**CS 4322 – Coding Assignment 7**

Do one of the two problems below. You can work individually or in groups of two. See Schedule for due date.

**Problem A**

(80 points max) Consider the situation where we have a class, *Provider* that performs a useful service, *service* for clients. For the sake of this assignment we will specify that the *service* method accepts a *Foo* object and calls its *increment* method:

**public** **void** service(Foo foo) {

 foo.increment();

}

The *Foo* class is provided in the Appendix, but has no real bearing for this assignment.

We need to limit the number of instances of *Provider* that exist at any one time. A *ProviderManager* provides global access to exactly 3 *Provider* instances. *ProviderManager* should have a way to:

* Get a *Provider* if one is available; if not, it will return null.
* Release a *Provider*
* Tell how many *Providers* are available

*Provider* must also supply a method, *release* that makes it available again in the *ProviderManager.* Thus, it should call the *ProviderManager’s release* method. There should only be one instance of the *ProviderManager* so model it with the Singleton pattern.

**Problem A Requirements**

1. Write the *Provider* and *ProviderManager* classes. The *Foo* class and a *ProviderManagerTester* have been provided in the Appendix A. You should study *ProviderManagerTester* to get a better idea of how these classes should work.

**Problem B**

(100 points max) Note that Problem A is a bit hokey in that an object in java cannot commit suicide (destroy itself). In other words, even though a *Provider* releases itself, and becomes available again in the *ProviderManager*, nothing prevents the client from continuing to use the *Provider* (that was just released).

A fix for this problem would be for the *ProviderManager* to keep track of who (which *Object*) has a reference to a particular *Provider*. Then, the *service* method would need a reference to the client (*this*) and *service* would check with the *ProviderManager* to make sure it has access to the *Provider*. In other words, suppose *Client1* obtained a *Provider,* used it, then released it. Then, *Client1* tried to call *service* again, the call would be denied (throw an exception) when the *ProviderManager* reported that it didn’t have access to it. Similarly, *Provider’s release* method would need a reference to the client so that it could communicate the client to the *ProviderManager* so that it could be properly released. You could use a data structure like this to implement a mechanism to store who has a *Provider*:

HashMap<Object,Provider> in *ProviderManager.*

Where the *key* is the (Object) client that has a *Provider.* This will take some thought. Consider this sample code for a Client:

**class** Client2 {

 **public** **void** run() {

 ProviderManager pMan1 = ProviderManager.*getManager*();

 Provider p1 = pMan1.getProvider(**this**);

 Foo f1 = **new** Foo();

 p1.service(**this**,f1);

 p1.release(**this**);

 **try** {

 p1.service(**this**, f1); // This should throw an exception

 }

 **catch**(RuntimeException e) {

 System.***out***.println(e);

 }

 }

}

Study the tester in Appendix B.

**Problem B Requirements**

1. Write the *Provider* and *ProviderManager* classes. The *Foo* class is found in Appendix A and a *ProviderManagerTester* is found in Appendix B. You should study *ProviderManagerTester* to get a better idea of how these classes should work.

**Appendix A**

1. *Foo* class

**public** **class** Foo {

 **private** **int** bit=0;

 **public** Foo() {

 }

 **public** **int** getBits() {

 **return** bit;

 }

 **public** **void** increment() {

 bit++;

 }

 @Override

 **public** String toString() {

 **return** "bit=" + bit;

 }

}

1. *ProviderManagerTester* class

**public** **class** ProviderManagerTester {

 **public** **static** **void** main(String[] args) {

 ProviderManager pMan1 = ProviderManager.*getManager*();

 // Verify that ProviderManager is a singleton.

 ProviderManager pMan2 = ProviderManager.*getManager*();

 System.***out***.println("pMan1.equals(pMan2)=" + pMan1.equals(pMan2));

 // Get some providers and call their service

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Provider p1 = pMan1.getProvider();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Foo f1 = **new** Foo();

 p1.service(f1);

 System.***out***.println("f1: " + f1);

 Provider p2 = pMan1.getProvider();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 p2.service(f1);

 System.***out***.println("f1: " + f1);

 Foo f2 = **new** Foo();

 p2.service(f2);

 System.***out***.println("f2: " + f2);

 Provider p3 = pMan1.getProvider();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Foo f3 = **new** Foo();

 p3.service(f3);

 System.***out***.println("f3: " + f3);

 // Try to get a 4th provider.

