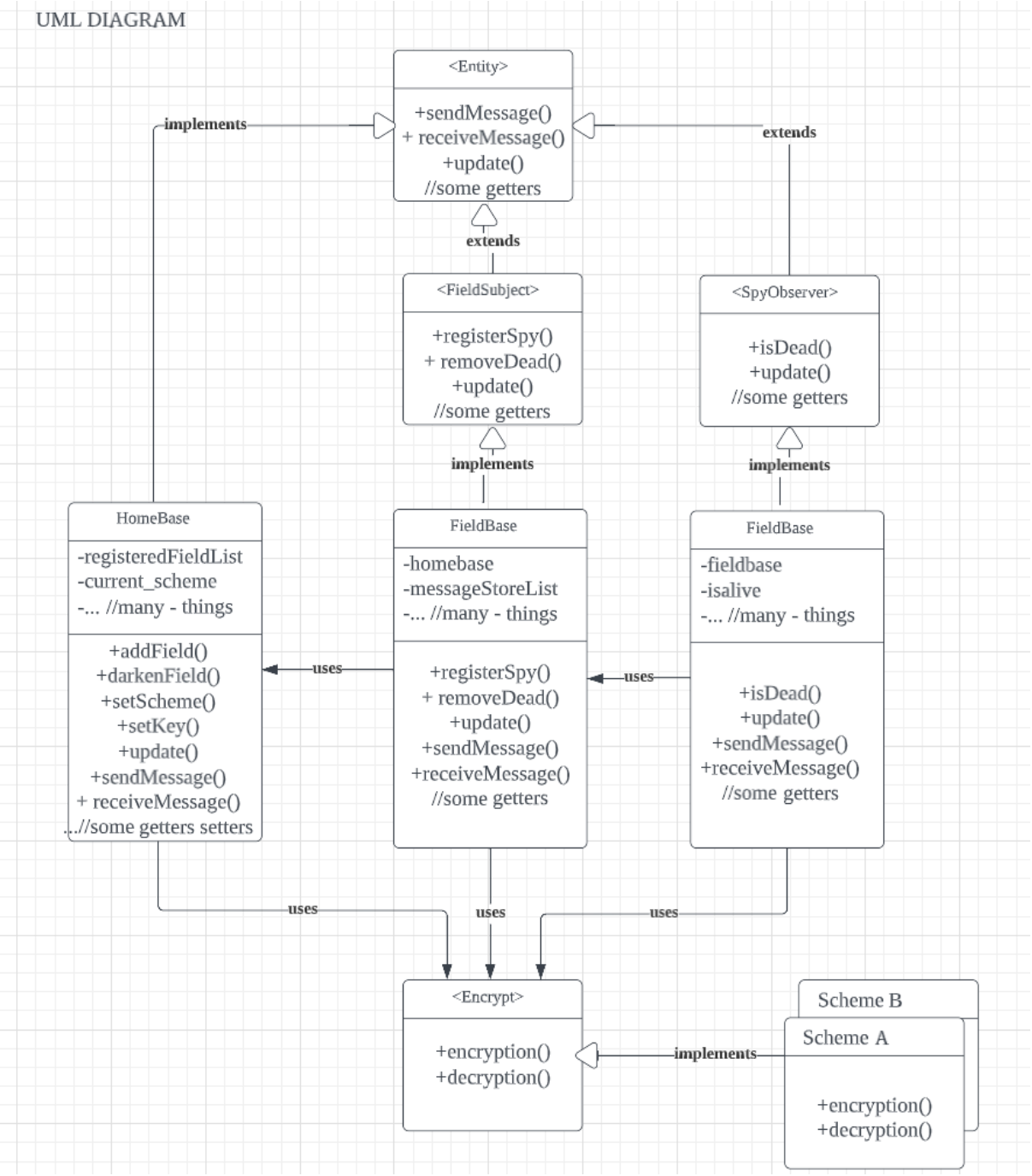
Part1

1. A UML class diagram of my design:



2. A list of design patterns you applied and why you applied them

Observer Pattern: (for fields and spies)

- to achieve registered/removed at any point in time
- to achieve sending data goals without change in Subject/Observer classes
- supports principle of loose coupling between objects interact with each other

Strategy Pattern: (for encryption schemes)

- to have different schemes [therefore different algorithms to encrypt and decrypt]
- to add/create/combine new en/decryption methods without changing other files

3. A list of design principles you feel your design enforces, and a justification as to why it does

- Open-Closed Principle: classes are open for extension. In this design, we can add new fields and spies, new scheme methods, and require no modification for existing files.
- Liskov Substitution Principle: children have all methods from their parents. In this design, Entity instances have all the methods sendMessage, receiveMessage and update() from interface<Entity>, and schemes have all methods from the <Encrypt> interface.
- Interface Segregation Principle: interfaces are separated with needs. In this design,entity implements interface <Entity> because they all need a message system, while Fields and Spies needs their own interface because there are different behaviours for them to sub() and unsub().

4. An explanation of how your design is intended to be used in the overall application

//register system explanation

- In the runner, firstly add a new HomeBase called homebase.
- Then, call homebase.setscheme() function to set a scheme for the homebase. setscheme() will call update() and update() will call all update functions for fields that are in the field list, that means, is registered. The update function in fieldbase will call spy.update() for all spies in spy list, so once you call homebase.update(), all fields and spies receive messages and their private current_scheme and current_key would be updated to the latest version.
- You can create new fields, feel free to add fields in homebase by homebase.addField() or remove fields by homebase.darkenField()
- You can create new spies, must input the field that spy belongs to because it is required in the contruster in spy. Once call spy.isDead(), that spy is removed from spylist in field.

//message system explanation

- As said before, scheme used are set and update by homebase, all entities have same current_shceme for now
- All entities use that scheme to encrypt messages for sendMessage() method, and use that scheme for receiveMessage() method. Messages would be stored in a private local arraylist, with a get function that users can see all message history and who sends this to them.
- New schemes can be created and used at any time.

Part2

For this portion, simply create a new class schemeAB implements <Encrypt>, use schemA and schemeB for this method

- set schemeAB.encryption as: schemeB.encryption(schemeA.encryption(m,k),k)
- set schemeAB.decryption as: schemeA.decryption(schemeB.decryption(m,k),k)
-

Set homebase.setscheme(schemeAB), then AB is now applied to the system.

We can layer as many as we want by editing the new class file, we can even have a method that allows users to iput schemes that they want to layer, and return a new schemeABCDE…