Singleton Pattern

CS356 Object-Oriented Design and Programming
http://cs356.yusun.io
October 27, 2014

Yu Sun, Ph.D.
http://yusun.io
yusun@csupomona.edu

CAL POLY POMONA

GoF Form of a Design Pattern

- The Pattern Name
  - Pattern name and classification, Intent, and Also-Known-As
- The Problem
  - Motivation, and Applicability
- The Solution
  - Structure (graphical), Participants (their classes/ objects/ responsibilities), Collaborations (of the participants), Implementation (hints, techniques), Sample code, Known uses, and Related patterns
- The Consequences
  - Consequences (trade-offs, concerns)

Creational Patterns

- Concerns the process of object creation

- Will cover
  - Abstract Factory / Factory Method
  - Singleton

- Will not cover
  - Builder
  - Prototype

Structural Patterns

- Deals with composition of classes or objects

- Will cover
  - Adapter
  - Façade
  - Composite
  - Decorator
  - Proxy
- Will not cover
  - Bridge
  - Flyweight

Behavioral Patterns

- Characterizes the ways in which classes or objects interact and distribute responsibility

- Will cover
  - Visitor
  - Observer
  - Strategy
  - Command
  - Chain of Responsibility

- Will not cover
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - State
  - Template

Creational Patterns

- Abstract the instantiation process
  - Make a system independent of how its objects are created, composed, and represented
- Important if systems evolve to depend more on object composition than on class inheritance
  - Emphasis shifts from hardcoding fixed sets of behaviors towards a smaller set of composable fundamental behaviors
- Encapsulate knowledge about the concrete classes that a system uses
- Hide how instances of classes are created and put together

Basic Definitions

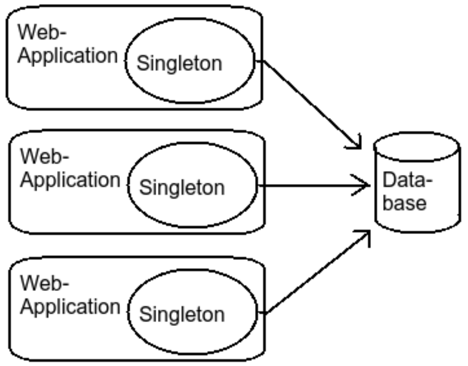
- Instantiation
  - The creation of an object from a class
- Abstract Class
  - Defines a common interface for its subclasses
  - Defers some implementation to its subclasses
  - Cannot be instantiated
- Concrete Class
  - Classes which can be instantiated

Singleton Pattern

Singleton

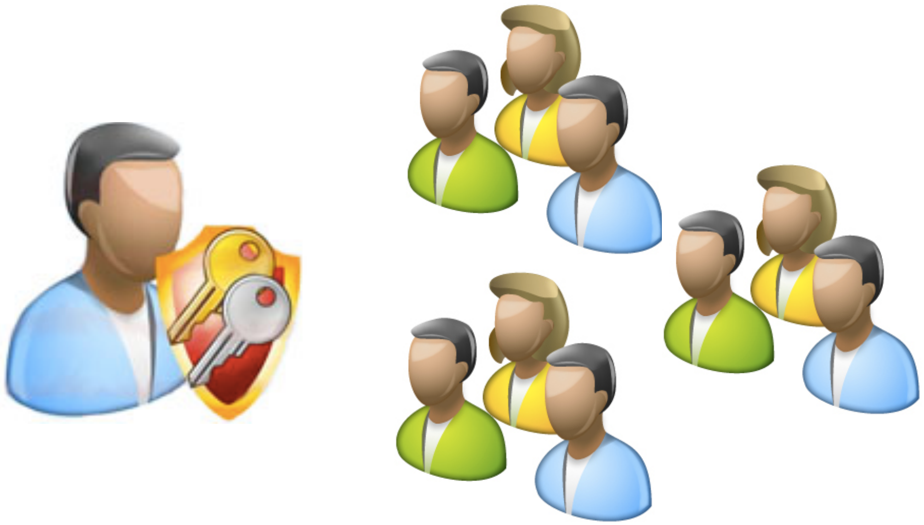
- Intent
  - Ensure a class has only one instance and provide a global point of access to it; class itself is responsible for sole instance
- Applicability
  - Want exactly one instance of a class
  - Accessible to clients from one point
  - Can also allow a countable number of instances
  - Global namespace provides a single object, but does not prevent other objects of the class from being instantiated

When do we need a Singleton?



