Activity 15-1 Singleton Pattern

(video segments from “Pile’o patterns”) ## The need for a single instance

- 09:25-11:50 the need for one instance of a class

Single instances can be used to allow an application to provide services, and have different components of your application access those services.

Example: We could be setting up the gateway in our applications thus (we’ll talk about the “instance” part):

```
ServiceLocator.getInstance().registerService("ProfileGateway", new MySQLGateway());
// ...
// much much later on ... in our interactors
// We need to do a downcast
ProfileGateway g = (ProfileGateway) ServiceLocator.getInstance().getService("ProfileGateway");
```

In order for this to work effectively, we must make sure that the code that adds the services and the code that uses the services are both referencing the same instance of ServiceLocator.

We have effectively three ways to address this:

- Simply create only one instance, and pass it to everyone that needs to know about it.
- Arrange your code to ensure that only one instance can be created (Singleton pattern).
- Store the “instance” information in static properties, accessed from non-static methods (Monostate pattern).

The singleton pattern

- 11:50-13:20 the static singleton pattern

Code at handout

- 13:20-16:00 the problem with static initialization and cyclic dependencies
- 16:00-17:10 using a dynamic singleton

Code at handout

- 17:10-18:20 the problem with threads
- 18:20-19:20 using synchronized

- 19:20-21:40 timing the cost of adding synchronized
- 21:40-23:40 double-checked locking and compiler optimization
- 23:40-24:10 making instance variable volatile
- 24:10-26:00 when to use each singleton pattern
- 26:00-26:40 singletons and testing

Monostate

Monostate prevents users of knowing that they are using a static variable.

Monostate uses static fields and non-static methods operating on those static fields. All methods operate on the same fields.

- 26:40-34:00 The monostate pattern

Null Object Pattern

How to handle returns from methods that “may fail”.

- 34:00-35:40 the problem with null checks
- 35:40-44:00 the “null object” pattern