CS 525 - ASD

Advanced Software Development

MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

Maharishi University
OF MANAGEMENT

CS 525 - ASD

Advanced Software Development

Maharishi University
OF MANAGEMENT

Lesson 9 Singleton pattern

L1: ASD Introduction
L2: Strategy, Template method
L3: Observer pattern
L4: Composite pattern, iterator pattern
L5: Command pattern
L6: State pattern
L7: Chain Of Responsibility pattern

Midterm

L8: Proxy, Adapter, Mediator
L9: Factory, Builder, Decorator, Singleton
L10: Framework design
L11: Framework implementation
L12: Framework example: Spring framework
L13: Framework example: Spring framework

Final

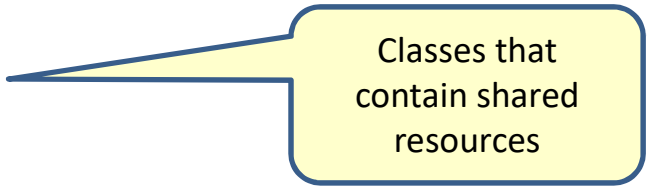
Singleton

- A singleton class can have only one instance.

- The office of the President of the United States is a *Singleton*. The United States Constitution specifies that there can be at most one active president at any given time.

Examples of singleton classes

- ConnectionPool
- PrinterBuffer
- Cache
- Configuration from configuration file



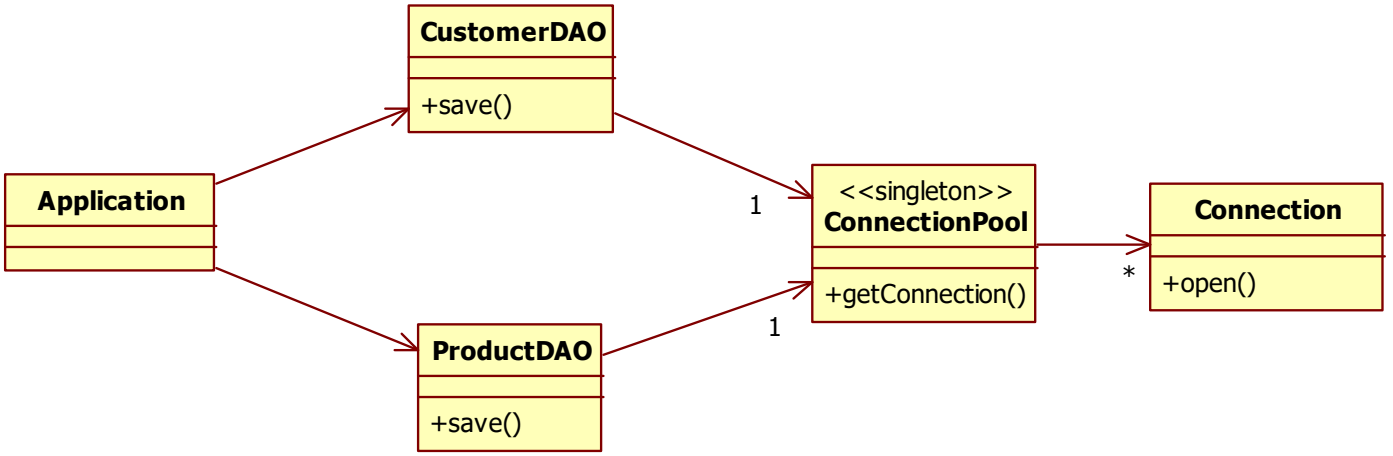
Singleton example

```
public class ConnectionPool {
  private static ConnectionPool pool = new ConnectionPool();
  //this is a pool with only 1 connection
  private Connection connection = new Connection();

  private ConnectionPool() {}

  public static ConnectionPool getPool() {
    return pool;
  }
  public Connection getConnection(){
    return connection;
  }
}
```