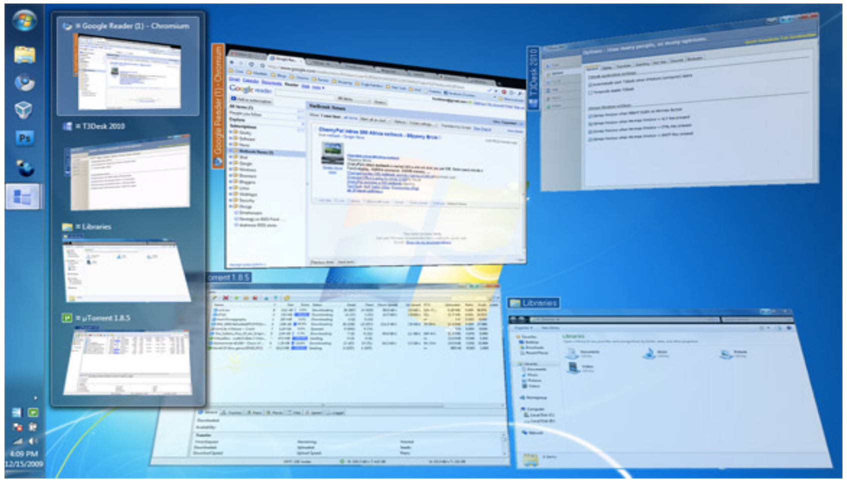
Database Connection

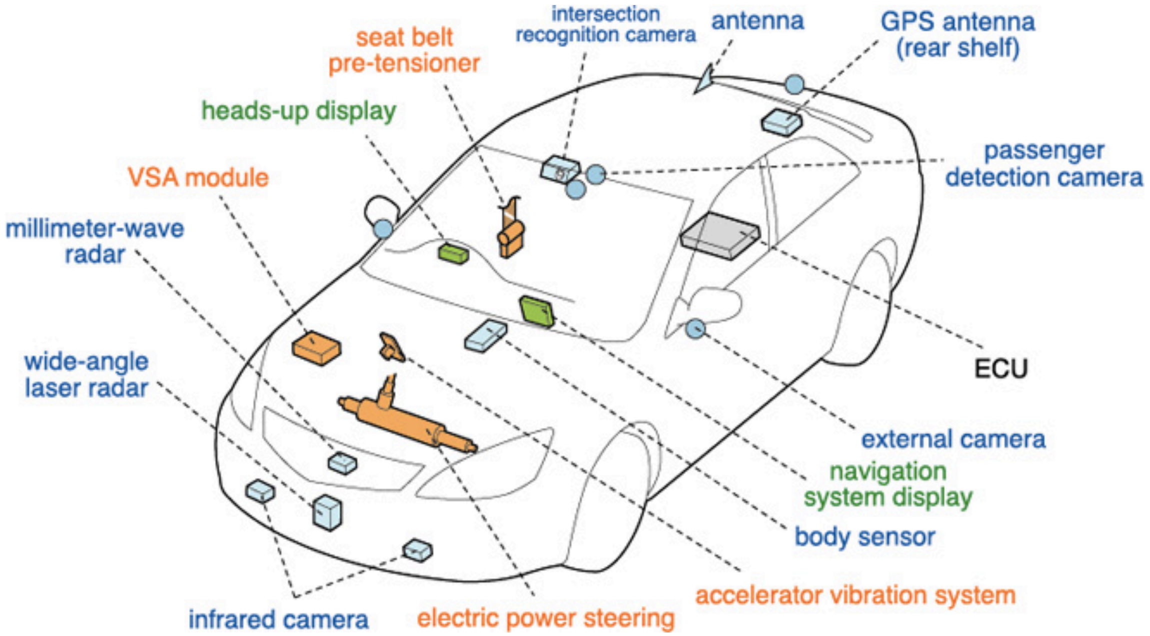
Camera API Object

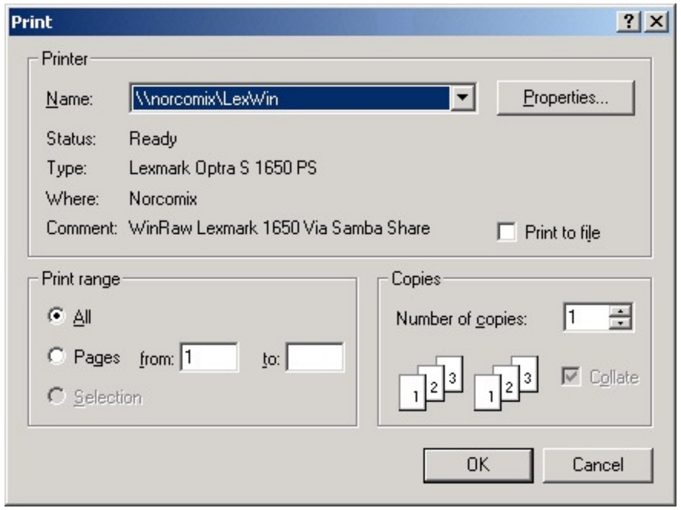


User Account Management

When do we need a Singleton?


Window Manager Object




Print

Printer

Name: \\norcomix\LexWin

Status: Ready

Type: Lexmark Optra S 1650 PS

Where: Norcomix

Comment: WinRaw Lexmark 1650 Via Samba Share

☐ Print to file

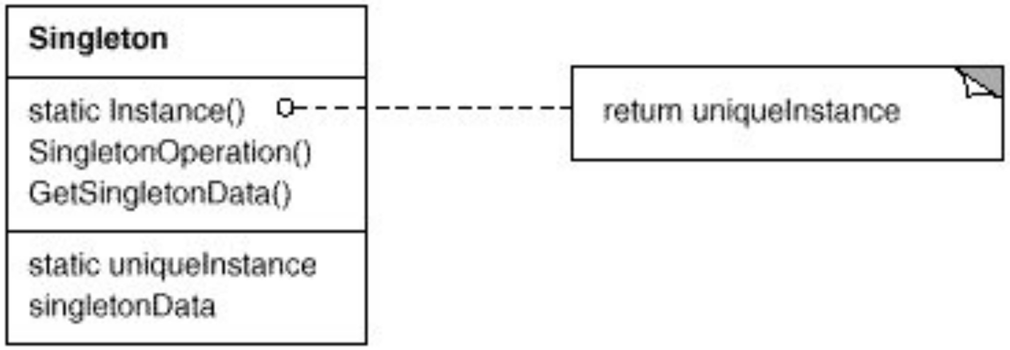
Print range: [✓ All] / Pages from: 1 to: / Selection

Copies: Number of copies: 1 — ☑ Collate

Printing Manager Object

Participants and Collaborations

- Singleton
  - Defines an `getInstance` method that becomes the single "gate" by which clients can access its unique instance.
    - `getInstance` is a class method (static method)
  - May be responsible for creating its own unique instance
  - Constructor placed in private/protected section
- Clients access Singleton instances solely through the `getInstance` method



Implementation: Ensuring a Unique Instance

```
public class Singleton {
  private static final Singleton instance = new Singleton();

  private Singleton() {}

  public static Singleton getInstance() {
    return instance;
  }
}
```

Implementation: Lazy Instantiation

```
public class Singleton {
  private static Singleton instance = null;

  private Singleton() {}

  public static Singleton getInstance() {
    if(instance == null) {
      instance = new Singleton();
    }
    return instance;
  }
}
```

What if there are subclasses?

```
public abstract class MazeFactory {

  private static MazeFactory instance = null;

  private MazeFactory() {}

  public static MazeFactory getInstance() {
    if (instance == null)
      return getInstance("enchanted"); // default instance
    else
      return instance;
  }

  public static MazeFactory getInstance(String name) {
    if(instance == null)
      if (name.equals(“bombed"))
        instance = new BombedMazeFactory();
      else if (name.equals(“enchanted"))
        instance = new EnchantedMazeFactory();

    return instance;
  }
}
```

Singleton with Subclasses

- Client code to create factory the first time

```
MazeFactory factory = MazeFactory.getInstance(“bombed");
```

- Client code to access the factory

```
MazeFactory factory = MazeFactory.getInstance();
```

- To add another subclass requires changing the instance() method!
- Constructors of BombedMazeFactory and EnchantedMazeFactory can not be private

Singleton with Subclasses (ver. 2)

```
public class EnchantedMazeFactory extends MazeFactory {

  private EnchantedMazeFactory() {}

  public static MazeFactory getInstance() {
    if(instance == null)
      instance = new EnchantedMazeFactory();

    return instance;
  }
}
```

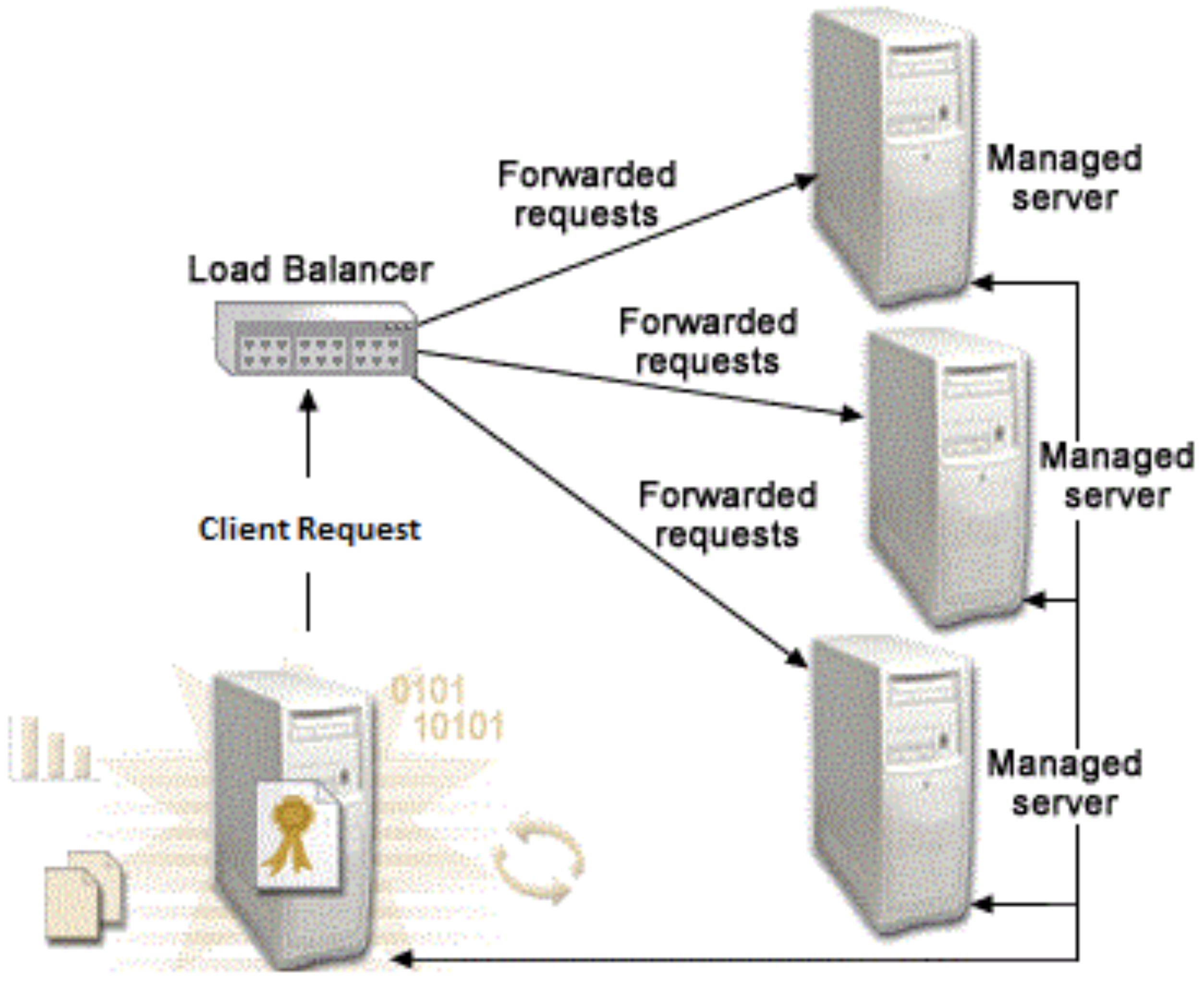
- Client code to create factory the first time

```
MazeFactory factory = EnchantedMazeFactory.getInstance();
```

- Client code to access the factory

```
MazeFactory factory = MazeFactory.getInstance();
```

Singleton Example – Load Balancer


Load Balancer
Forwarded requests
Forwarded requests
Forwarded requests
Managed server
Managed server
Managed server
Client Request
0101
10101