- **What first execute when an object is created?**
  - First static block
  - Then execute only empty block when object calls
  - Then execute what inside the constructor

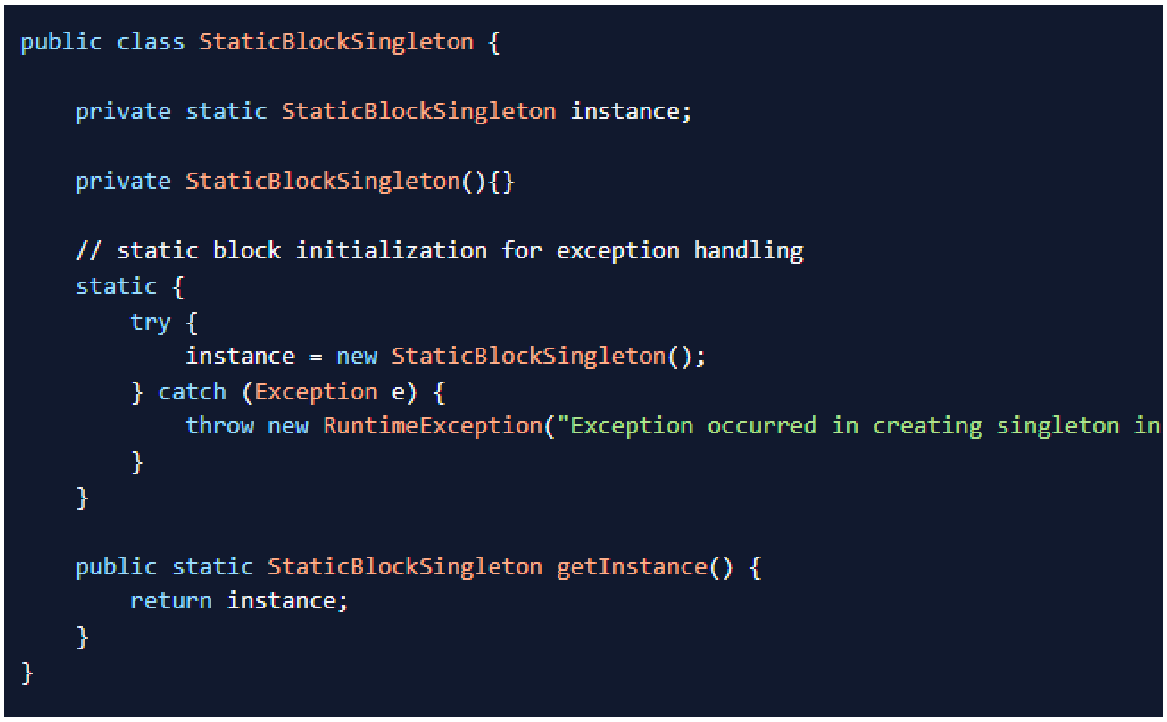
Eager initialization

- Singleton instance is created at the time of class loading with ensuring thread safety
- The instance is created when class loaded into the memory
- *This method used when our class is not using lot of resources*
- *This used for file creation, database creation*
- *Not provide exception handling*

```java
public class Singleton {
    // Eagerly creating the instance of the Singleton
    private static final Singleton INSTANCE = new Singleton();

    // Private constructor to prevent external instantiation
    private Singleton() { }

    // Public method to provide access to the instance
    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

Static block initialization

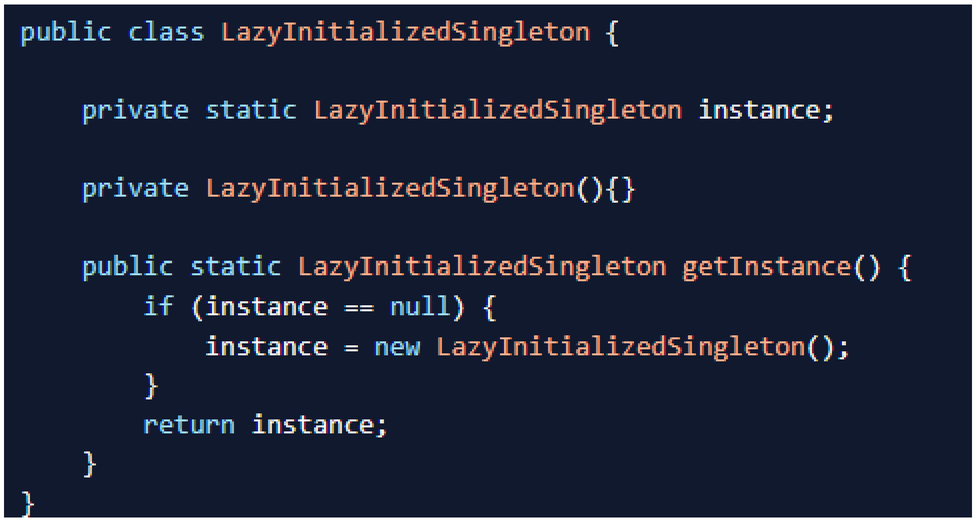
- Similar to eager initialization
- It give good advantage for static block initialization for **thread safe singleton**



