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 8

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

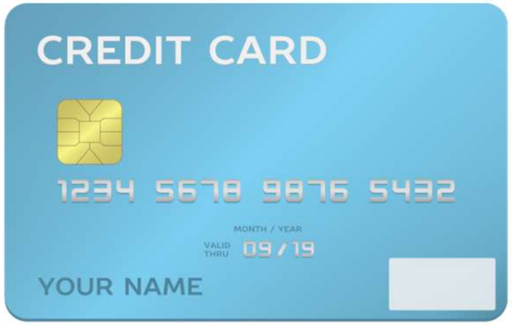
Midterm

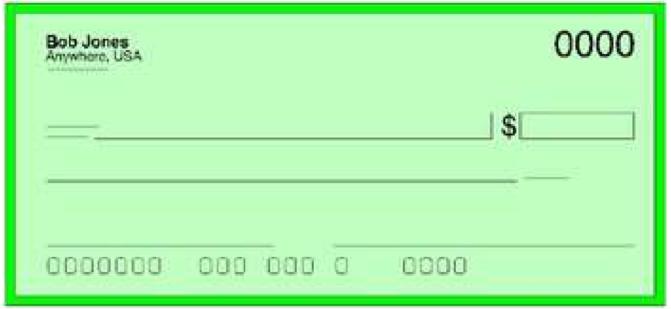
L8: Proxy, Adapter, Mediator
L9: Factory, Builder, Decorator, Singleton
L10: Framework design
L11: Framework implementation
L12: Framework example: Spring framework
L13: Framework example: Spring framework

Final

Proxy pattern

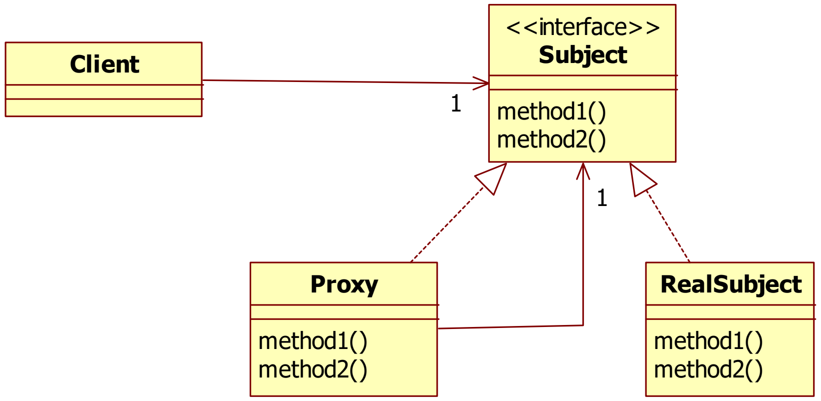
- Provides a surrogate or placeholder for another object.



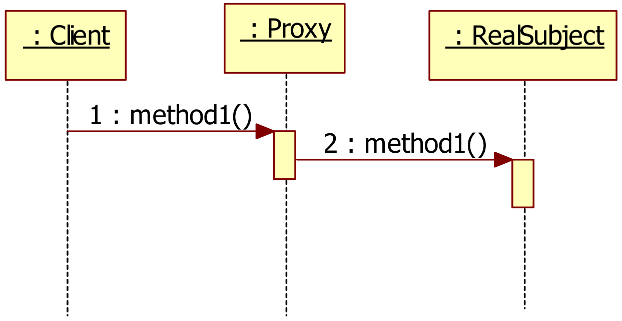


Proxy pattern

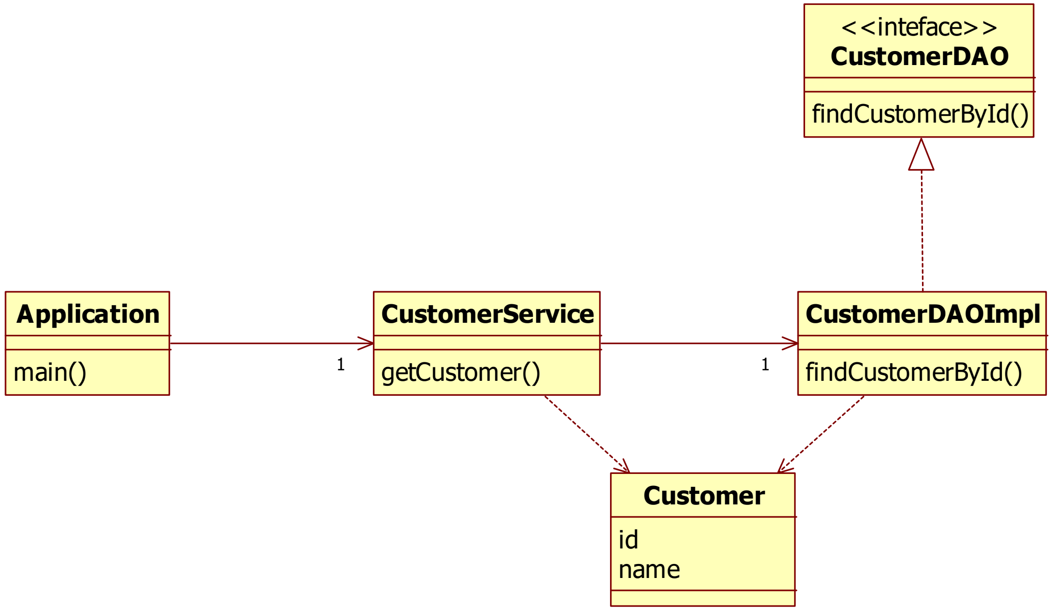
- Remote proxy
- Caching proxy
- Synchronization proxy
- Security proxy
- Transactional proxy
- Lazy load proxy
- Logging proxy
- …

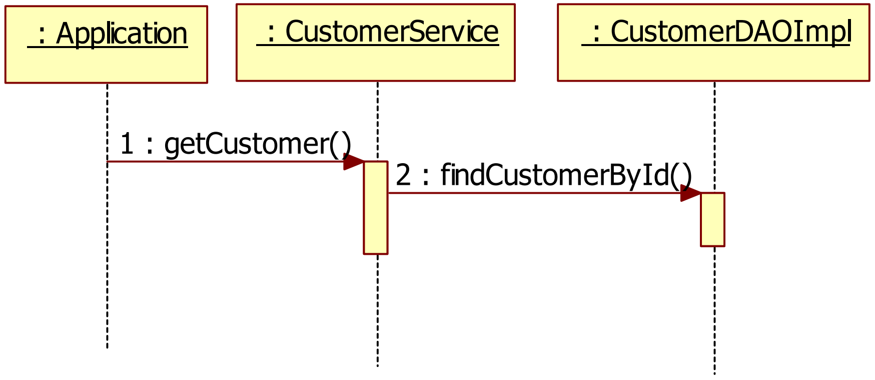


The proxy always implement the public interface of the RealSubject



Example without proxy

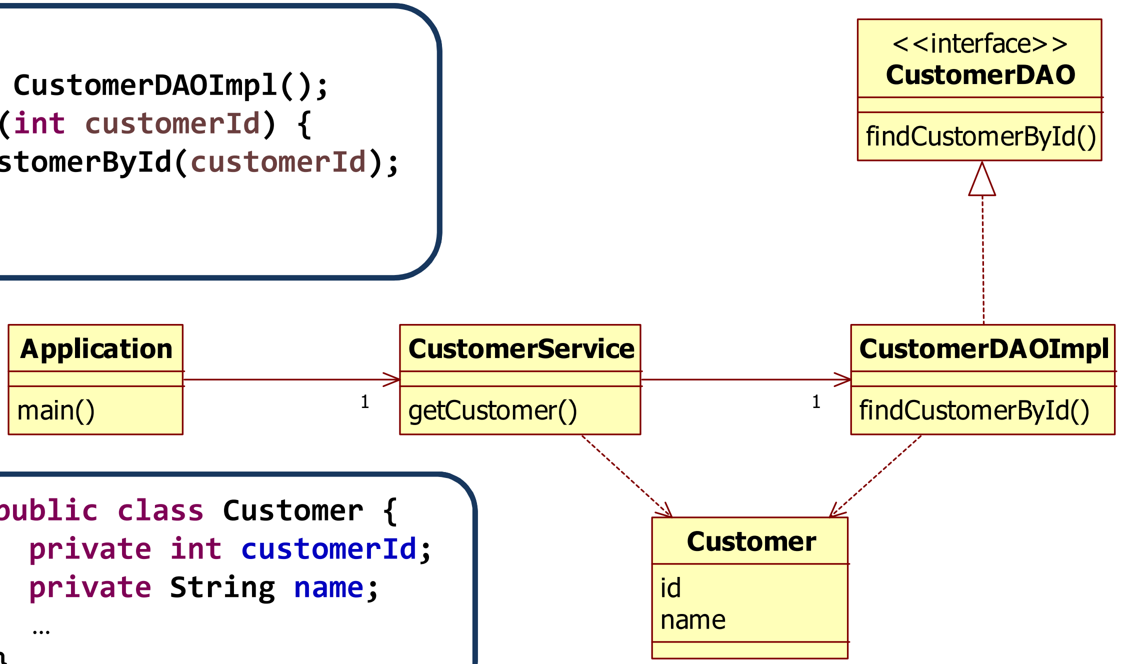




Example without proxy

```
public interface CustomerDAO {
  Customer findCustomerById(int customerId);
}
```

```
public class CustomerService {
  CustomerDAO customerDAO = new CustomerDAOImpl();
    public Customer getCustomer(int customerId) {
      return customerDAO.findCustomerById(customerId);
  }
}
```

