
# Mapping Design to Code

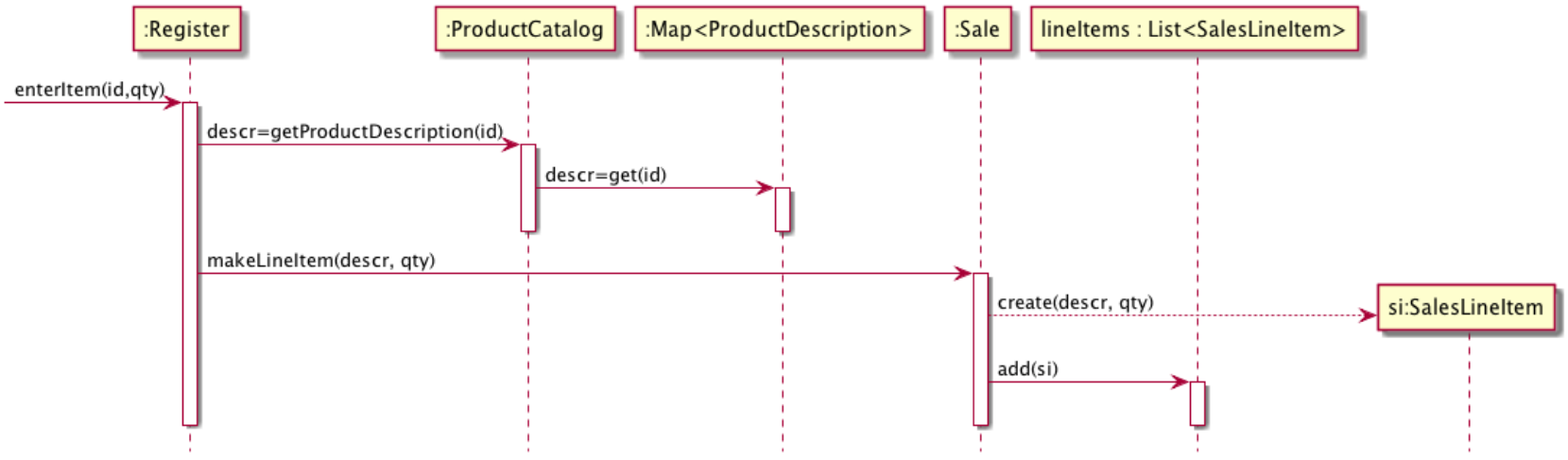
Mikael Svahnberg[1]

2016-04-21

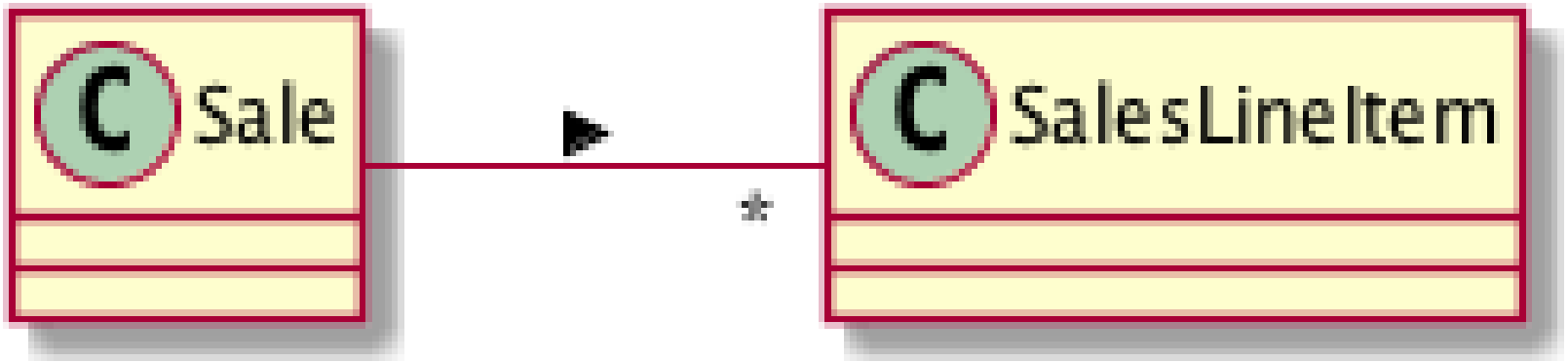
[1] Mikael.Svahnberg@bth.se

# Example: From Class Diagram to Code

# Example: From Interaction Diagrams to Code



# Example: Collections



# Discuss: Order of Implementation

- In which order should classes be implemented?
  - Larman: “Least coupled to most coupled”
  - Other suggestions:
    - Use case per use case, create stubs first, fill them out as you go.
    - First write test cases per use case, then add methods to classes (and create classes) to pass the tests.
    - First write interfaces for all classes, then inherit and implement the classes

