SM (M) 2018-2019 Lab 6: Design Patterns

During this Lab, we'll explore some simple implementations of design patterns to prepare for how these patterns will be presented in the exam.

Pair Programming

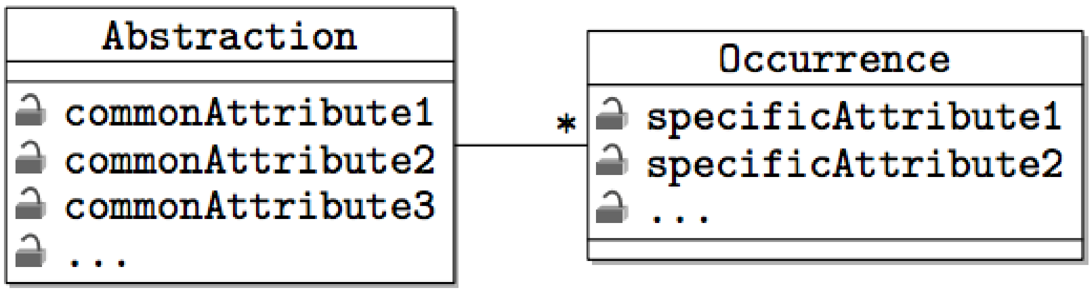
For this lab, we will be working in pairs using the pair programming technique that is popular in agile development approaches. This means two people, ONE computer. Work together to find the solutions, and switch roles at the halfway point.

Code Style

Be sure to code using appropriate code style (good comments, no sloppy structure).

Problem 1: Abstraction Occurrence

Review the abstraction occurrence design in the book or elsewhere. This is a structural design pattern. Be sure both people in your programming team understand the pattern, working together to prepare to programming.



Now open your Eclipse workspace and start a new project called Lab 3. Create a package called AbstractionOccurrence.

The following example shows how the Abstraction Occurrence pattern can be used. Write this example in Java.

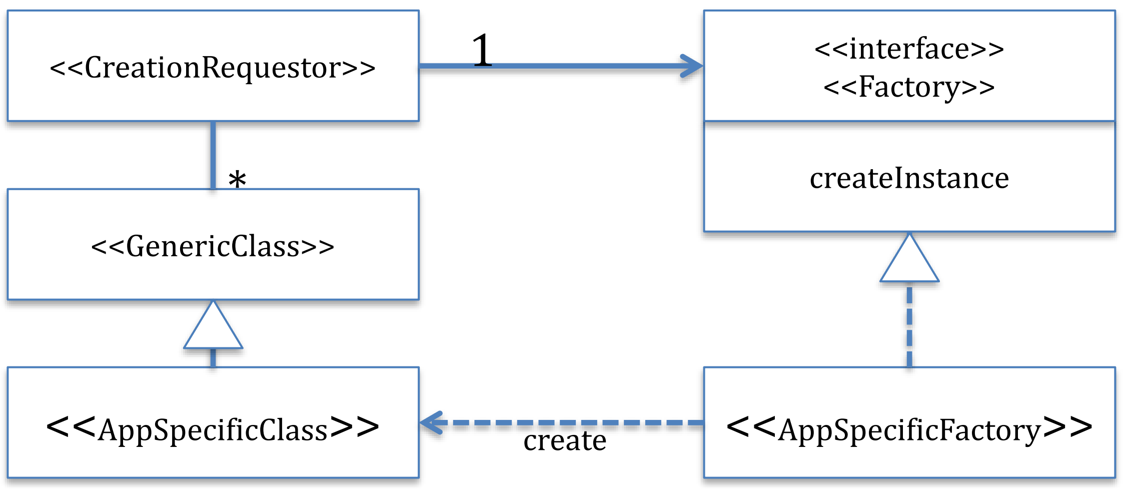
In a library, there are often multiple copies of the same books. In order to create a Java program for managing the library collection, the Abstract Occurrence pattern can be useful. Create two classes called Title and LibraryItem. Title has attributes for name, author, isbn, and publication date. LibraryItem has a bar code number. To finish the implementation of this pattern, add a 1 to many relationship between LibraryItem and Title.

Create a new package called main. Add a class to this package called DesignPatternsMain. Add a main method to this class that instantiates 2 Title objects and 6 LibraryItem objects with all the attributes and associations.

Key Point: How does abstraction occurrence differ from inheritance?

Problem 2: Factory Pattern

Review the Factory pattern using the design pattern overview on the Moodle webpage. This is a creational design pattern.



Read the full instructions below. Before you start programming, match up all of the classes/methods described below to the diagram above.

An example of a factory pattern is a factory for making cars. The factory will return car objects of different types based on arguments given to the car factory. To implement this pattern, create classes CarFactory, Car (abstract), and ReliantRobin, Lada, and Mini (these should extend Car). Choose some attributes to add to these classes.

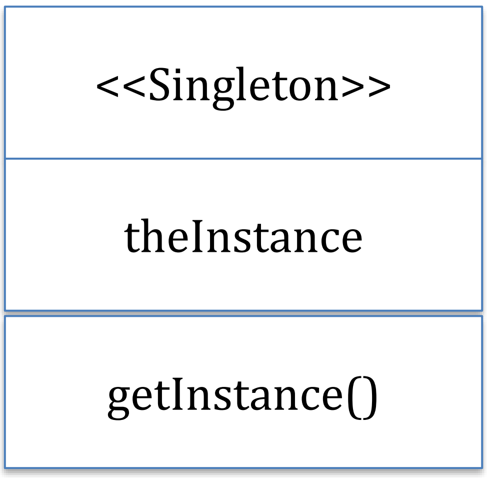
To turn this set of classes into a Factory, Car should have the abstract method makeCar(). When implemented, makeCar() should simply print the car name to the console. CarFactory should implement the interface CarBuilder with a method buildCar(String carName). Link up your CarFactory and Car classes to build cars.

Add additional code to your main method in the DesignPatternsMain package to request the factory to make one of each kind of car.

Key Point: Factory hides construction logic from API. Understand when abstract classes are used and how constructers are structured.

Problem 3: Singleton

The Singleton Pattern ensures that only one instantiation of the Singleton class can ever be accessed. Key factors to this are a private constructor and a public instance.



To implement this pattern, let's imagine we have a system that manages a company, we may want the company represented as a Singleton so that it can be accessed throughout the system. Create a class called Company using the Singleton design pattern.

Key Issue: When should the Singleton's Instance first be instantiated? Compile time? Run time? There are different ways to structure the creation logic, so meaningfully choose one.