

## Chapter 2: THE SINGLETON PATTERN

1) The Singleton Pattern .....	2-2
2) C++ Implementation .....	2-3
3) Java Implementation .....	2-4
4) Java API Examples .....	2-5

## The Singleton Pattern

- Use the Singleton pattern when:
  - ▶ There must be exactly one instance of a class.
  - ▶ The sole instance must be easily accessible.
- Benefits
  - ▶ Improvement over the use of a global variable
  - ▶ Controlled access to the sole instance
- Variations
  - ▶ Can be modified to permit a variable number of instances

## C++ Implementation

```
1. class Singleton
2. {
3.     private:
4.         static Singleton* _instance;
5.
6.     protected:
7.         Singleton();
8.
9.     public:
10.        static Singleton* getSingleton();
11. };
12.
13. Singleton* Singleton::_instance = 0;
14.
15. Singleton* Singleton::getSingleton()
16. {
17.     if (_instance == 0)
18.     {
19.         instance = new Singleton;
20.     }
21.     return _instance;
22. }
```

## Java Implementation

```
1. class Singleton
2. {
3.     private static Singleton instance = null;
4.
5.     protected Singleton() {...}
6.
7.     public static Singleton getInstance()
8.     {
9.         if (instance == null)
10.        {
11.            instance = new Singleton();
12.        }
13.        return instance;
14.    }
15. }
16.
17.
18. // A variation where the sole
19. // instance is "pre-constructed."
20.
21. class Singleton
22. {
23.     private static Singleton instance =
24.         new Singleton();
25.
26.     protected Singleton() {...}
27.
28.     public static Singleton getInstance()
29.     {
30.         return instance;
31.     }
32. }
```

## Java API Examples

- To use the instance methods of the `Runtime` class, you must obtain a reference to the sole `Runtime` object by using a static method of the class.

```
Runtime rt = Runtime.getRuntime();  
System.out.print ("Free memory = ");  
System.out.println (rt.freeMemory());
```

- ▶ Examine the source code for the `java.lang.Runtime` class to see the Singleton pattern.

- All parts of an application must use the same `SecurityManager` object. The unique instance is accessed through a static method of the `System` class.

```
SecurityManager manager =  
    System.getSecurityManager();
```

- ▶ Note: Once a `SecurityManager` has been set for a program, it cannot be changed.

**This Page Intentionally Left Blank**

Evaluation