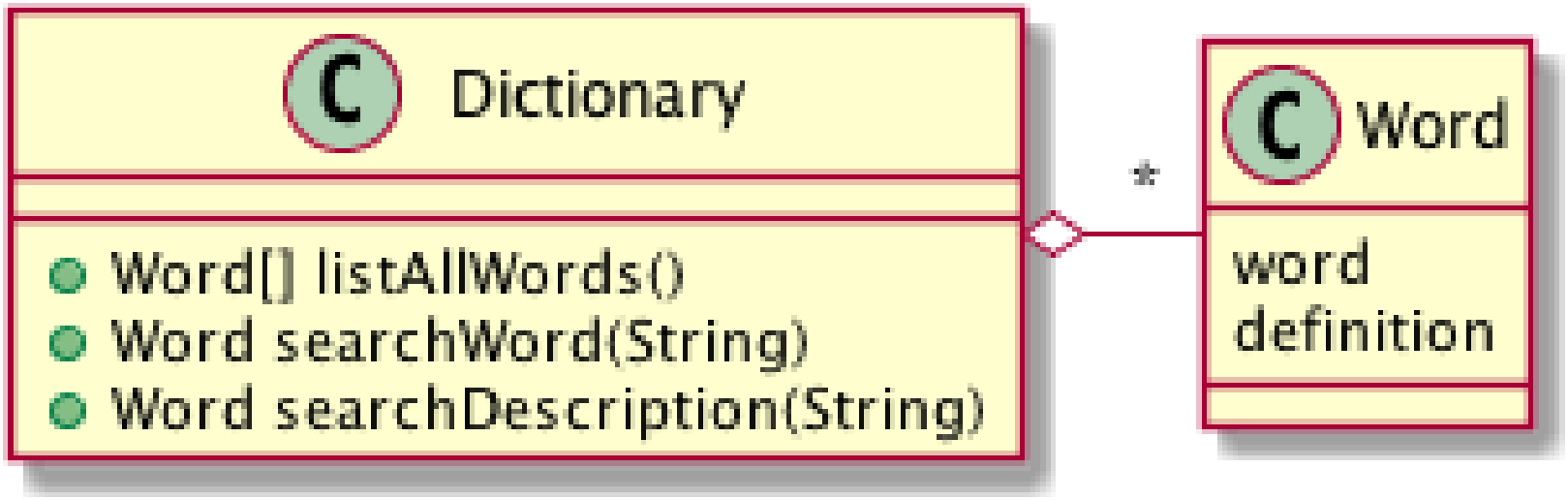

# Task

## Dictionary

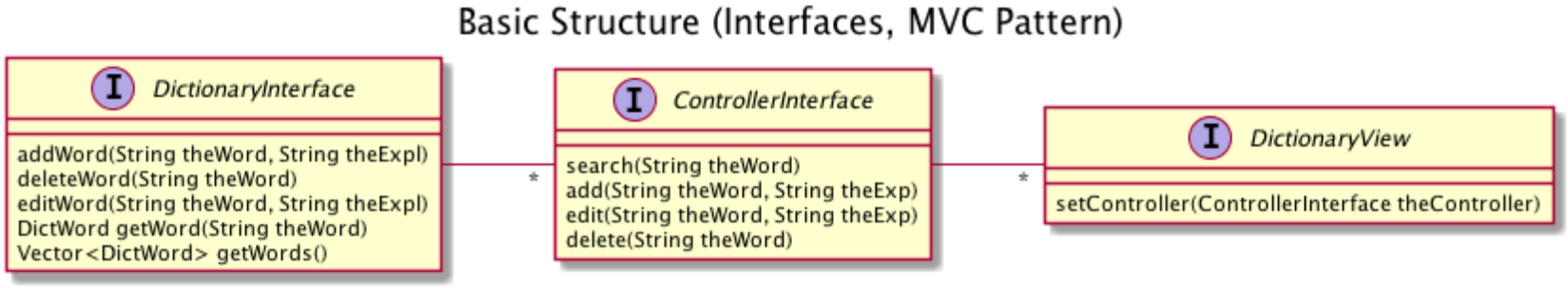
Write a dictionary program where you have words and their definitions.

