# Singleton Pattern

Isaac Griffith

CS 2263
Department of Informatics and Computer Science
Idaho State University

## Outcomes

After today's lecture you will be able to:

- Understand the use of the Singleton Design Pattern
- Use and implement the Singleton Pattern

# Inspiration

“If we play genie and grant client wishes, we are apt to construct castles of code in the air.” – Larry Constantine

## Singleton Pattern

## Purpose

- Singleton is about the creation of a single instance of a class
- It is also the simplest in terms of class diagram
- But, possibly most complex in programming

## Where it is used

- Thread Pools
- Caches
- Dialog Boxes
- Preferences
- Registry Settings
- Logging
- Device Drivers

# The Chocolate Factory

- Choc-O-Holic, Inc’s industrial Strength Chocolate Boiler
- Computer Controlled, we’ll show the control code in a minute

# Convention

- Singleton is a convention for ensuring one and only one instance of a class
- So the question is how do we do this?

# Convention

- Singleton is a convention for ensuring one and only one instance of a class
- So the question is how do we do this?

```
public MyClass {

  private MyClass() {}

}
```

- How do we make use of this?

## Convention

We could then:

```
public MyClass {

  public static MyClass getInstance() {
  }
}
```

Idaho State University Computer Science

# Convention

```java
public MyClass {

  public static MyClass getInstance() {
    return new MyClass();
  }
}
```

- What is the problem with this?

ROAR

## Putting it all together

```java
public class Singleton {

  private static Singleton uniqueInstance;

  private Singleton() {}

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

# Back to the chocolate factory...

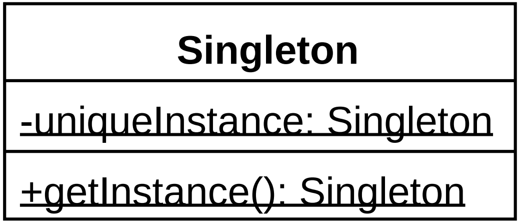
- How can we apply this to the chocolate boiler controller?

## Singleton Pattern Defined

- **Singleton Pattern** ensures a class has only one instance, and provides a global point of access to it.
- We allow this class to manage its single instance.
- We prevent any other class from creating an instance.
- We provide a global point of access.
- We also implement it to perform lazy initialization.

# Singleton Class Diagram

The Simplest Class Diagram



Idaho State University Computer Science

## Back at the Chocolate Factory

- Unfortunately they are having problems:
  - Somehow the fill() method was able to start filling the boiler even though it was already boiling.
  - What happened?

```
ChocolateBoiler boiler = ChocolateBoiler.getInstance();
boiler.fill();
boiler.boil();
boiler.drain();
```

ROAR

# Multiple Threads

### Thread One

```
public static ChocolateBoiler
  getInstance() {}


if (uniqueInstance == null) {}

  uniqueInstance =
     new ChocolateBoiler();
  return uniqueInstance;
```

### Thread 2

```

public static ChocolateBoiler
  getInstance() {}


if (uniqueInstance == null) {}



uniqueInstance =
   new ChocolateBoiler();
return uniqueInstance;
```

Idaho State University
Computer Science

# Dealing with Multithreading

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

  private Singleton() {}

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

ROAR

## Improving Multithreading

4 Choices:

1. Do nothing if the performance of `getInstance` isn't critical
2. Move to an eargerly created instance

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

  private Singleton() {}

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

# Improving Multithreading

4 Choices:

❶ Do nothing if the performance of `getInstance` isn’t critical

❷ Move to an eargerly created instance

❸ Use “double-checked locking” to reduce the use of synchronization

## Improving Multithreading

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

  private Singleton() {}
}
```

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

# Improving Multithreading

4 Choices:

1. Do nothing if the performance of `getInstance` isn't critical
2. Move to an eagerly created instance
3. Use "double-checked locking" to reduce the use of synchronization
4. Use a static inner class with lazy loading

# Improving Multithreading

```java
public class Singleton {
  private Singleton() {}

  public static Singleton getInstance() {
    return Helper.INSTANCE;
  }

  private static class Helper {
    public Singleton INSTANCE = new Singleton();
  }
}
```

## Are there any questions?

ROAR