Health: 👍❤️❤️❤️
Health: 💕💕

Inventory: 💫
Health: 🎃 🎃

How to implement this?

Inventory: 🎮
OBSERVER PATTERN

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
Subject
- Attach (Observer)
- Detach (Observer)
- Notify()

Observer
- Update()

ConcreteObserver
- Update()
- observerState
- subject.getState()
HEALTH

 Health: 💖💖

 OBSERVER

 Inventory: 🎮
Benefits:

Loose coupling between modules

Broadcast communication

Subject does not have to know the class type of an observer

Model/View/Controller frameworks
Keep in mind:

METHOD CALLS ARE synchronized
Keep in mind:

DELETING AN OBSERVER INSTANCE DOES NOT automatically detach the observer from the subject
Keep in mind:

GARBAGE COLLECTOR does not clean up attached observers
Keep in mind:

LAPSED LISTENER PROBLEM
common source of memory leaks for OOP languages
Keep in mind:

ALWAYS DETACH OBSERVERS!

to avoid memory problems and more bugs
Observer in C#

Language level support:

- delegate
- event and eventhandler
- action (action `<T>`, action `<T1, T2>` etc)
- func<TResult> (func<T,TResult>, etc)

Interface level support:

- System.IObservable<T>
- System.IObserver<T>
HOMEWORK: 22.03 - 05.04

- Read Observer chapter from Game Programming Patterns
  http://www.gameprogrammingpatterns.com/observer.html

- Read Observer chapter from CGLearn
  https://cglearn.eu/student/materials

- Do the Observer task from CGLearn