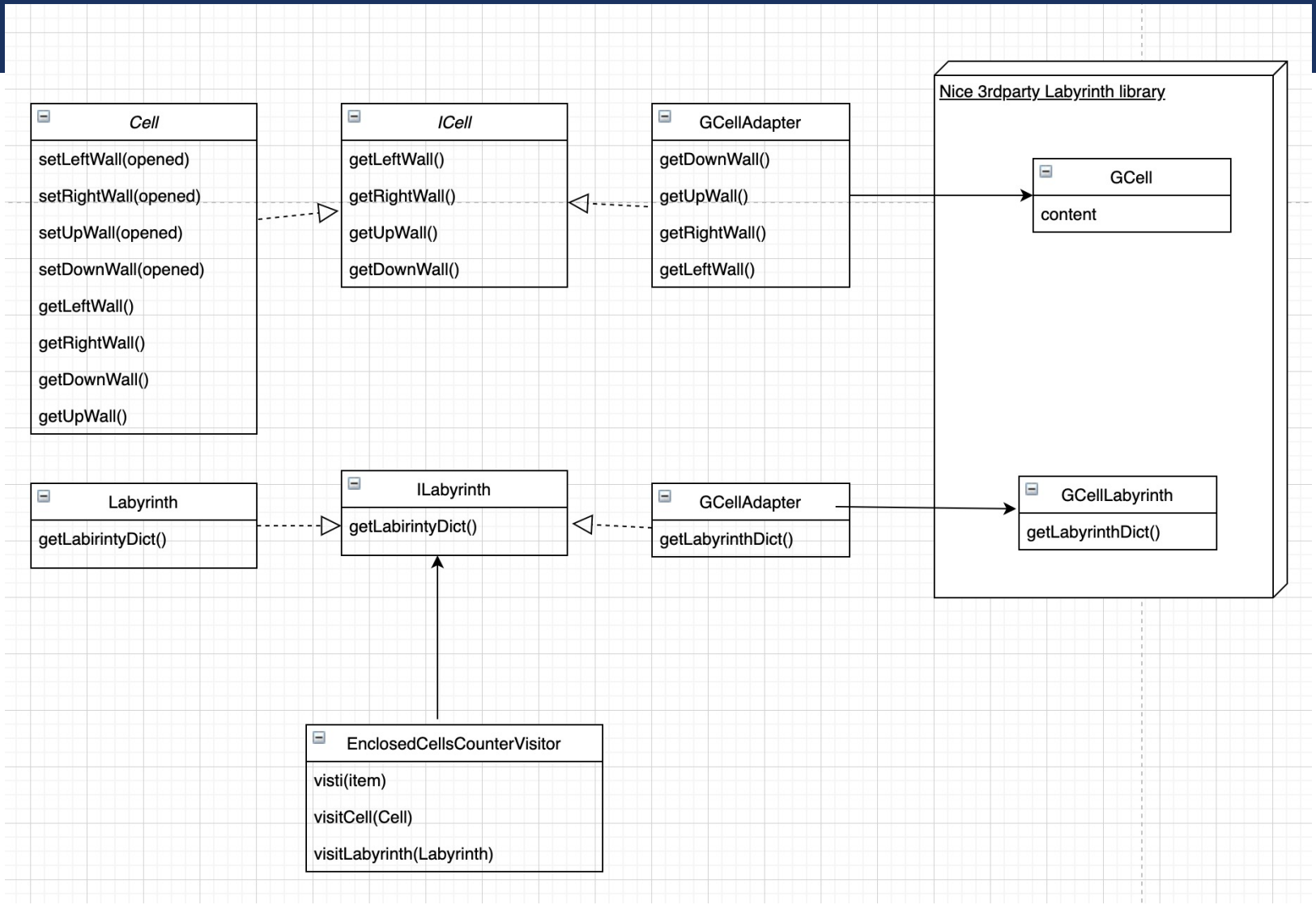
ADAPTER (OR WRAPPER)

- Intent: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces
- Adapting one input to a different predetermined output.
- Participants:
  - Target - defines the domain-specific interface that Client uses.
  - Client - collaborates with objects conforming to the Target interface.
  - Adaptee - defines an existing interface that needs adapting.
  - Adapter - adapts the interface of Adaptee to the Target interface.
- Clients call operations on an Adapter instance. In turn, the adapter calls Adaptec operations that carry out the request.



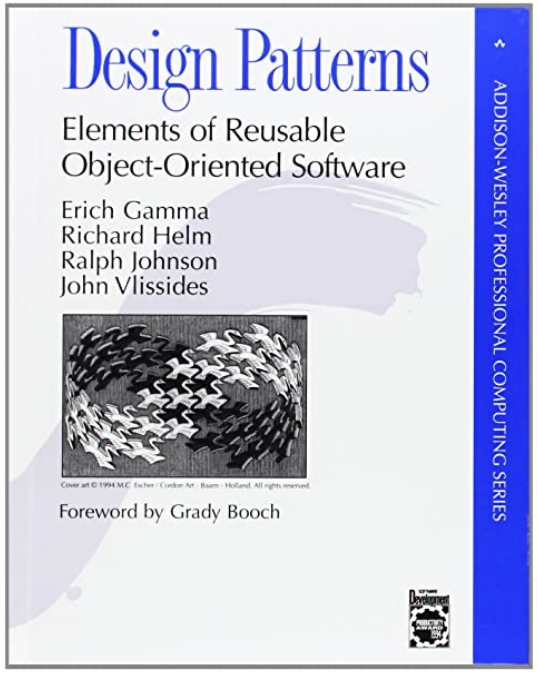

USB-C→USB-AF
USB C
10nF×30V
5,1 kΩ
TX+
TX-
GND
RX+
RX-
VCC
D-
D+
GND
Shield
USB A

EXAMPLE


Cell
setLeftWall(opened)
setRightWall(opened)
setUpWall(opened)
setDownWall(opened)
getLeftWall()
getRightWall()
getDownWall()
getUpWall()
ICell
getLeftWall()
getRightWall()
getUpWall()
getDownWall()
GCellAdapter
getDownWall()
getUpWall()
getRightWall()
getLeftWall()
Nice 3rdparty Labyrinth library
GCell
content
Labyrinth
getLabirintyDict()
ILabyrinth
getLabirintyDict()
GCellAdapter
getLabyrinthDict()
GCellLabyrinth
getLabyrinthDict()
EnclosedCellsCounterVisitor
visti(item)
visitCell(Cell)
visitLabyrinth(Labyrinth)

BOOKS

- *Design Patterns: Elements of Reusable Object-Oriented Software*
  by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides



WHAT'S NEXT

- May 22 - Saturday, 12:30 Q&A
- May 26 - Wednesday, 14:30 Lecture
- May 29 - Saturday, 12:30 Q&A, Labyrtinth task **deadline**