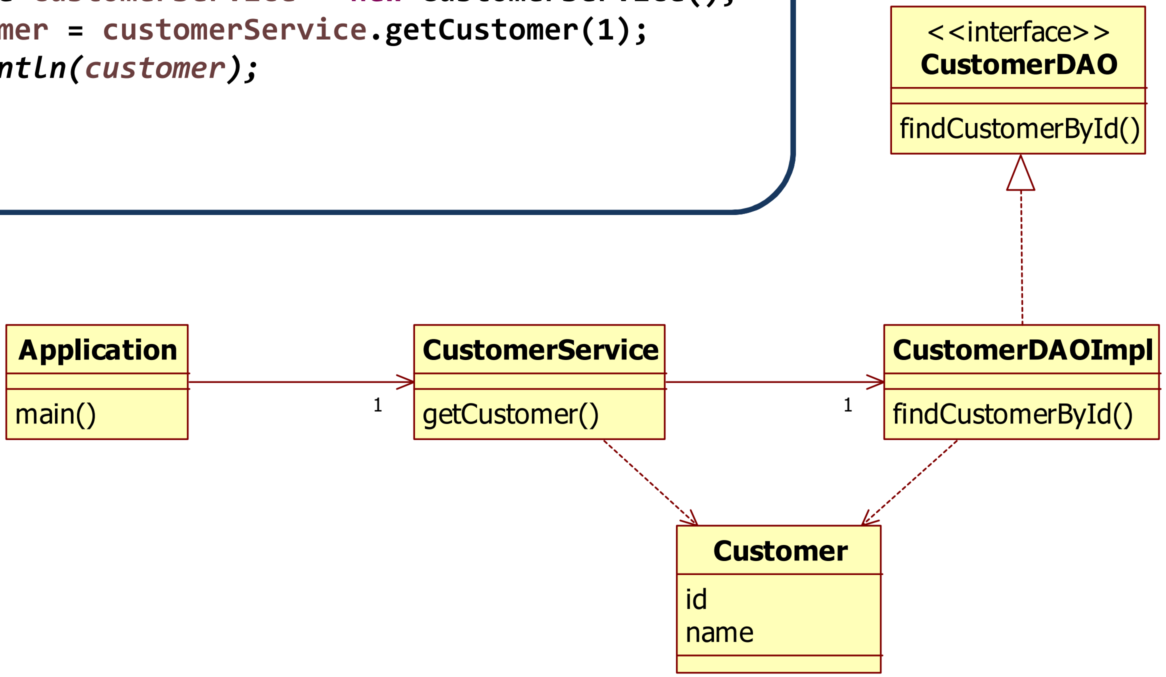


```
public class Customer {
  private int customerId;
  private String name;
  …
}
```

```
public class CustomerDAOImpl implements CustomerDAO{
  public Customer findCustomerById(int customerId) {
    return new Customer(customerId, "Frank Brown");
  }
}
```

Example without proxy: application

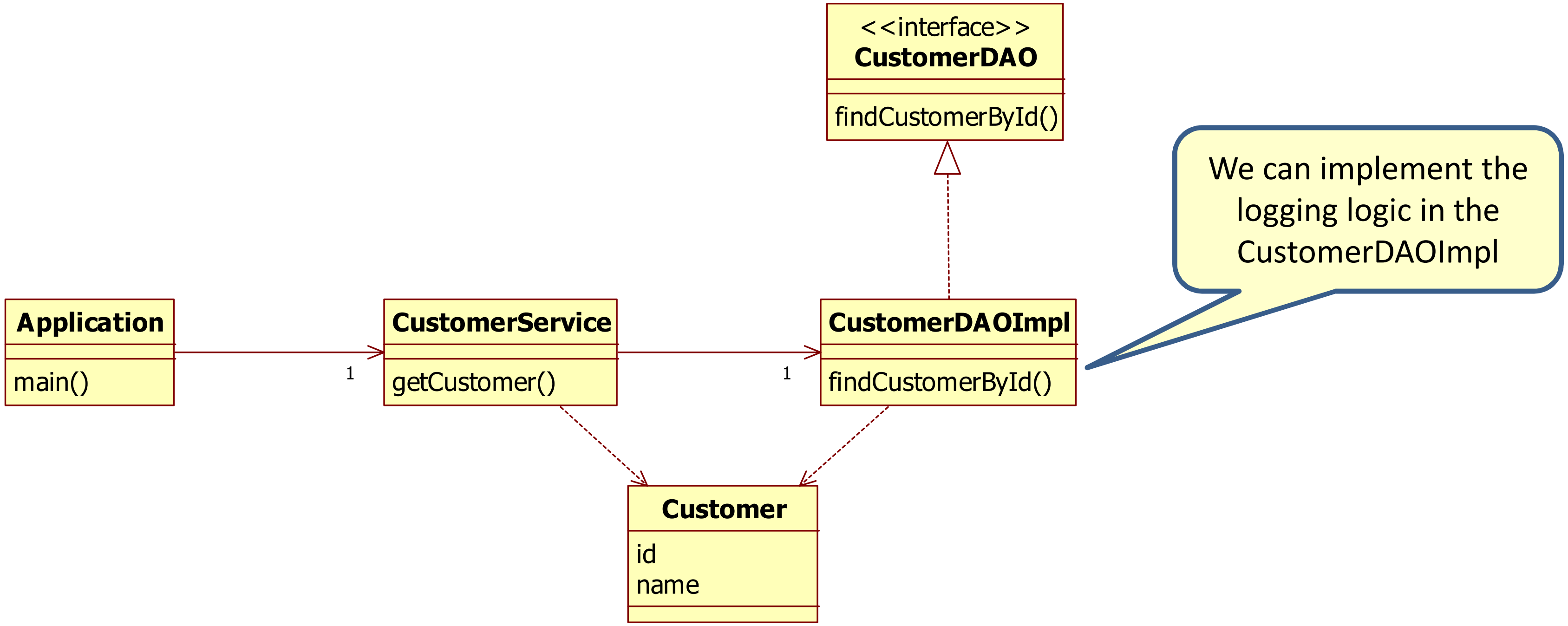
```java
public class Application {
  public static void main(String[] args) {
    CustomerService customerService = new CustomerService();
    Customer customer = customerService.getCustomer(1);
    System.out.println(customer);
  }
}
```



```
Customer [customerId=1, name=Frank Brown]
```

New requirement: logging

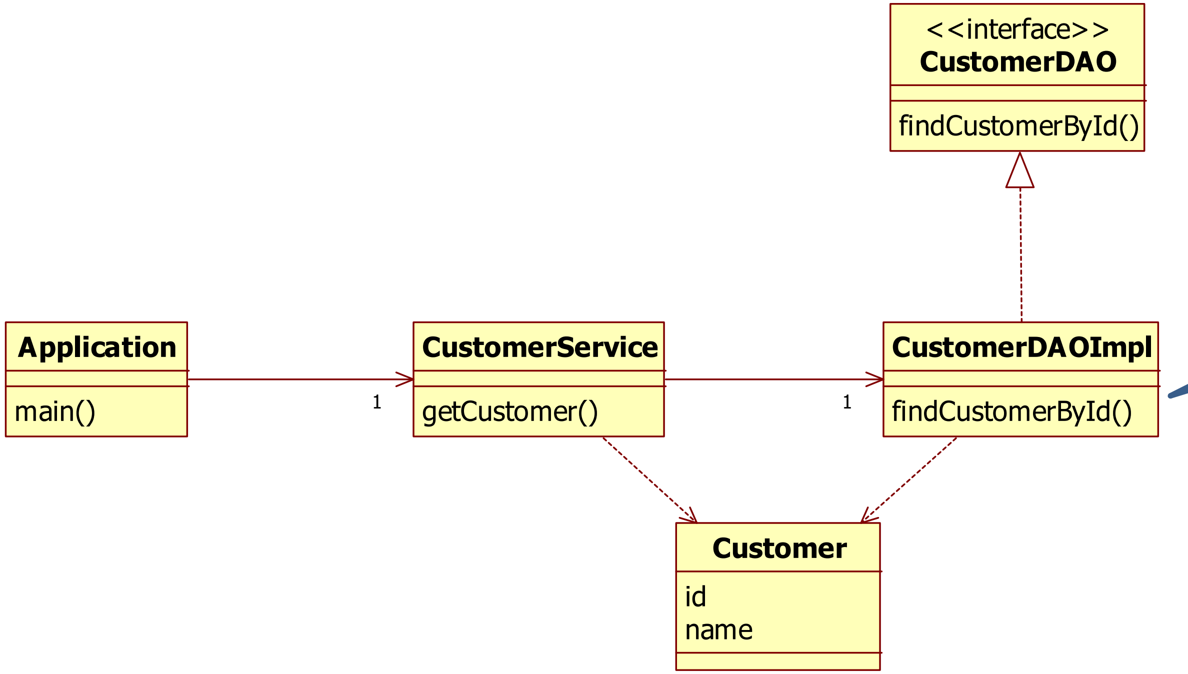
- For maintainability reasons we want to log every action on the database



