Design patterns used, and the reasons for them

Decorator pattern

We implemented a decorator pattern for the `EncrypterDecrypter` class. This allows for easily adding layers to the to the encrypter and decrypter, and assures that encryption and decryption is done in the proper order. An `EncrypterDecrypter` can now be constructed in the following way:

```
new Reverser(new SeededEncDec(42));
```

Or, if a different order is desired:

```
new SeededEncDec(42, new Reverser());
```

If the amount of encryption is to be expanded, this would easy for the developer and less error prone.

Observer pattern

We applied the observer pattern for the `Server`, by creating a new `ServerObserver` interface. Any instance of `Server` can accept an arbitrary amount of `ServerObserver` instances. An implementor of `ServerObserver` can choose to implement several members and get notified when specific events occur using `server.addObserver(...)`. The reason for implementing this design pattern is that this reduces the amount of responsibilities, like logging or updating a UI, which are offloaded to different classes. Another reason is that it eliminates the need of having to 'check' if an event has occurred in order to perform updates elsewhere. All together, this improves maintainability and extendability.

Feature selection from the user perspective

For the particular implementation in this task, the feature selection for the user is non-existent once the program has been compiled. In order to select features, the user would have to obtain the source code and manually create new classes or change which classes are included when adding observers/decorators. These design patterns do allow for a smooth implementation of runtime paramaters. For instance, one could have a configuration file like the following:

```
{
    "doLog": true,
    "reverserEncryption": true,
    "randomEncryption": false,
}
```

And subsequently somewhere in the source code:

```
if (config.doLog) {
    server.addObserver(logger)
```

```
}
EncrypterDecrypter encDec = new DummyEncrypter();
if (config.reverserEncyption) {
    encDec = new Reverser(encDec);
}
if (config.randomEncryption) {
    encDec = new SeededEncDec(42, encDec);
}
```