Design Patterns

宋 杰
Song Jie
东北大学 软件学院
Software College, Northeastern University

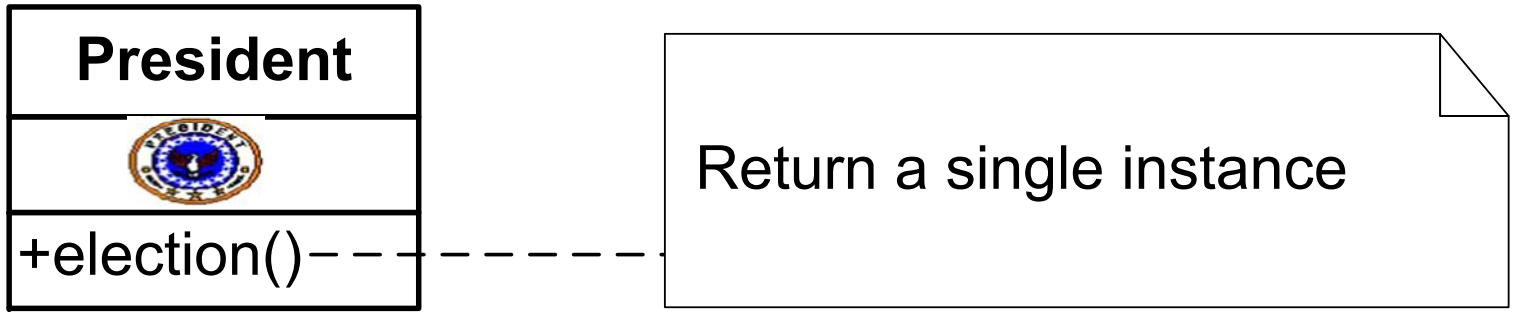
3. Singleton Pattern

Intent

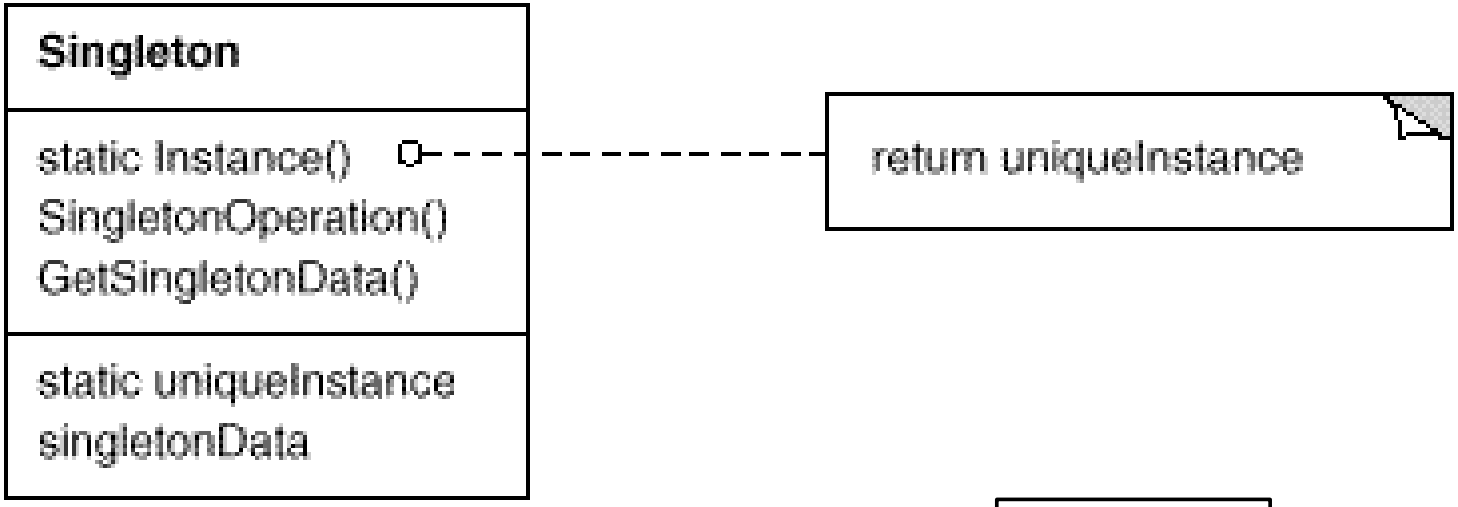
- Ensure a class only has one instance, and provide a global point of access to it.
  - Singleton should have one and only one instance;
  - Singleton should create the instance himself;
  - Singleton should provide an approach to access the instance.

