Name:

Factory Method Pattern

CSC-490, Winter 2020

1. A *factory* refers to a method or class whose (single) purpose is to create objects. For example, suppose we have an abstract class `Pet` with lots of concrete subclasses `PetDog`, `PetCat`, `PetFish`, `PetSnake`, etc. Each pet has a data member `name`.

   (a) (2pts) Draw a class diagram for the classes described above. Leave some room.

   (b) (4pts) In the CoCalc folder `FactorMethod/PetFactory` you'll find the code for a class called `PetFactory`. Add this class to your diagram. Use dashed lines with arrows pointing from the factory class to any classes whose objects the factory creates (we'll call these "creates-a arrows").

   (c) (2pts) Which class is a factory? Which method is a factory?

   (d) (2pts) In this example the factory method is static, which is not uncommon (however not all factory methods are static!). How do you call the factory method in order to create a pet called `pet`?

2. Suppose a class `A` is a factory that creates objects of type `B`.

   (a) (2pts) Draw the associated class diagram.

   (b) (3pts) Which of the classes must be concrete, and why?

   (c) (2pts) Draw a class diagram with abstract classes `AbstractA` and `AbstractB`, and derived concrete classes `ConcreteA` and `ConcreteB`. Add a creates-a arrow from the appropriate A to B.

last updated: January 16, 2020

3. The following example is taken (and only slightly modified) from the book "Head First Design Patterns".

   - There is a class `Pizza` with a string data member `type`. This class also has methods `prepare()`, `bake()`, `cut()`, and `box()`.
   - There is a class `PizzaStore` that has a method `Pizza* orderPizza(type)`. In order to implement that method, the pizza store must create a new pizza of the appropriate type and then prepare, bake, cut, and box the pizza before returning it. Since the pizza store needs to create pizzas, it should have a factory method. However, each concrete pizza store will create different pizzas, so the factory method in `PizzaStore` should be abstract.
   - Pick your favorite two pizza places and make them into two concrete classes. If you don't have two favorites, you can use mine: `NorthEndPizzaStore` and `FlyingPiePizzaStore`. These concrete pizza stores know how to create pizzas. In particular, one can create a `NorthEndCheesePizza` and a `NorthEndPepperoniPizza` while the other can create a `FlyingPieCheesePizza` and a `FlyingPiePepperoniPizza`.

   (a) (10pts) Draw a class diagram for this story. Be sure to include all the appropriate creates-a arrows.

   (b) (5pts) Include pseudocode for the concrete factory methods and the method `orderPizza(type)`.

4. (a) (5pts) Draw the class diagram for the Factory Method pattern involving the classes `Product`, `Creator`, `ConcreteProduct`, and `ConcreteCreator`.

   (b) (4pts) Which classes from the Pizza example are playing the various roles in the Factory Method Pattern?

   Abstract Product:

   Concrete Product(s):

   Abstract Creator:

   Concrete Creator(s):

5. (5pts) Draw the class diagram for the motivating example of the Factory Method pattern from the GoF (i.e. the example with Document and Application classes). Use our conventions for class diagrams.

6. For this problem you will need the code in the CoCalc folder `FactoryMethod/ThreeLittlePigs`.

   (a) (5pts) Draw a class diagram for the program.

   (b) (5pts) Which classes are playing the various roles in the Factory Method Pattern?

   Abstract Product:

   Concrete Product(s):

   Abstract Creator:

   Concrete Creator(s):

   Not part of the Factory Method Pattern:

   (c) (15pts) The program does not currently work. To get it working you need to complete the five commented TODO steps in the files. You should start with TODO 0, located in `Pig.cpp`. The other 4 TODOs are to implement the 4 concrete factory methods in the program. When you're done, running the program should allow you to enter either 1, 2, or 3 and see the story play out.