- Users shall be able to browse all words.
- Users shall be able to search for words
- Users shall be able to search for definitions.
- The system shall maintain a log of activities.
- Other requirements:
  - The system shall use a graphical user interface
  - The system shall store the words and their definitions between sessions.

# Conceptual Model



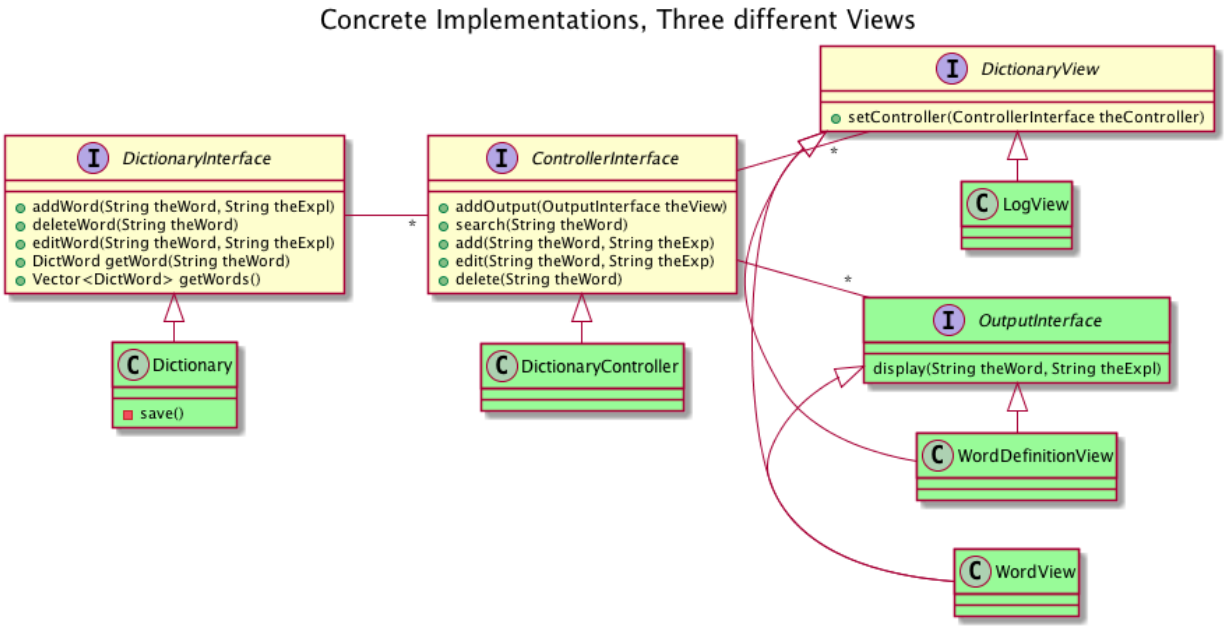
# Class Diagram I




## Note

- Views are loosely connected to Controller (pointer given via `setController()`)
- Views have no direct connection to the Dictionary.
  - Controller ensures views “behave”.
  - Dictionary ensures integrity of Data Model
- Controller loosely connected to Dictionary (pointer given to constructor)