Declare a private static instance of the class

Make the constructor private

Add a static method to get an instance of the singleton class



The application

```
public class CustomerDAO {
  Connection conn;

  public CustomerDAO() {
    conn = ConnectionPool.getPool().getConnection();
  }
  public void save(){
    conn.open();
  }
}
```

```
public class Connection {
  public void open(){
    System.out.println("open connection to DB");
  }
}
```

```
public class ProductDAO {
  Connection conn;

  public ProductDAO() {
    conn = ConnectionPool.getPool().getConnection();
  }
  public void save(){
    conn.open();
  }
}
```

```
public class Application {

  public static void main(String[] args) {
    CustomerDAO cdao = new CustomerDAO();
    cdao.save();
    ProductDAO pdao = new ProductDAO();
    pdao.save();
  }
}
```

Eager and lazy instantiation

```
public class ConnectionPool {
  private static ConnectionPool pool = new ConnectionPool();
  //this is a pool with only 1 connection
  private Connection connection = new Connection();

  private ConnectionPool() {}

  public static ConnectionPool getPool() {
    return pool;
  }
  public Connection getConnection(){
    return connection;
  }
}
```

Eager instantiation

```
public class ConnectionPool {
  private static ConnectionPool pool;
  // this is a pool with only 1 connection
  private Connection connection = new Connection();

  private ConnectionPool() {}

  public static ConnectionPool getPool() {
    if (pool == null) {
      pool = new ConnectionPool();
    }
    return pool;
  }

  public Connection getConnection() {
    return connection;
  }
}
```

Lazy instantiation

Issues with singleton

- With reflection you can still create more instances of the singleton
  - Make the singleton reflection safe
- If 2 threads create a singleton at almost the same time, you might end up with 2 instances
  - Make the singleton thread safe
- With serialization and deserialization we might end up with 2 instances
  - Make the singleton serialization safe

Reflection

```
public class ReflectionSingletonTest {

  public static void main(String[] args) {
    ConnectionPool instanceOne = ConnectionPool.getPool();
    ConnectionPool instanceTwo = null;
    try {
       Constructor[] constructors = ConnectionPool.class.getDeclaredConstructors();
       for (Constructor constructor : constructors) {
         //Below code will break the singleton pattern
         constructor.setAccessible(true);
         instanceTwo = (ConnectionPool) constructor.newInstance();
         break;
       }
    } catch (Exception e) {
            e.printStackTrace();
    }
    System.out.println(instanceOne.getClass().getName()+" with hascode: " + instanceOne.hashCode());
    System.out.println(instanceTwo.getClass().getName()+" with hascode: " + instanceTwo.hashCode());
  }
}
```

Two instances of the ConnectionPool

```
reflection.ConnectionPool with hascode: 366712642
reflection.ConnectionPool with hascode: 1829164700
```

Make the singleton reflection safe

```
public class ConnectionPool {
  private static ConnectionPool pool;
  // this is a pool with only 1 connection
  private Connection connection = new Connection();

  private ConnectionPool() {
    // Prevent form the reflection api.
    if (pool != null) {
      throw new RuntimeException("Use getInstance() method to get the single instance of this
                                  class.");
    }
  }

  public static synchronized ConnectionPool getPool() {
    if (pool == null) {
      pool = new ConnectionPool();
    }
    return pool;
  }

  public Connection getConnection() {
    return connection;
  }
}
```

Throw exception if constructor is called twice

```
java.lang.reflect.InvocationTargetException
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
... 5 more
Exception in thread "main" java.lang.NullPointerException
reflection.safe.ConnectionPool with hascode: 2018699554at
reflection.safe.ReflectionSingletonTest.main(ReflectionSingletonTest.java:22)
```

Thread safety

```
public class SingletonTest {
  public static void main(String[] args) {
        //Thread 1
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
            ConnectionPool instance1 = ConnectionPool.getPool();
                System.out.println("Instance 1 hash:" + instance1.hashCode());
            }
        });

        //Thread 2
        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
            ConnectionPool instance2 = ConnectionPool.getPool();
                System.out.println("Instance 2 hash:" + instance2.hashCode());
            }
        });

        //start both the threads
        t1.start();
        t2.start();
    }
}
```

```
Instance 1 hash:1870487130
Instance 2 hash:354710606
```