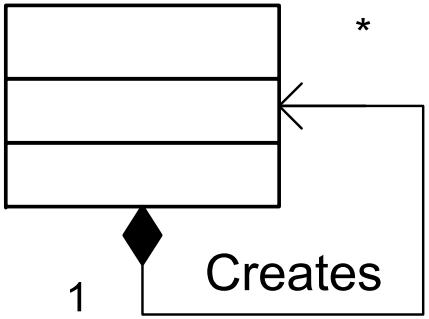
Intent

- A country should have one president;



Structure


Singleton
static Instance()
SingletonOperation()
GetSingletonData()
static uniqueInstance
singletonData
return uniqueInstance


*
1
Creates

Participants

- **Singleton**
  - Be responsible for creating its own unique instance.
  - Defines an static instance method that lets clients access its unique instance.

Consequences

- Controlled access to sole instance;
- More flexible than static class (class with all static properties and methods).
  - Static class must be stateless; Singleton could be stateful.

Applicability

- There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
- On the contrary, if a system allow multiple instances, it is unnecessary to use singleton.
- DO NOT reduce the number of instances for using singleton
  - Connection object of database
  - Printer in an system
  - Utilized (Tools) class

Implementation: Eager Singleton

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

    private Singleton() {
    }

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

Implementation: Lazy Singleton

```
public final class Singleton {
    private static Singleton uniqueInstance;

    private Singleton() {
    }

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

Examples

- Windows Recycle Bin
- Java Runtime

Extension 1: Different between eager and lazy singleton

- Eager Singleton initialized itself when class is loaded, it is statically loaded. Lazy Singleton initialized itself when instance is first required.
  - From resources utilizing: Eager Singleton is worse than Lazy Singleton;
  - From runtime efficient: Eager Singleton is better than Lazy Singleton.
- Lazy Singleton have potential risk when in a multi-threads environment, because it is possible that several threads concurrently required the instance. It will cause that multiple instances are created.
- Eager Singleton is satisfied by Java language. On the contrary, it is not suitable in C++ language because the order of static initialization is unfixed.

Example: class MenuTree

- User
- “Eager” initialized ?
- “Lazy” initialized?
- Group
- static HashMap<String, MenuTree> cache

Extension 2: State of Singleton

- Stateful object
  - A stateful object contains and maintain a internal state that is retained across method calls and transactions.
  - Two stateful objects of one class are not same.
  - e.g. Constructor have arguments or class contains properties.
- Stateless object
  - A stateless object does not have any state between calls to its methods.
  - Two stateless objects of one class are same.
  - e.g. The class do not defines any properties.
- An singleton instance could be stateful, or stateless.
  - Stateful instance whose state can be shared among clinets could be a singleton.

Classifications of state

- Stateful
  - Mutable (Changeable State)
    - Changed by context (Extrinsic state)
      - Sharable
      - Unsharable
    - Changed by itself (Intrinsic state)
      - Sharable *(generally)*
  - Immutable(Unchanged State)
    - Sharable *(generally)*
- Stateless
  - Sharable

Extension 3: Singleton in distributed system

- Multiple JVM in a distributed system, OR
- Multiple class loader
  - If the singleton is stateless or immutable, it is no problem;
  - If the singleton is stateful and mutable , it will cause inconsistent.
- Stateless or immutable singleton is safety and recommended.
- For example, and singleton for counting.

Extension 4: Singleton and inheritance

- An singleton which are not final (or protected constructor) will allow the sub-class
  - Incomplete singleton if constructor of subclass is public;
  - It is useful in some special situation but not recommended.

Extension 5: Singleton and multiple threads

- In the multiple threads environment
  - Lazy Singleton is unsafe
  - Eager Singleton is safe

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

Extension 5: Singleton and multiple threads

```
public final class Singleton {
    private static Singleton uniqueInstance;

    private Singleton() {
    }

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

Extension 5: Singleton and multiple threads

- Optimized as:
  - Not really solo instance

```
public static Singleton getInstance() {
    if (uniqueInstance == null) {
        // multiple threads will stop here
        synchronized (Singleton.class) {
            uniqueInstance = new Singleton();
        }
    }
    return uniqueInstance;
}
```

Extension 5: Singleton and multiple threads – Double Checked

```
public static Singleton getInstance() {
    if (uniqueInstance == null) {
        // multiple threads will stop here
        synchronized (Singleton.class) {
            if (uniqueInstance == null) {
                uniqueInstance = new Singleton();
            }
        }
    }
    return uniqueInstance;
}
```

Potential Problem in Java

Extension 5: Singleton and multiple threads – Double Checked

- *uniqueInstance* = new Singleton();
  - locating memory;
  - Invoking the constructor Singleton();
  - Constructing the members of class;
  - …
  - *uniqueInstance* = reference of located memory;

  - locating memory;
  - *uniqueInstance* = reference of located memory;
  - Invoking the constructor Singleton();
  - Constructing the members of class;
  - …

Extension 6: Multiton Pattern

- Multiton (多例)
- An “singleton” class which create and manage multiple (numbered) instances, then provide global points to assess them .
- Be treated as a kind of enumerations.
- Currency, language, region, a group of config-file can adopt Multiton Pattern well.

Let’s go to next…