```java
public class StaticBlockSingleton {

    private static StaticBlockSingleton instance;

    private StaticBlockSingleton(){}

    // static block initialization for exception handling
    static {
        try {
            instance = new StaticBlockSingleton();
        } catch (Exception e) {
            throw new RuntimeException("Exception occurred in creating singleton in
        }
    }

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

Lazy initialization

- Provide global access method
- Lazy initialization creates the Singleton instance only when it is first accessed.
- This approach is more efficient
- But require careful handling with multi thread

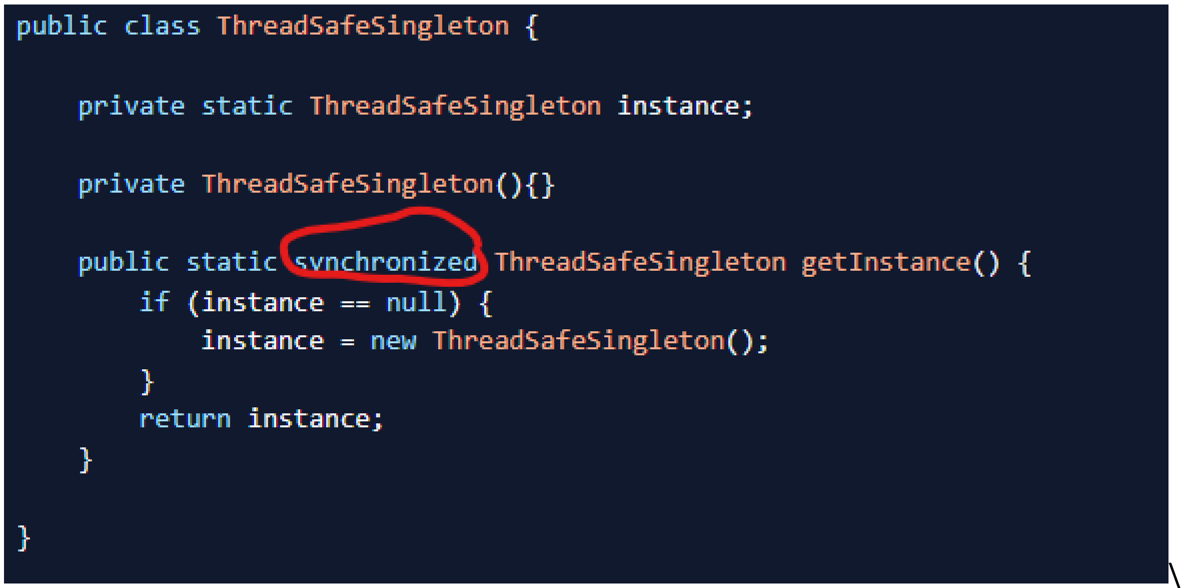


```java
public class LazyInitializedSingleton {

    private static LazyInitializedSingleton instance;

    private LazyInitializedSingleton(){}

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

- When it comes **to multithread system**, it cause issues if multiple threads are **inside if condition** at the same time.

Thread safe singleton

- A simple way to create a thread-safe singleton class
- Provide method synchronization
- This first way provide thread safety

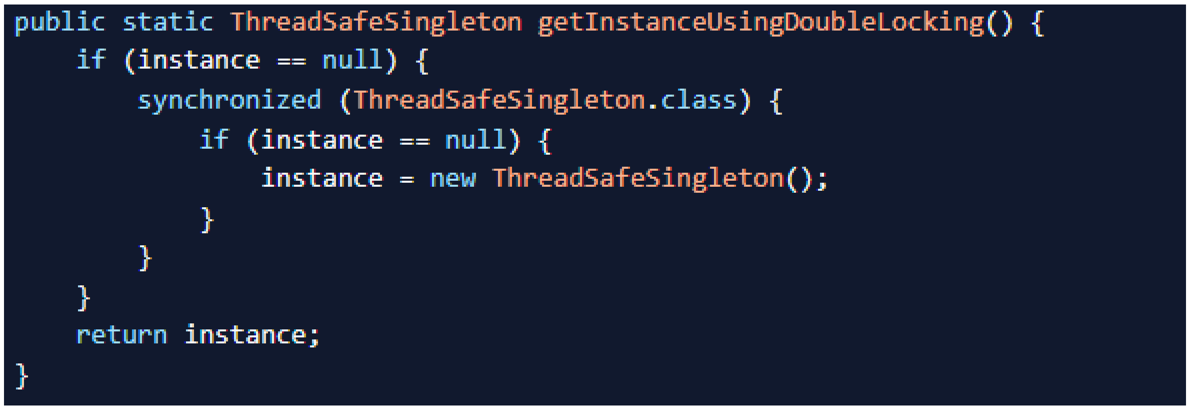