We can implement the logging logic in the CustomerDAOImpl

New requirement: caching

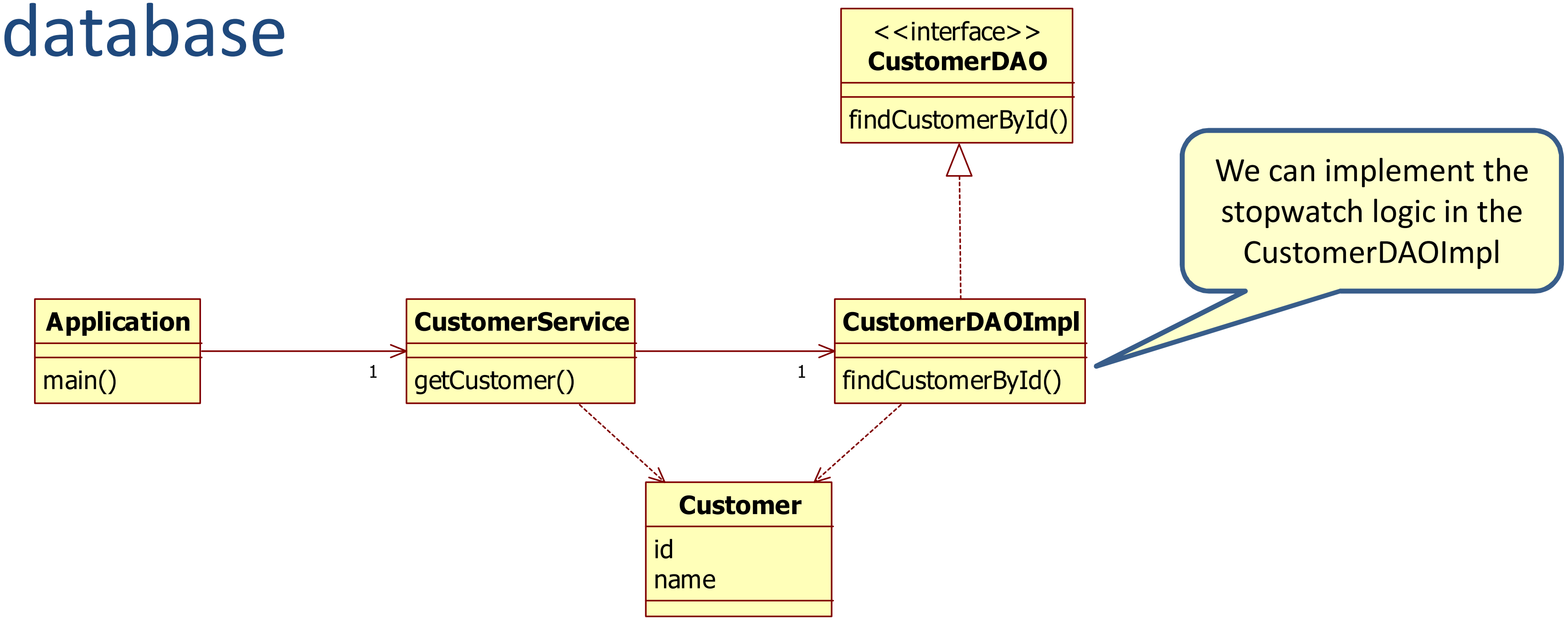
- For performance reasons we want to cache the Customers we retrieve from the database



We can implement the caching logic in the CustomerDAOImpl

New requirement: time measurement

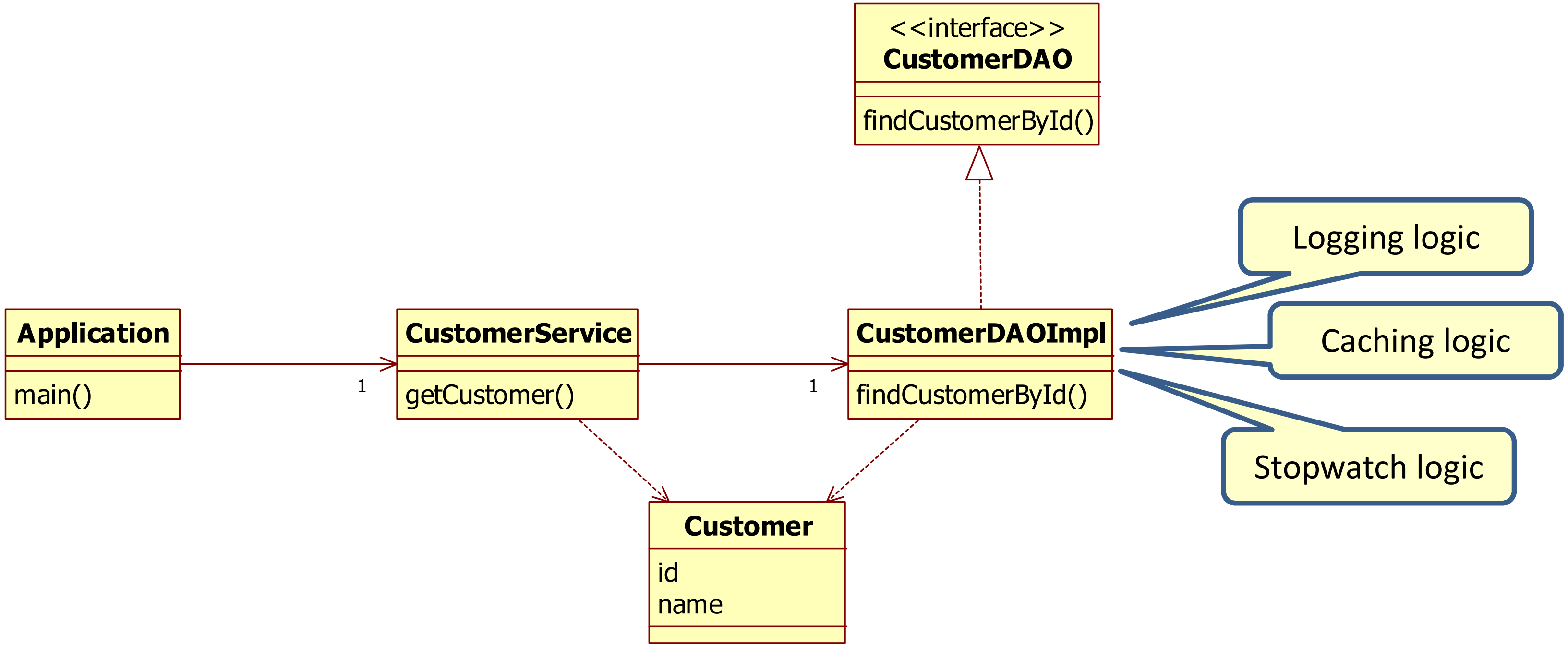
- For performance management reasons we want to measure the time of every call to the database



