Lesson 2 Singleton

The Singleton Pattern

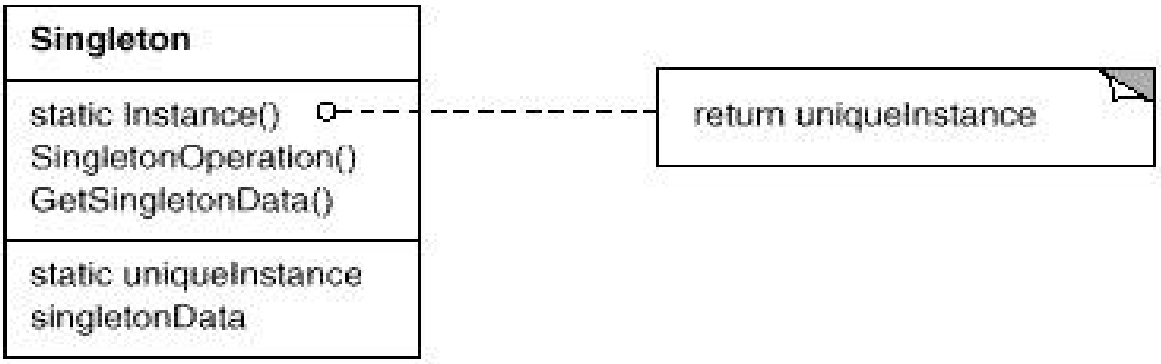
1. Intent

   Ensure a class only has one instance, and provide a global point of access to it

2. Motivation

   a. Sometimes we want just a single instance of a class to exist in the system
   b. For example, we want just one factory for a family of products
   c. We need to have that one instance easily accessible
   d. And we want to ensure that additional instances of the class cannot be created

3. Structure



4. Consequences

Benefits:
Controlled access to sole instance.
Controlled number of instances of a certain class.

5. How to implement the Singleton pattern?

A more “traditional” approach

```
public class Singleton {
	private static Singleton uniqueInstance = null;
	private int data = 0;

	/**
	* Returns a reference to the single instance. Creates the
	instance if it
	 * does not yet exist. (This is called lazy instantiation.)
	 */
	public static Singleton getInstance() {
		if (uniqueInstance == null)
			uniqueInstance = new Singleton();
		return uniqueInstance;
	}

	/**
	 * The Singleton Constructor. Note that it is private! No client
can
	 * instantiate a Singleton object directly!
	 */
	private Singleton() {
	}
	// Accessors and mutators here!
}
```

Note that the singleton instance is only created when needed. This is called lazy instantiation.
Questions: What if two threads concurrently invoke the getInstance() method? Any thread safety issues with lazy instantiation? How could we prevent this?

(What is multi-threading and thread safety?)

Pencil and Paper 2-1

1) Provide a "traditional" implementation of the Singleton Pattern (lazy instantiation) that is thread safe (like the example we saw in class). Explain why your code guarantees thread safety.
2) Is there a better and simpler way to implement a singleton class (that guarantees single instance and thread safety by itself)? Provide a solution with Java code.
   Submit a Word document for both 1 and 2.