Factory

Lecture 13

COMP 401, Fall 2018

Factory Design Pattern

- When direct construction of an object is complicated or harmful
  - … or at least undesired
  - By “direct construction”, I mean by using the *new* keyword
- Several different contexts when factory design pattern is appropriate:
  - Singleton
    - When there should only be one instance of a particular class in the whole system.
  - Multiton
    - When there should only be one instance of of an object associated with some identifying property
  - Dynamic subclass binding
    - When the specific subclass of an object can only be determined at the time that it is created
  - Complex construction
    - When the construction of the object requires complex validation and/or side effects that must be handled.
    - When null should be a valid result
    - When an existing object might need to be provided instead of a new one.

The Basic Factory Pattern

- Make the constructor private
  - This prevents direct use of the *new* keyword for creating new objects outside of the class.
- Provide static class methods that construct new instances and return them.
  - These are the "factories"
  - Return type of the factory method is the same as the class.

```
class FPClass {
        private FPClass() {
                // Private constructor
        }

        public static FPClass factoryMethod() {
                return new FPClass();
        }
}
```

```
FPClass o = FPClass.factoryMethod();
```

Singleton

One Object To Rule Them All

- When there should only be one instance of a particular class in the whole system at any given time.
  - Generally the object represents some sort of system-wide resource that may be needed by many objects in different parts of the software.
  - Modifies basic factory pattern by maintaining a single instance.
    - » Created on demand when first requested
      - Lazy initiation
    - » Stored in a private static variable and retrieved by a static getter
      - Static getter is the factory method
- lec13.ex1
  - Simple logging mechanism.
- lec13.ex2
  - A variant that allows for singleton instance to be one of several subclasses.

Multiton

- Maintains a single instance of the object with regard to some uniquely identifying characteristic of object.
- Multiton factory method
  - Static (as per the general factory pattern)
  - Provided all info. needed to create a new object if necessary.
  - Determines if corresponding object already exists.
    - If so, returns the existing object
    - If not, creates the new object and inserts it into a static structure
    - Usually implemented using a “map”

Map<K,V> and HashMap<K,V>

- Map is a collection of key/value pairs
  - Sometimes called a "dictionary"
- Java Collections Framework
  - Interface: Map<K,V>
  - Implementation: HashMap<K,V>
  - K and V are placeholders for type names
    - K for the type of the key, and V for the type of the value.
    - Must be reference types
- Basic operations:
  - Creating a new map

    ```
    Map<K,V> m = new HashMap<K,V>()
    ```
  - Inserting a value associated with a key

    ```
    m.put(key, value);
    ```
  - Retrieving a value associated with a key

    ```
    V value = m.get(key);
    ```
  - Checking to see if key already in the map

    ```
    m.containsKey(key);  // Returns boolean
    ```
  - Removing a key/value pair from the map

    ```
    m.remove(key)
    ```

Multiton Example

- lec13.ex3
  - Student is a class that models students
    - Uniquely identified by PID property
  - getStudent() is factory method
    - Notice that all info needed to create student is provided.
  - lookupStudent() is a factory method variant that only retrieves existing objects.
    - Does not create object if it doesn't exist (but must signal that fact)
    - In example, this is done by returning *null*
    - What would another option have been?
    - lec12.ex4

Value Types In A Reference Type Context

- Integer reference type
  - Map<K,V> can only work with reference types but the key we want to use is simple integer
  - Java provides reference type versions of basic value types for these kinds of situations.
    - Integer, Double, Character
- Automatic “boxing” and “unboxing”
  - As of Java 1.7, can use value type where reference type is expected and Java will automatically “box” the value.
  - Similarly, will “unbox” a reference type to the value type when assigning to a value type variable.

Dynamic Subclass Binding

- Useful when choice needs to be made between several different subclasses
  - Leaves decision about which subclass to use to the factory method
    - Factory method given any/all information that is relevant to decision and for creating new object if necessary.
- lec13.ex5

A rose by any other name

- Some design pattern textbooks / frameworks associate the name “Factory” specifically and only for dynamic subclass binding use case.
  - Other cases described as separate patterns
- In my presentation, I’ve grouped all of these use cases under the name “Factory” more generally.
  - Just something to be aware of if/when you encounter the term in tutorials/documentation/etc.