We can implement the stopwatch logic in the CustomerDAOImpl

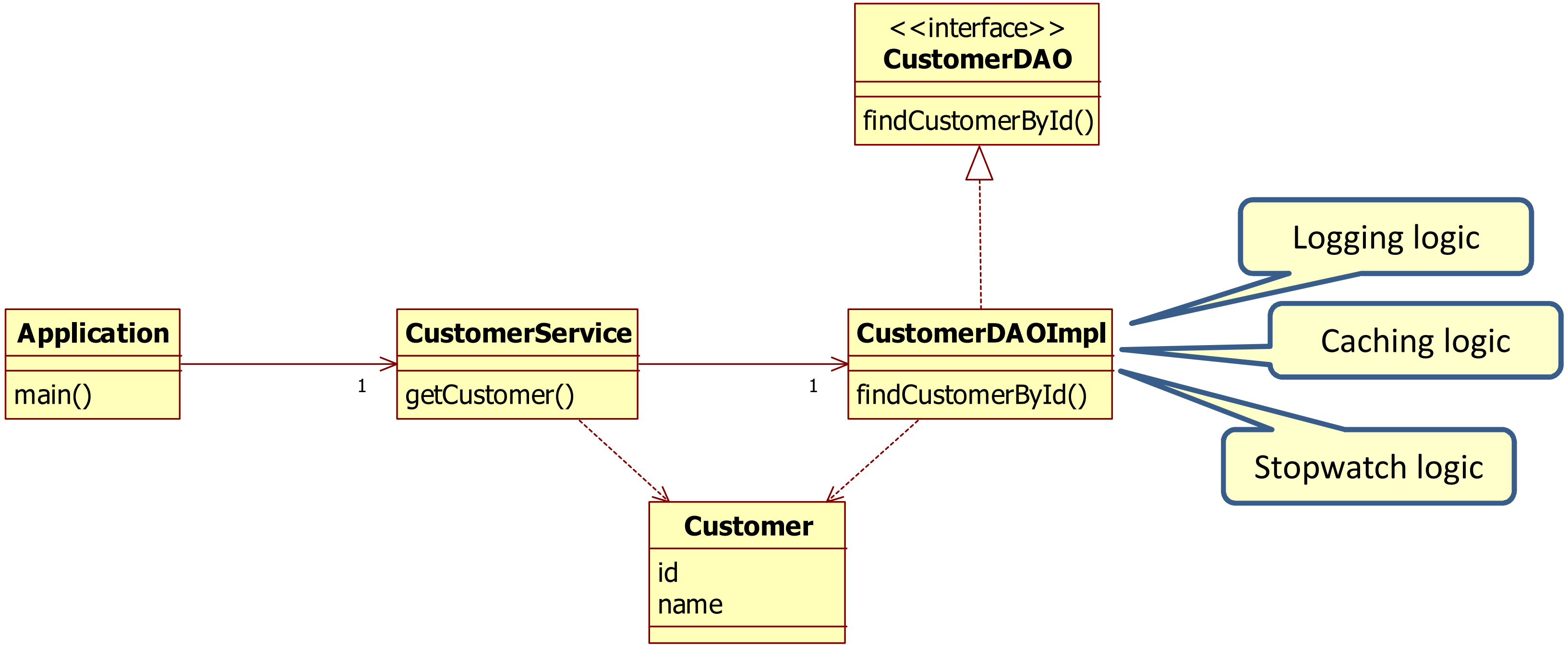
Single responsibility

- A class has one reason to change

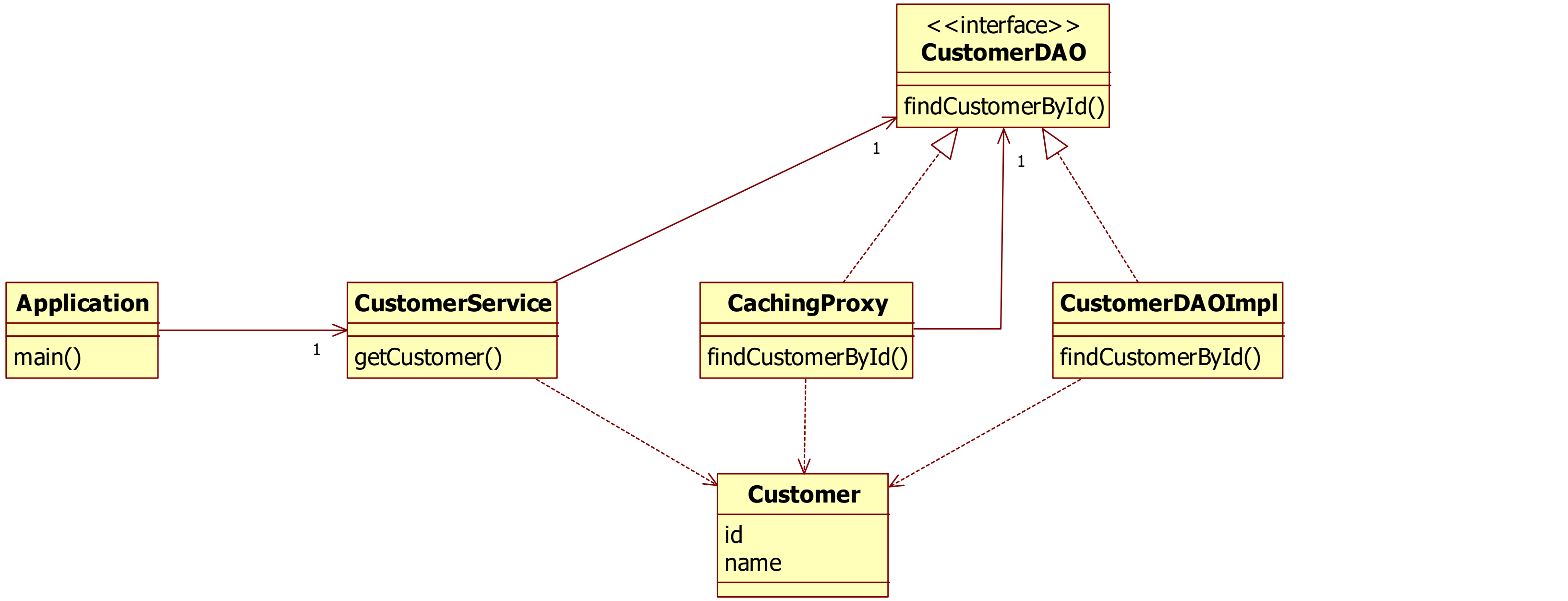


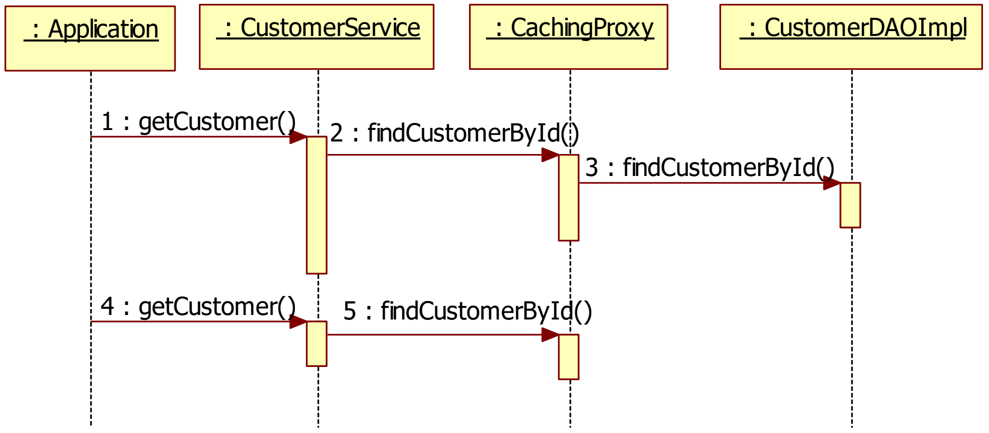
Open-closed principle

- Your design should be open for extension, but closed for change



Caching proxy



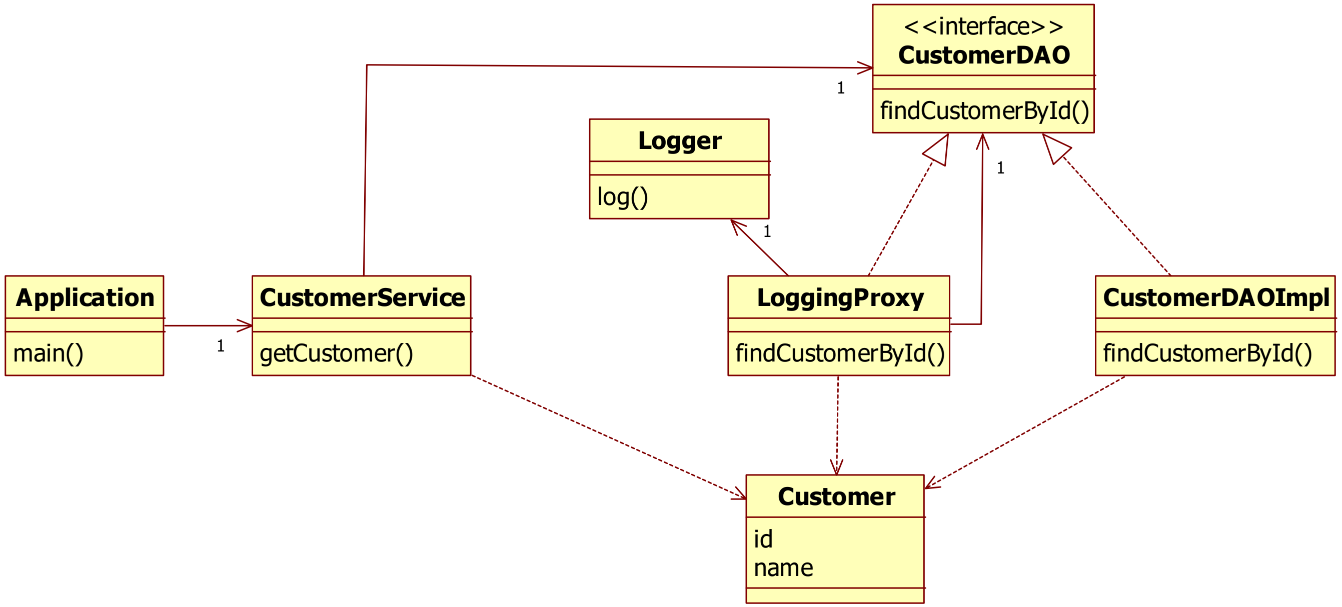


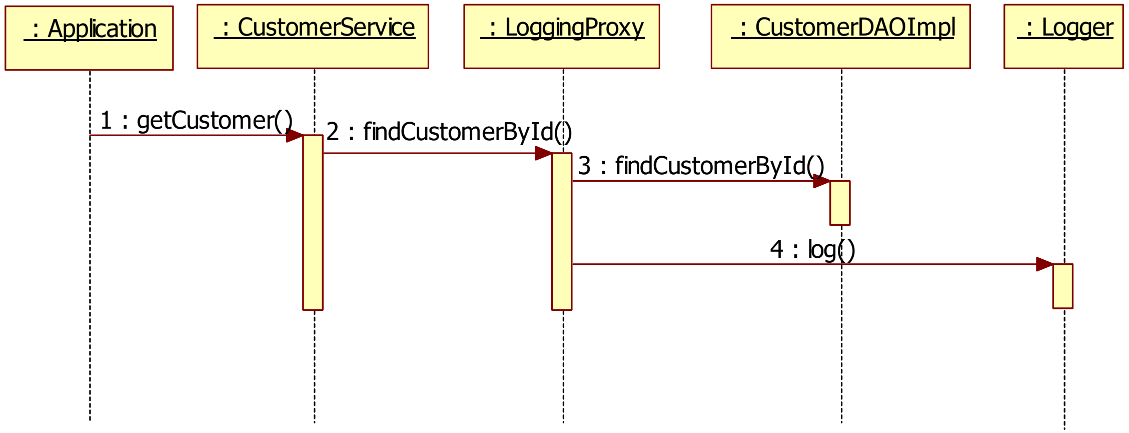
Caching proxy code

```java
public class CustomerService {
  CustomerDAO customerDAO = new CustomerDAOImpl();
  CustomerDAO cachingProxy = new CachingProxy(customerDAO);

  public Customer getCustomer(int customerId) {
    return cachingProxy.findCustomerById(customerId);
  }
}
```

```java
public class CachingProxy implements CustomerDAO{
  CustomerDAO customerDAO;
  Map<Integer,Customer> customerCache = new HashMap<Integer,Customer>();

  public CachingProxy(CustomerDAO customerDAO) {
    this.customerDAO = customerDAO;
  }
  public Customer findCustomerById(int customerId) {
    Customer cachedCustomer = customerCache.get(customerId);