Two instances of the ConnectionPool

Thread safety solution 1

```
public class ConnectionPool {
  private static ConnectionPool pool;
  // this is a pool with only 1 connection
  private Connection connection = new Connection();

  private ConnectionPool() {
    // Prevent form the reflection api.
    if (pool != null) {
      throw new RuntimeException("Use getInstance() method to get the single instance
                                  of this class.");
    }
  }

  public static synchronized ConnectionPool getPool() {
    if (pool == null) {
      pool = new ConnectionPool();
    }
    return pool;
  }

  public Connection getConnection() {
    return connection;
  }
}
```

synchronized

Performance problem

```
Instance 2 hash:892687863
Instance 1 hash:892687863
```

Thread safety solution 2

```
public class ConnectionPool {
  private static ConnectionPool pool;
  // this is a pool with only 1 connection
  private Connection connection = new Connection();

  private ConnectionPool() {
    // Prevent form the reflection api.
    if (pool != null) {
      throw new RuntimeException("Use getInstance() method to get the single instance
                                  of this class.");
    }
  }

  public static ConnectionPool getPool() {
    // Double check locking pattern
    if (pool == null) { // Check for the first time
      synchronized (ConnectionPool.class) { // Check for the second time.
        if (pool == null)  pool = new ConnectionPool();
      }
    }
    return pool;
  }

  public Connection getConnection() {
    return connection;
  }
```

Only use synchronized if the pool is not created yet

```
Instance 2 hash:892687863
Instance 1 hash:892687863
```

Serialization

```
public class SingletonTest {
  public static void main(String[] args) {
    try {
      ConnectionPool instance1 = ConnectionPool.getPool();
      ObjectOutput out = new ObjectOutputStream(new FileOutputStream("filename.ser"));
      out.writeObject(instance1);
      out.close();

      // deserialize from file to object
      ObjectInput in = new ObjectInputStream(new FileInputStream("filename.ser"));
      ConnectionPool instance2 = (ConnectionPool) in.readObject();
      in.close();

      System.out.println("instance1 hashCode=" + instance1.hashCode());
      System.out.println("instance2 hashCode=" + instance2.hashCode());

    } catch (IOException | ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

```
public class ConnectionPool implements Serializable{
```

```
public class Connection implements Serializable{
```

```
instance1 hashCode=865113938
instance2 hashCode=1831932724
```

Two instances of the ConnectionPool

Serialization solution

```
public class ConnectionPool {
  private static ConnectionPool pool;
  private Connection connection = new Connection();

  private ConnectionPool() {
    // Prevent form the reflection api.
    if (pool != null) {
      throw new RuntimeException("Use getInstance() method to get the single instance
                                  of this class.");
    }
  }

  public static ConnectionPool getPool() {
    // Double check locking pattern
    if (pool == null) { // Check for the first time
      synchronized (ConnectionPool.class) { // Check for the second time.
        if (pool == null)  pool = new ConnectionPool();
      }
    }
    return pool;
  }

  public Connection getConnection() {
    return connection;
  }

  // This method is called immediately after an object of this class is deserialized.
  protected Object readResolve() {
    return getPool();
  }
}
```

Implement the readResolve() method

```
instance1 hashCode=865113938
instance2 hashCode=865113938
```

Main point

- A singleton is a class that can have only one instance.
- Pure consciousness is the unified basis of all of creation.

Connecting the parts of knowledge with the wholeness of knowledge

1. The decorator class decorates a target class.
2. The factory pattern, builder pattern and singleton pattern are patterns that help in constructing objects

3. **Transcendental consciousness** is the field of all knowledge.
4. **Wholeness moving within itself:** In unity consciousness, one appreciates the inherent underlying unity that underlies all the diversity of creation.