```
public class ThreadSafeSingleton {

    private static ThreadSafeSingleton instance;

    private ThreadSafeSingleton(){}

    public static synchronized ThreadSafeSingleton getInstance() {
        if (instance == null) {
            instance = new ThreadSafeSingleton();
        }
        return instance;
    }

}
```

\

Disadvantage and solution

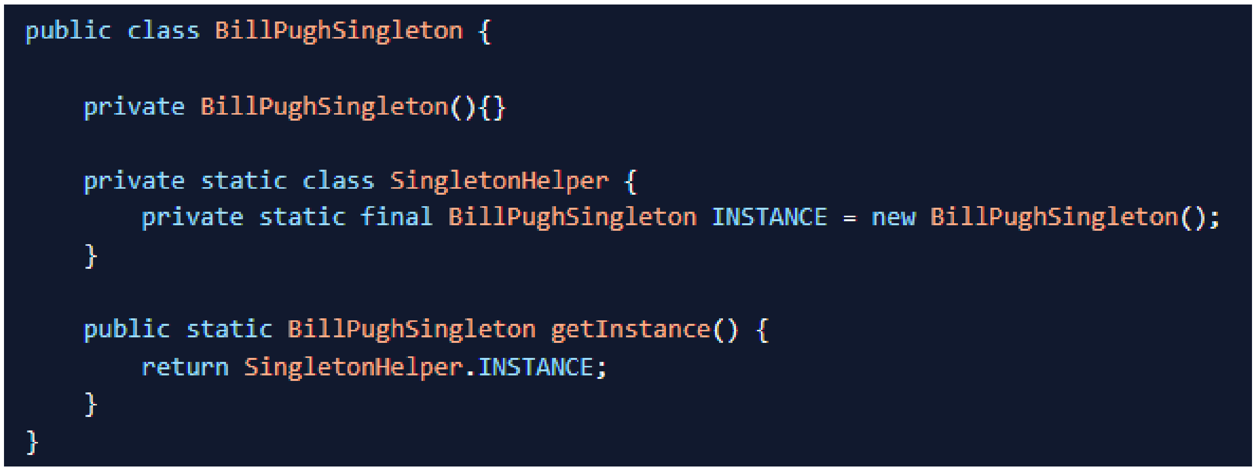
- Above reduce the performance because cost associated with synchronized method.
- To avoid extra overhead every time we used synchronized inside if condition.



```
public static ThreadSafeSingleton getInstanceUsingDoubleLocking() {
    if (instance == null) {
        synchronized (ThreadSafeSingleton.class) {
            if (instance == null) {
                instance = new ThreadSafeSingleton();
            }
        }
    }
    return instance;
}
```

Bill Pugh Singleton Implementation

- Java 5 has issues in memory model
- Above methods fail when more threads try to get the instance simultaneously
- Solution provide by Bill Pugh,



```
public class BillPughSingleton {

    private BillPughSingleton(){}

    private static class SingletonHelper {
        private static final BillPughSingleton INSTANCE = new BillPughSingleton();
    }

    public static BillPughSingleton getInstance() {
        return SingletonHelper.INSTANCE;
    }
}
```