    if (cachedCustomer == null) {
      Customer customer = customerDAO.findCustomerById(customerId);
      customerCache.put(customerId, customer);
      return customer;
    }
    else
      return cachedCustomer;
  }
}
```

Logging proxy





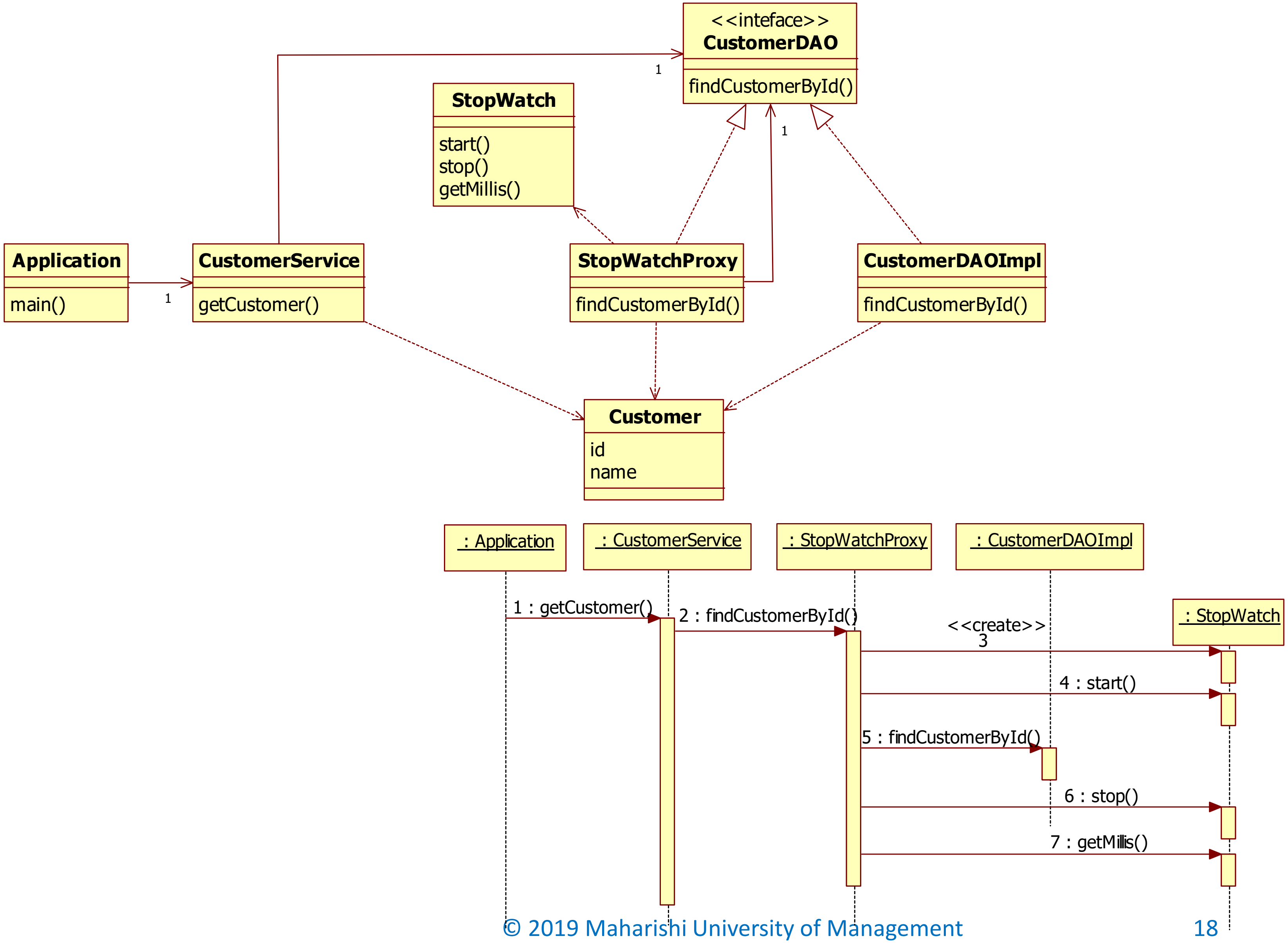
Logging proxy code

```java
public class CustomerService {
  CustomerDAO customerDAO = new CustomerDAOImpl();
  CustomerDAO loggingProxy = new LoggingProxy(customerDAO);

  public Customer getCustomer(int customerId) {
    return loggingProxy.findCustomerById(customerId);
  }
}
```

```java
public class LoggingProxy implements CustomerDAO {
  CustomerDAO customerDAO;
  Logger logger = new Logger();

  public LoggingProxy(CustomerDAO customerDAO) {
    this.customerDAO = customerDAO;
  }

  public Customer findCustomerById(int customerId) {
    Customer customer = customerDAO.findCustomerById(customerId);
    logger.log("getting customer with id= " + customerId);
    return customer;
  }
}
```

```java
public class Logger {
  public void log(String message) {
  System.out.println(message);
  }
}
```

