CS 351
Design of Large Programs
Singleton Pattern

Brooke Chenoweth

University of New Mexico

Spring 2024

The Notion of a Singleton

There are many objects we only need one of:

- Thread pools, caches, dialog boxes, logging objects, device drivers, etc.
- In many cases, instantiating more than one of such objects creates all kinds of problems
  - incorrect program behavior
  - resource overuse
  - inconsistent results

The Notion of a Singleton

- We could just use global (static) variables
- The Singleton pattern gives all of the upsides without the downsides
  e.g., object isn't forced to be created when the application starts
- Basically, the Singleton is used anytime you want a set of objects in the application to use the same global resource

Towards a Singleton

- In Java, how do you create a single object?

Towards a Singleton

- In Java, how do you create a single object?

  ```
  new MyClass()
  ```

- And if you call that a second time?

Towards a Singleton

- In Java, how do you create a single object?

  ```
  new MyClass()
  ```

- And if you call that a second time?
  You get a second, distinct object
- How could you prevent such instantiation?

Towards a Singleton

- In Java, how do you create a single object?

```
new MyClass()
```

- And if you call that a second time?
  You get a second, distinct object
- How could you prevent such instantiation?

```
public class MyClass {
  private MyClass() {}
}
```

- Who can use such a private constructor?

Towards a Singleton

- In Java, how do you create a single object?

  `new MyClass()`

- And if you call that a second time?
  You get a second, distinct object
- How could you prevent such instantiation?

```
public class MyClass {
  private MyClass() {}
}
```

- Who can use such a private constructor?
  Only code within `MyClass`

The Next Step

- How can you get access to code within `MyClass` if you can't instantiate it?

The Next Step

- How can you get access to code within `MyClass` if you can't instantiate it?
- What does this do?

```
public class MyClass {
  public static MyClass getInstance() {
    // code goes here
  }
}
```

The Next Step

- How can you get access to code within `MyClass` if you can't instantiate it?
- What does this do?

```
public class MyClass {
  public static MyClass getInstance() {
    // code goes here
  }
}
```

- How would you call that?

The Next Step

- How can you get access to code within `MyClass` if you can't instantiate it?
- What does this do?

```
public class MyClass {
  public static MyClass getInstance() {
    // code goes here
  }
}
```

- How would you call that?

```
MyClass.getInstance();
```

The Next Step

- How can you get access to code within `MyClass` if you can't instantiate it?
- What does this do?

```
public class MyClass {
  public static MyClass getInstance() {
    // code goes here
  }
}
```

- How would you call that?

```
MyClass.getInstance();
```

- How would you fill out the implementation to make sure that only a single instance of `MyClass` is ever created?

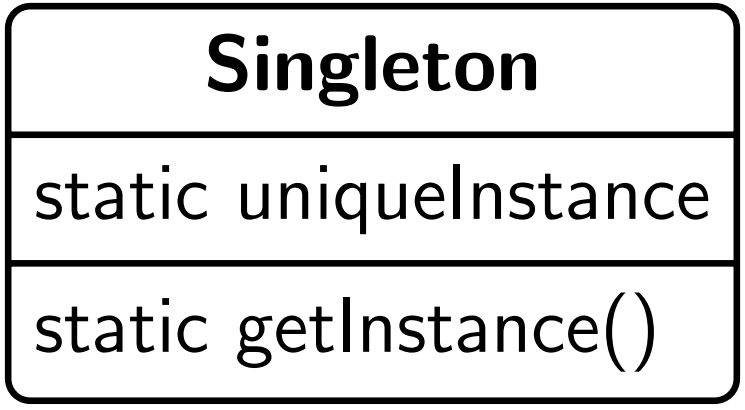
The Classic Singleton

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

  // additional instance variables

  private Singleton() {}

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

  // additional methods

}
```

The Singleton Pattern

The Singleton Pattern ensures a class has only one instance and provides a global point of access to that instance.

The Singleton Class Diagram



We have a problem...

- The Singleton pattern, as we have implemented it, is not *thread safe*
- When multiple threads invoke the `getInstance()` method, multiple instances of the object may be created!

Possible solution

- One simple solution is to use eager instantiation instead of lazy instantiation

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

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

- We will need to return to this when we study concurrent programming!

Some Questions

- What's the difference between a Singleton and a class in which all of the methods and variables are static?

Some Questions

- What's the difference between a Singleton and a class in which all of the methods and variables are static?
  - Using the Singleton pattern instead allows for complex initialization (especially if that initialization involves other classes and objects)
  - Without the Singleton pattern, you can still implement these things, but the result are common "order of initialization" bugs that are hard to pin down

Some Questions

- What's the difference between a Singleton and a class in which all of the methods and variables are static?
  - Using the Singleton pattern instead allows for complex initialization (especially if that initialization involves other classes and objects)
  - Without the Singleton pattern, you can still implement these things, but the result are common "order of initialization" bugs that are hard to pin down
- Why can't you subclass a Singleton?

Some Questions

- What's the difference between a Singleton and a class in which all of the methods and variables are static?
  - Using the Singleton pattern instead allows for complex initialization (especially if that initialization involves other classes and objects)
  - Without the Singleton pattern, you can still implement these things, but the result are common "order of initialization" bugs that are hard to pin down
- Why can't you subclass a Singleton?
  - You can't extend a class with a private constructor
  - All of the derived classes share the same static variable "instance"