ILLO

DESIGN PATTERNS

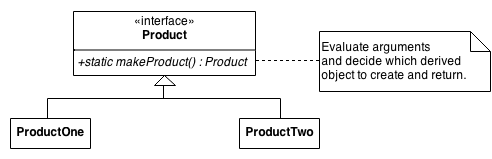
Root Digital

# 1: Factory Method Design Pattern for Activity Sources

Refresher on key terms for this pattern

* **Exercise**: Descriptively, an exercise. Has instructions how to perform the Exercise, as well as a graphic displaying it being performed. This is the core functionality.
* **Activity Source**: A grouping of Exercises. Provides methods such as adding, removing, or reordering Exercises.
* **Workout**: A subclass of Activity Source which maintains order among its Exercises. Exercises will always be provided to the user in a particular order.
* **Exercise Set**: Another subclass of Activity Source which does not maintain order – Exercises are provided to the user in a random order.

To handle the creation of Activity Sources and its subclasses, we will use a Factory Method within the Activity Source class. That is, class ActivitySource will have children classes – Workout, ExerciseSet – which inherit methods and properties from ActivitySource. ActivitySource will have a protected constructor, but a public, static makeActivitySource() method for creating instances of Workout and ExerciseSet. This is similar to the Product interface example from the sourcemaking.com, shown below.



A diagram for Illo’s implementation of the Factory Method Design Pattern is shown on the next page.

### Class Diagram

This design will accomplish a couple things. As shown, the ActivitySource class will define most of the behavior of Exercise Sets and Workouts, but will not be able to be instantiated itself on account of the protected constructor. Instead, the static makeActivitySource() will be used to make new Exercise Sets and Workouts. This way, the abstract method nextExercise() always has a strict, predictable behavior. This could also be done by completing separating Exercise Sets and Workouts, but that would result in a significant amount of code repetition, making future development and maintenance more difficult.

This design also allows us to continue to diverge Exercise Sets and Workouts with other features, as shown with completion variable in Workout. This variable will keep track of the number of times that a user completes a Workout – every time they complete every Exercise start to finish. This does not make sense to include in an ExerciseSet, where ordering is inconsistent.

# 2: Singleton for system log files

Text

Description automatically generated

We are using a singleton design pattern for a Logger class that can write to a log when requested. Having this one globally accessible instance is important for the application so that it can write to the log whenever we need it to. The log can be incredibly useful for troubleshooting and things of that nature, so having a class that can be globally accessed is very convenient so we can write to the log whenever. Other reasons why using a singleton design pattern for our logs include ease of use, customization, and resource efficiency.

# Illo Class Diagram

This diagram represents the current relationships between the non-UI classes of Illo.