Stopwatch proxy



StopWatch proxy code

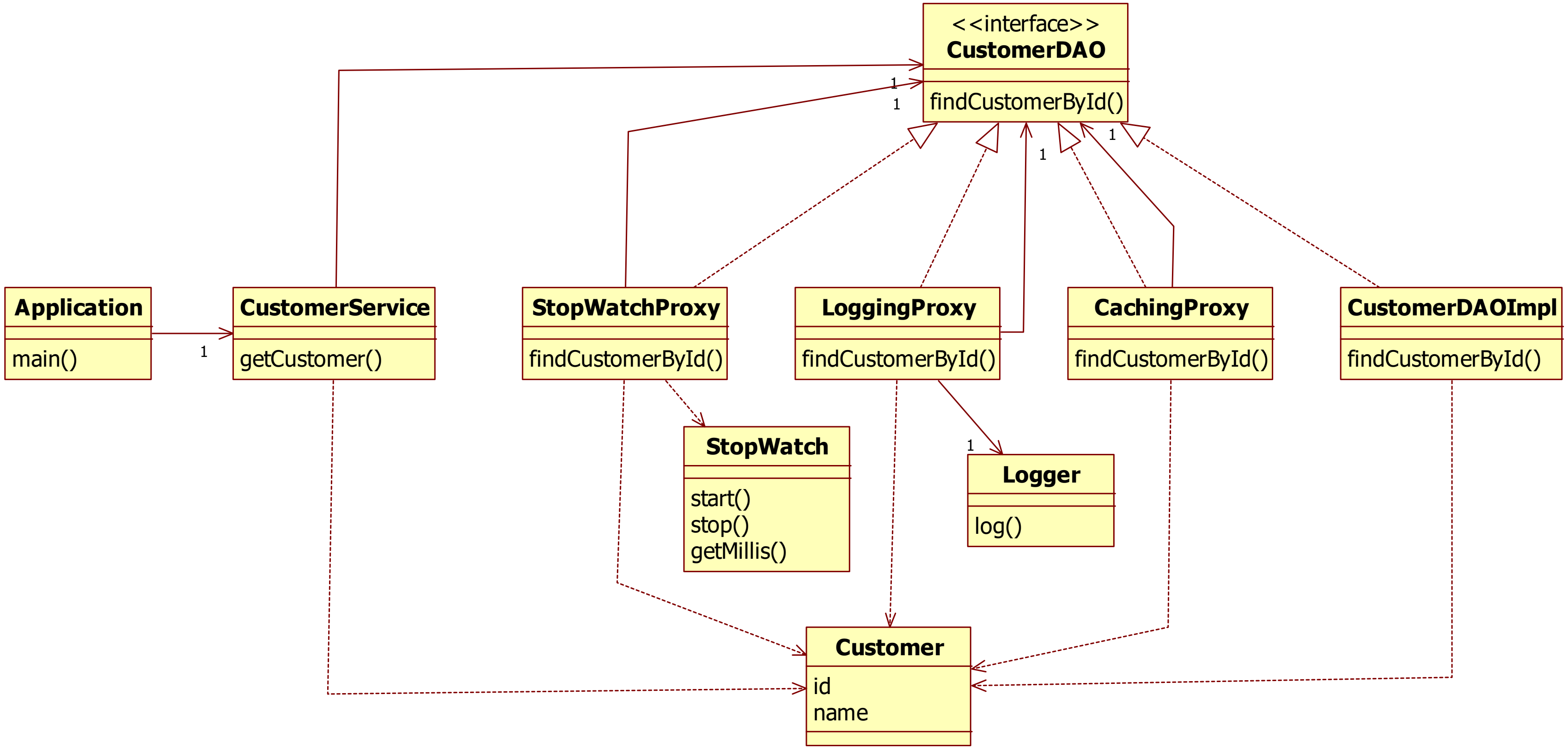
```java
public class CustomerService {
  CustomerDAO customerDAO = new CustomerDAOImpl();
  CustomerDAO stopWatchProxy = new StopWatchProxy(customerDAO);

  public Customer getCustomer(int customerId) {
    return stopWatchProxy.findCustomerById(customerId);
  }
}
```

```java
public class StopWatchProxy implements CustomerDAO {
  CustomerDAO customerDAO;

  public StopWatchProxy(CustomerDAO customerDAO) {
    this.customerDAO = customerDAO;
  }

  public Customer findCustomerById(int customerId) {
    StopWatch stopwatch = new StopWatch();
    stopwatch.start();
    Customer customer = customerDAO.findCustomerById(customerId);
    stopwatch.stop();
    System.out.println("The method CustomerDAO.getCustomer took
                       "+stopwatch.getMillis()+" ms");
    return customer;
  }
}
```

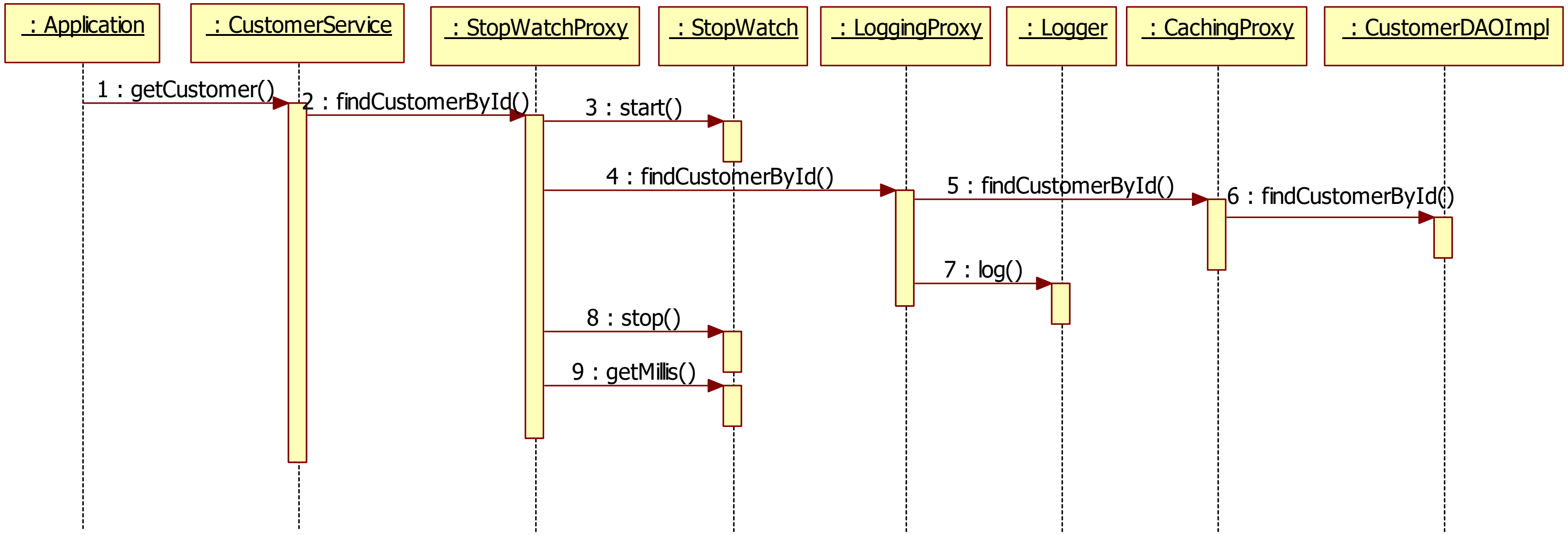
StopWatch code

```java
public class StopWatch {

  private long start = 0;
  private long finish = 0;
  private long timeElapsed = 0;

  public void start() {
    start = System.currentTimeMillis();
  }
  public void stop() {
    finish = System.currentTimeMillis();
  }
  public long getMillis() {
    timeElapsed = finish - start;
    return timeElapsed;
  }
}
```

Multiple proxies class diagram



Multiple proxies scenario



Nested proxies

```
public class CustomerService {
  CustomerDAO customerDAO = new CustomerDAOImpl();
  CustomerDAO cachingProxy = new CachingProxy(customerDAO);
  CustomerDAO loggerProxy = new LoggingProxy(cachingProxy);
  CustomerDAO stopWatchProxy = new StopWatchProxy(loggerProxy);

  public Customer getCustomer(int customerId) {
    return stopWatchProxy.findCustomerById(customerId);
  }
}
```

Creating a chain of nested proxies

Problem with simple proxy

```
public class LoggingProxy implements CustomerDAO {
  CustomerDAO customerDAO;
  Logger logger = new Logger();

  public LoggingProxy(CustomerDAO customerDAO) {
    this.customerDAO = customerDAO;
  }

  public Customer findCustomerById(int customerId) {
    Customer customer = customerDAO.findCustomerById(customerId);
    logger.log("getting customer with id= " + customerId);
    return customer;
  }
}
```

This proxy can only be used in front of classes that implement the CustomerDAO interface

- We want a generic proxy that can be used in front of any class
  - Dynamic proxy

Dynamic stopwatch proxy

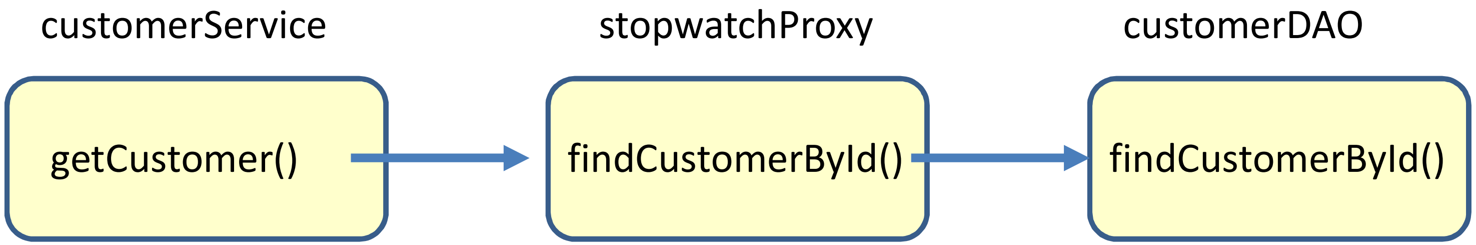
```java
import java.lang.reflect.*;

public class StopWatchProxy implements InvocationHandler {
  private Object target;

  public StopWatchProxy(Object target) {
    this.target = target;
  }

  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    StopWatch stopwatch = new StopWatch();
    stopwatch.start();
    // invoke the method on the target
    Object returnValue = method.invoke(target, args);

    stopwatch.stop();
    System.out.println("The method " + method.getName() + " took " + stopwatch.getMillis() + " ms");
    return returnValue;
  }
}
```

Reflection: A technique to examine or modify the behavior of methods, classes, interfaces at runtime.

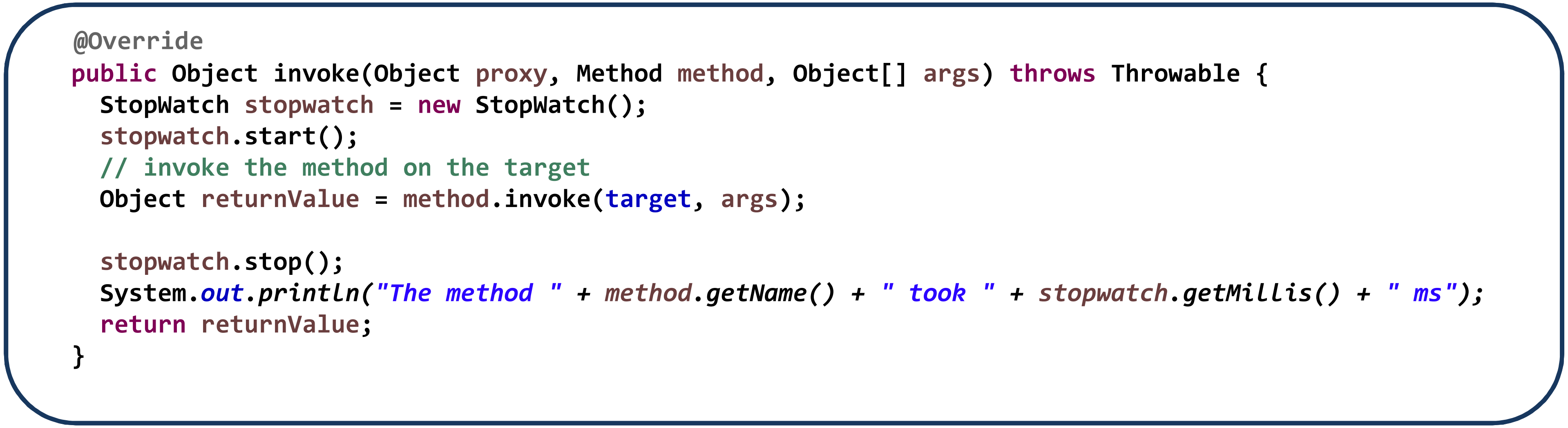
Invoking the dynamic proxy

```
import java.lang.reflect.Proxy;

public class CustomerService {
  CustomerDAO customerDAO = new CustomerDAOImpl();
  ClassLoader classLoader = CustomerDAO.class.getClassLoader();
  CustomerDAO stopWatchProxy = (CustomerDAO)
         Proxy.newProxyInstance(classLoader,
                                 new Class[] { CustomerDAO.class },
                                 new StopWatchProxy(customerDAO));


    public Customer getCustomer(int customerId) {
      return stopWatchProxy.findCustomerById(customerId);
    }
}
```

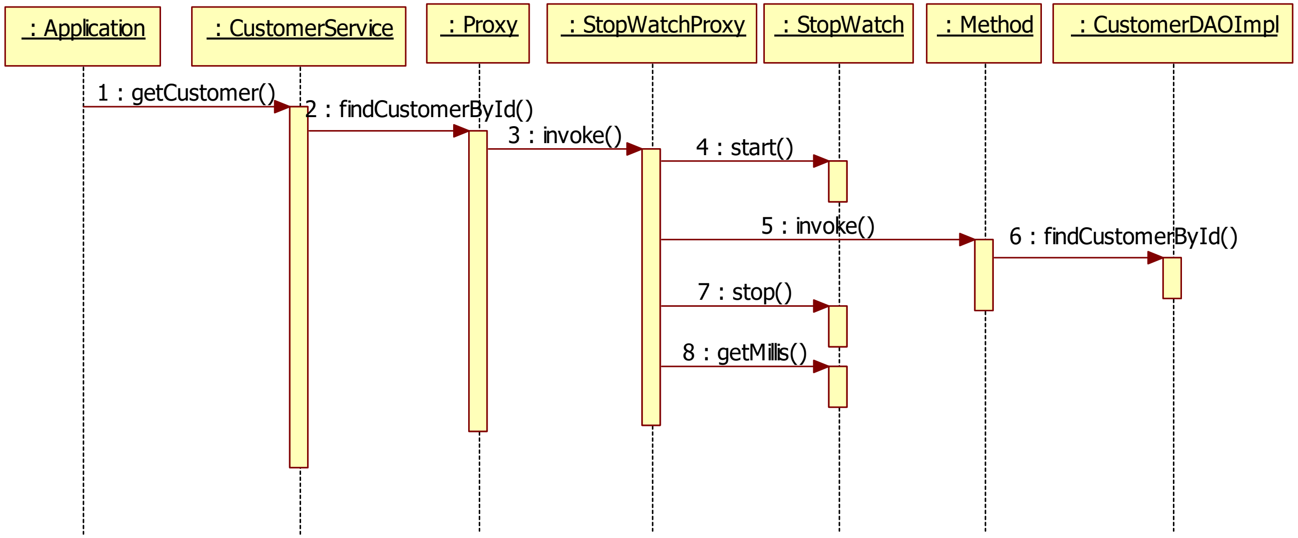
Create a Proxy



What really happens



```
@Override
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
  StopWatch stopwatch = new StopWatch();
  stopwatch.start();
  // invoke the method on the target
  Object returnValue = method.invoke(target, args);

  stopwatch.stop();
  System.out.println("The method " + method.getName() + " took " + stopwatch.getMillis() + " ms");
  return returnValue;
}
```



Proxy vs. dynamic proxy

```java
public class StopWatchProxy implements CustomerDAO {
  CustomerDAO customerDAO;

  public StopWatchProxy(CustomerDAO customerDAO) {
    this.customerDAO = customerDAO;
  }

  public Customer findCustomerById(int customerId) {
    StopWatch stopwatch = new StopWatch();
    stopwatch.start();
    Customer customer = customerDAO.findCustomerById(customerId);
    stopwatch.stop();
    System.out.println("The method CustomerDAO.getCustomer took
                       "+stopwatch.getMillis()+" ms");
    return customer;
  }
}
```

This proxy can only be used in front of classes that implement the CustomerDAO interface

You need to implement all interface methods

Is very specific in what you want to do on a CustomerDAO class

Proxy vs. dynamic proxy

```java
public class StopWatchProxy implements InvocationHandler {
  private Object target;

  public StopWatchProxy(Object target) {
    this.target = target;
  }

  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    StopWatch stopwatch = new StopWatch();
    stopwatch.start();
    // invoke the method on the target
    Object returnValue = method.invoke(target, args);

    stopwatch.stop();
    System.out.println("The method " + method.getName() + " took " + stopwatch.getMillis() + " ms");
    return returnValue;
  }
}
```

This proxy can be used in front of every class

You only need to implement the invoke() method

Must be very generic if you want to apply this proxy for any other class

Dynamic logging proxy

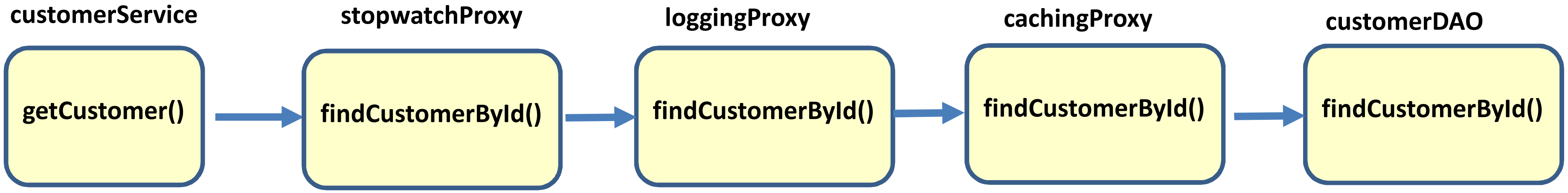
```
public class LoggingProxy implements InvocationHandler {
  private Object target;
  Logger logger = new Logger();

  public LoggingProxy(Object target) {
    this.target = target;
  }

  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    // invoke the method on the target
    Object returnValue = method.invoke(target, args);
    logger.log("Calling method" + method.getName() + " with argument(s):");
    for(int p=0; p<args.length;p++){
      logger.log(" Param[" + p + "]: " + args[p].toString());
    }
    return returnValue;
  }
}
```