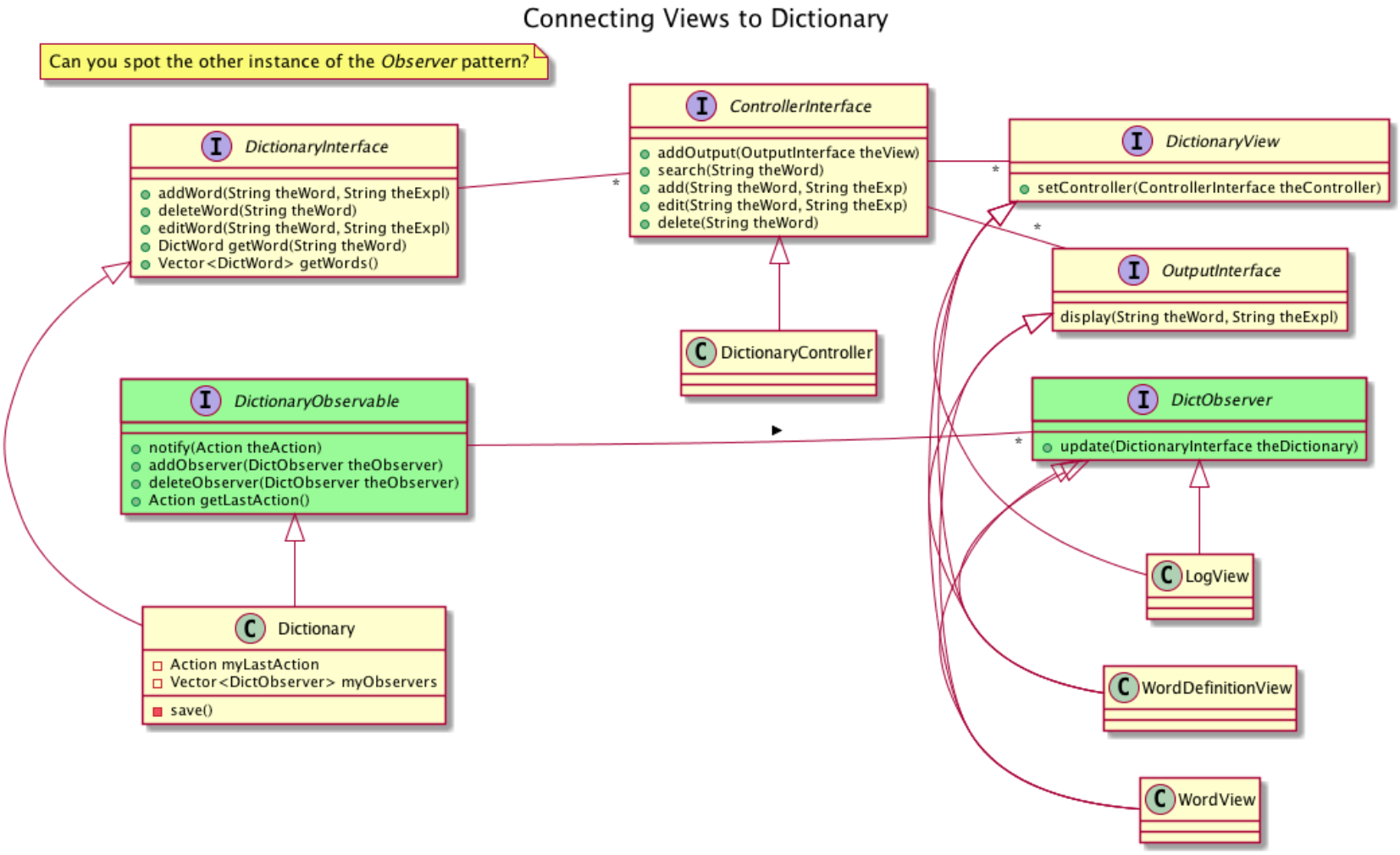
# Class Diagram II



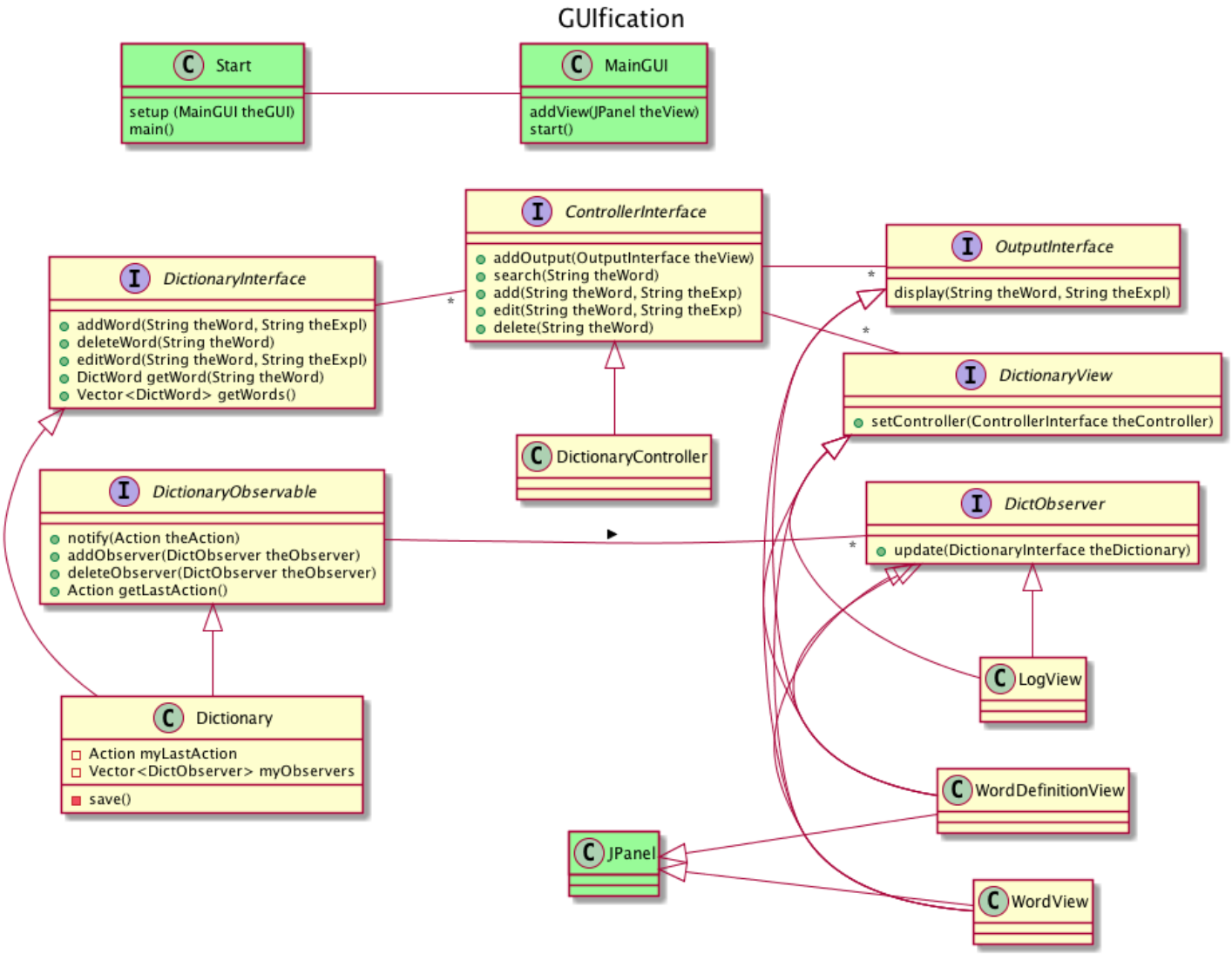

## Note

- `OutputInterface` to keep track of which word to display
  - Keeps Controller ignorant of concrete views (dependency injection)
- Views *should* track all changes (`CRUD` – Create, Remove, Update, Delete)

# Class Diagram III



# Class Diagram IV



# Class Diagram: setup method

```
public static void start::setup(MainGUI theGUI) {
  // Create Dictionary
  Dictionary theDict = new Dictionary("dict.txt");
  debugDict(theDict); // Make sure there is stuff in it.

  // Create Views
  LogView lv=new LogView();
  WordView wv=new WordView();
  WordDefinitionView wdv=new WordDefinitionView();

  // Initialise views where necessary
  wv.getWords(theDict);

  // Create and Connect the Controller
  DictionaryController dc=new DictionaryController(theDict, wdv);
  lv.setController(dc);
  wv.setController(dc);
  wdv.setController(dc); // Circular, but ok

  // Add stuff to GUI
  // theGUI.addView(lv) // skip the LogView; it prints to console/file
  theGUI.addView(wv);
  theGUI.addView(wdv);

  // Connect views to dictionary, so that changes are reflected
  theDict.addObserver(lv);
  theDict.addObserver(wv);
  theDict.addObserver(wdv);
}
```

# Discussion: Order of Implementation

- Use Cases
  - Create Word
  - Read Word (Search)
  - Update Word
  - Delete Word

Class Diagram: