Singleton Pattern

CS480 Software Engineering
http://cs480.yusun.io
February 20, 2015

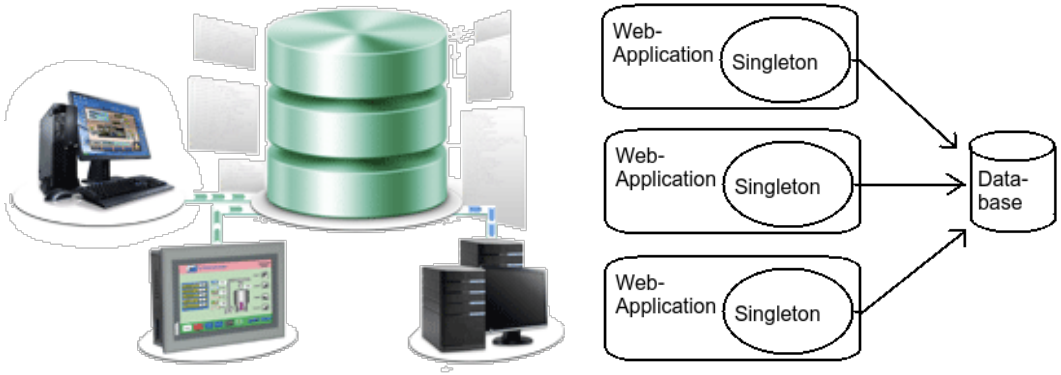
Yu Sun, Ph.D.
http://yusun.io
yusun@cpp.edu

CAL POLY POMONA

Singleton

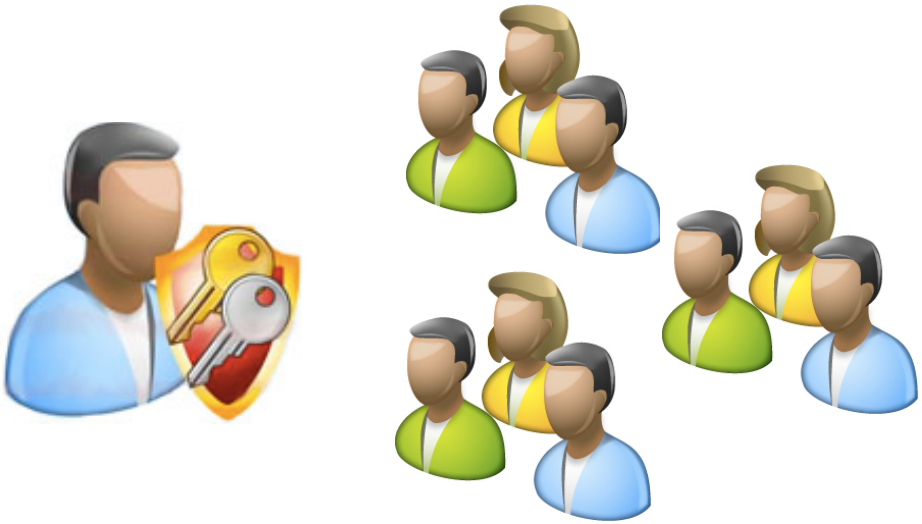
- Intent
  - Ensure a class has only one instance and provide a global point of access to it; class itself is responsible for sole instance
- Applicability
  - Want exactly one instance of a class
  - Accessible to clients from one point
  - Can also allow a countable number of instances
  - Global namespace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

When do we need a Singleton?



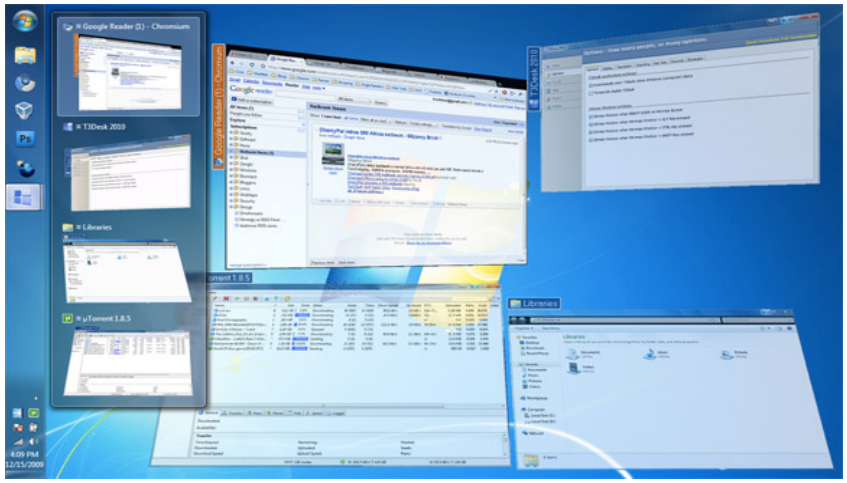
Database Connection

Camera API Object

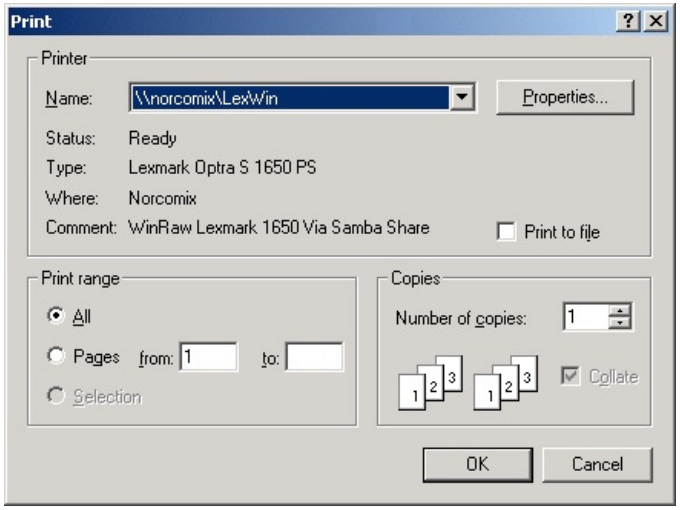


User Account Management

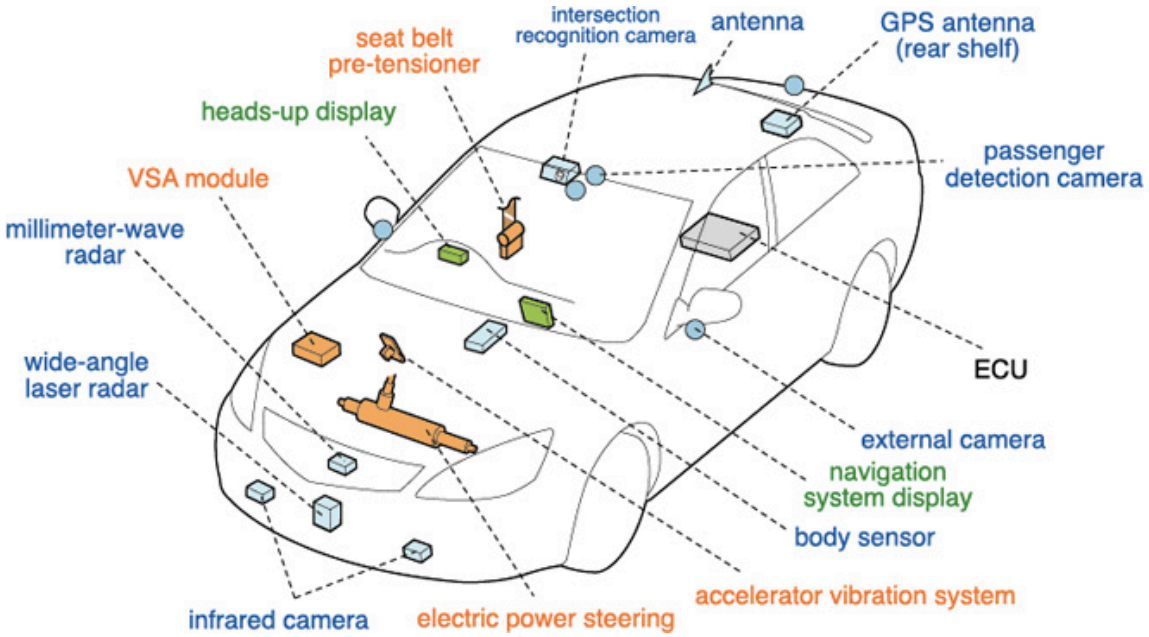
When do we need a Singleton?



Window Manager Object

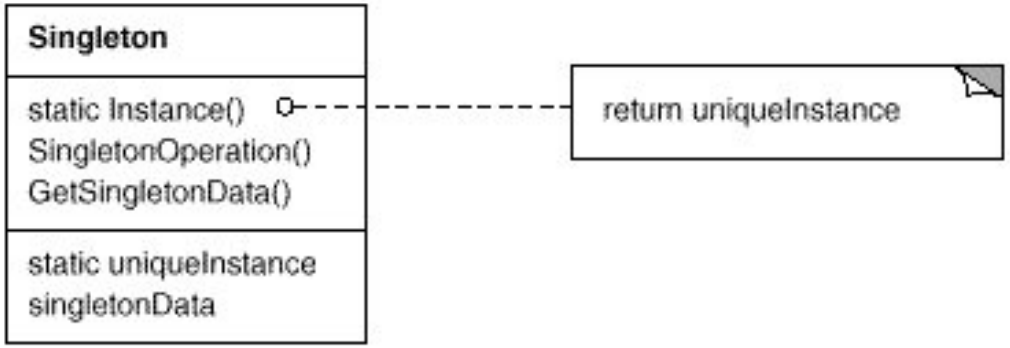


Printing Manager Object



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();
```