Dynamic caching proxy

```java
public class CachingProxy implements InvocationHandler {
  private Object target;
  Map<String, Object> cache = new HashMap<String, Object>();

  public CachingProxy(Object target) {
   this.target = target;
  }

  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    String key = ""+args[0];
    Object cachedObject = cache.get(key);
    if (cachedObject == null) {
      Object result = method.invoke(target, args);
      cache.put(key, result);
      return result;
    } else
      return cachedObject;
  }
}
```

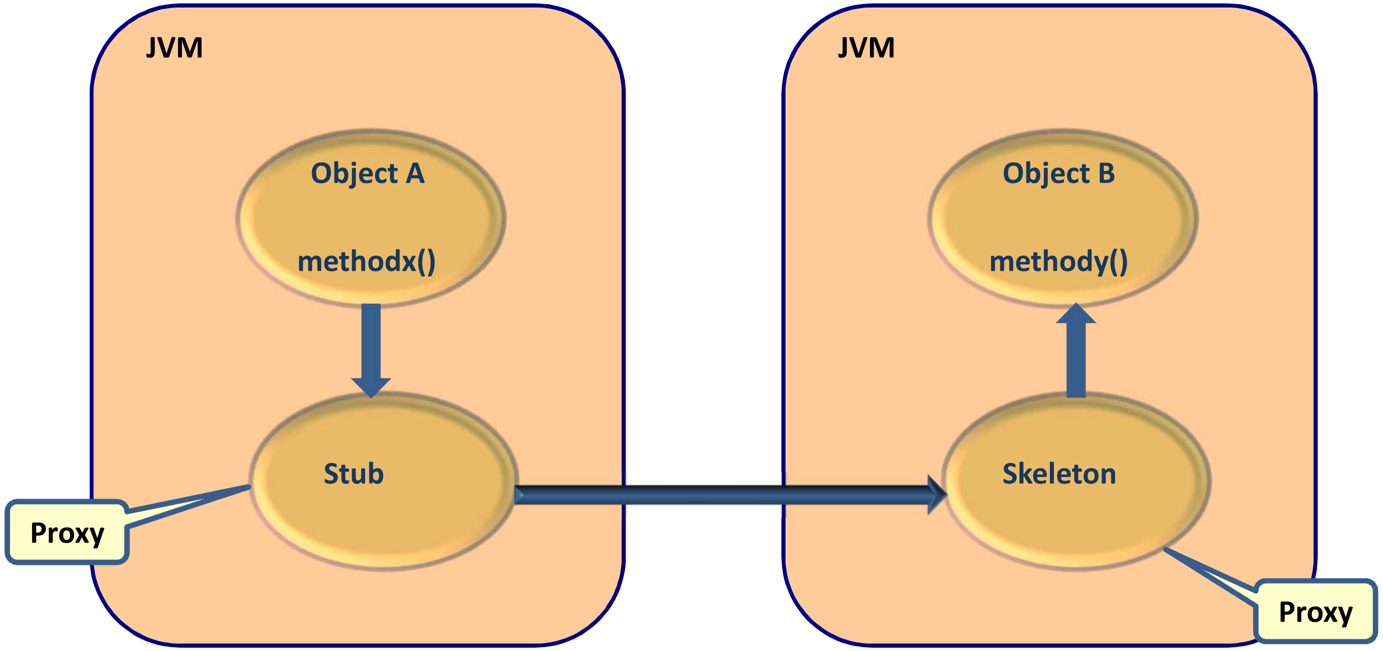
Nested dynamic proxies

```
public class CustomerService {
  CustomerDAO customerDAO = new CustomerDAOImpl();
  ClassLoader classLoader = CustomerDAO.class.getClassLoader();
  CustomerDAO cachingProxy =
    (CustomerDAO) Proxy.newProxyInstance(classLoader,
                                         new Class[] { CustomerDAO.class },
                                         new CachingProxy(customerDAO));
  CustomerDAO loggingProxy =
    (CustomerDAO) Proxy.newProxyInstance(classLoader,
                                         new Class[] { CustomerDAO.class },
                                         new LoggingProxy(cachingProxy));
  CustomerDAO stopwatchProxy =
    (CustomerDAO) Proxy.newProxyInstance(classLoader,
                                         new Class[] { CustomerDAO.class },
                                         new StopWatchProxy(loggingProxy));

  public Customer getCustomer(int customerId) {
    return stopwatchProxy.findCustomerById(customerId);
  }
}
```



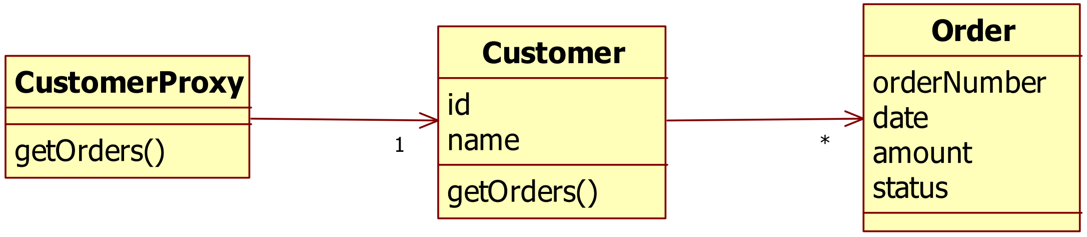
Where are proxies used: RPC

- Remote Procedure Calls
  - Remote Method Invocation (RMI)



Where are proxies used: Hibernate

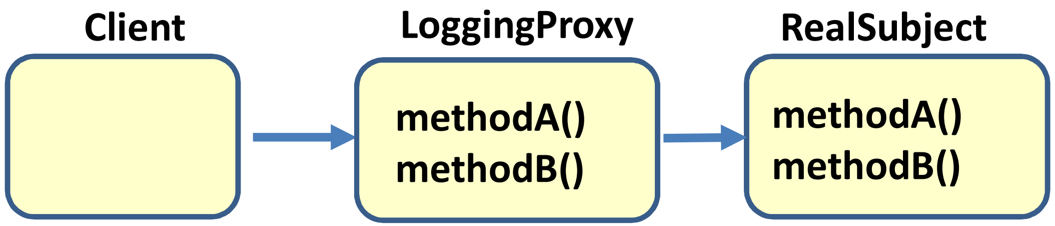
- Hibernate is an Object Relational Mapper (ORM) framework used for persisting objects
- It uses lazy loading using proxies

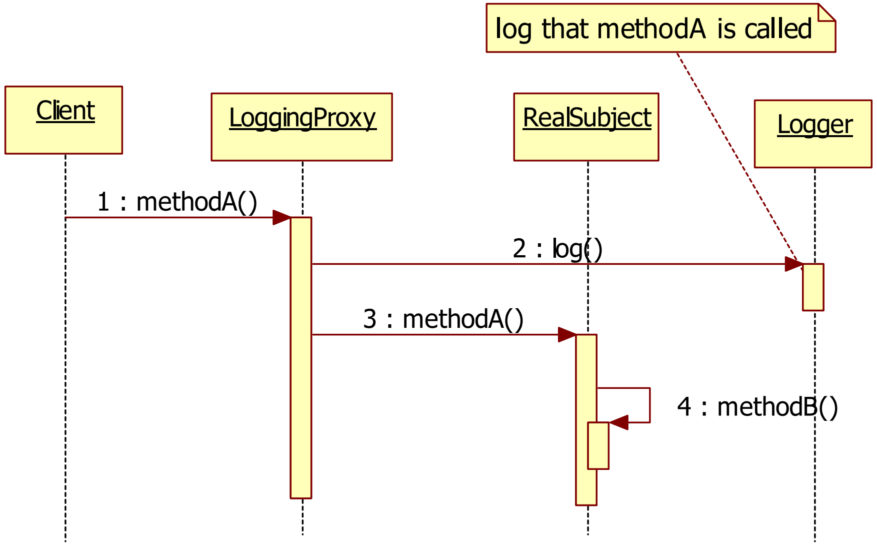


For performance reasons is the to-many relationship between Customer and Order lazy loaded

Issue with a proxy

- If a method of the real subject calls a method of itself, this will not go through the proxy.





Main point

- The Proxy pattern provides a surrogate or placeholder for another object to control access to it.

- In Unity Consciousness one realizes that every relative object you see around you, is just an expression of the same Pure Consciousness you experience within yourself.