 System.***out***.println("Try to get another provider: " + pMan1.getProvider());

 // Release a provider.

 p2.release();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 // Get another provider.

 Provider p4 = pMan1.getProvider();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 p4.service(f3);

 // Release all providers

 p1.release();

 p3.release();

 p4.release();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 }

}

Output from *ProviderManagerTester*

pMan1.equals(pMan2)=true

Num providers available: 3

Num providers available: 2

f1: bit=1

Num providers available: 1

f1: bit=2

f2: bit=1

Num providers available: 0

f3: bit=1

Try to get another provider: null

Num providers available: 1

Num providers available: 0

Num providers available: 3

**Appendix B**

**public** **class** ProviderManagerTester {

 **public** **static** **void** main(String[] args) {

 Client1 c1 = **new** Client1();

 c1.go();

 Client2 c2 = **new** Client2();

 c2.run();

 Client3 c3 = **new** Client3();

 c3.walk();

 Client4 c4 = **new** Client4();

 c4.spin();

 Client5 c5 = **new** Client5();

 c5.fall();

 }

}

**class** Client1 {

 **public** **void** go() {

 System.***out***.println("-->Client1.go()");

 ProviderManager pMan1 = ProviderManager.*getManager*();

 // Verify that ProviderManager is a singleton.

 ProviderManager pMan2 = ProviderManager.*getManager*();

 System.***out***.println("pMan1.equals(pMan2)=" + pMan1.equals(pMan2));

 // Get some providers and call their service

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Provider p1 = pMan1.getProvider(**this**);

 **if**(p1==**null**) {

 System.***out***.println("No provider available");

 }

 **else** {

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Foo f1 = **new** Foo();

 p1.service(**this**,f1);

 System.***out***.println("f1: " + f1);

 p1.service(**this**,f1);

 System.***out***.println("f1: " + f1);

 }

 }

}

**class** Client2 {

 **public** **void** run() {

 System.***out***.println("-->Client2.run()");

 ProviderManager pMan1 = ProviderManager.*getManager*();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Provider p1 = pMan1.getProvider(**this**);

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Foo f1 = **new** Foo();

 p1.service(**this**,f1);

 System.***out***.println("f1: " + f1);

 p1.release(**this**);

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 **try** {

 p1.service(**this**, f1);

 }

 **catch**(RuntimeException e) {

 System.***out***.println(e);

 }

 }

}

**class** Client3 {

 **public** **void** walk() {

 System.***out***.println("-->Client3.walk()");

 ProviderManager pMan1 = ProviderManager.*getManager*();

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Provider p1 = pMan1.getProvider(**this**);

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Foo f1 = **new** Foo();

 p1.service(**this**,f1);

 System.***out***.println("f1: " + f1);

 }

}

**class** Client4 {

 **public** **void** spin() {

 System.***out***.println("-->Client4.spin()");

 ProviderManager pMan1 = ProviderManager.*getManager*();

 // Get some providers and call their service

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Provider p1 = pMan1.getProvider(**this**);

 **if**(p1==**null**) {

 System.***out***.println("No provider available");

 }

 **else** {

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 }

 }

}

**class** Client5 {

 **public** **void** fall() {

 System.***out***.println("-->Client5.fall()");

 ProviderManager pMan1 = ProviderManager.*getManager*();

 // Get some providers and call their service

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 Provider p1 = pMan1.getProvider(**this**);

 **if**(p1==**null**) {

 System.***out***.println("Tried to get provider, but none available");

 }

 **else** {

 System.***out***.println("Num providers available: " + pMan1.numProvidersAvailable());

 }

 }

}

Output from *ProviderManagerTester*

-->Client1.go()

pMan1.equals(pMan2)=true

Num providers available: 3

Num providers available: 2

f1: bit=1

f1: bit=2

-->Client2.run()

Num providers available: 2

Num providers available: 1

f1: bit=1

Num providers available: 2

java.lang.RuntimeException: Denied Access to Provider

-->Client3.walk()

Num providers available: 2

Num providers available: 1

f1: bit=1

-->Client4.spin()

Num providers available: 1

Num providers available: 0

-->Client5.fall()

Num providers available: 0

Tried to get provider, but none available