1) a) O padrão Singleton, garante que uma certa classe tenha apenas uma instância fornecendo um ponto de acesso global para a mesma.

b) Usamos este padrão, quando queremos um ponto único de criação de instâncias e quando também queremos apenas uma instância de uma dada classe em razão da necessidade de evitar perdas de desempenho em suas conexões geradas por múltiplas instâncias.

c) Segue abaixo um exemplo de implementação:

**Exemplo 1**:

public class Singleton {

private static Singleton uniqueInstance;

private Singleton() {

}

public static synchronized Singleton getInstance() {

if (uniqueInstance == null)

uniqueInstance = new Singleton();

return uniqueInstance;

}

}

Acima temos a implementação do padrão Singleton. Nesse exemplo nota-se a presença do synchronized, isso se deve pois se tirássemos o synchronized e tentássemos criar duas instâncias da classe num determinado momento verificaríamos que isso é possível. Utilizando synchronized tem-se a certeza que o método nunca será acessado por duas threads ao mesmo tempo.

O construtor é privado evitando que essa classe seja instanciada fora dela. Assim, para podermos instanciar ou acessar uma instância da classe criou-se um atributo público e estático (da classe) que retorna através de um método estático uma única instância dessa classe. Como getInstance() é estático ele pode ser chamado de outra classe sem precisar instanciar a classe Singleton. Caso a classe já tenha sido instanciada o atributo não será nulo, assim retorna-se a única instância já criada.

**Exemplo 2**:

public class Singleton {

private static Singleton uniqueInstance = new Singleton();

private Singleton() {

}

public static Singleton getInstance() {

return uniqueInstance;

}

}



d) Declarando getInstance() como synchronized: uma desvantagem é diminuir o desempenho de quaisquer programa por um fator de 100, só deveríamos utilizá-lo quando o desempenho não seja comprometedor ao programa, o synchronized é uma chave, e a thread só poderá executar aquele trecho de código quando estiver em posse desta mesma chave, e em suma, pode haver uma sobrecarga de threads ou processos;

Eager initialization: com esta solução pode-se garantir que não irá ter sobrecarga, é a mais rápida das soluções, e é bem útil quando sabemos que nosso programa sempre criará e usará uma instância Singleton, o construtor sé chamado na primeira vez que a classe for carregada, quando tentamos executar quaisquer métodos da referida classe;

Trava duplamente verificada: é um método quem tem uma melhor elaboração, ele tem a opção e função de verificar se foi criada uma instância se não, apenas sincroniza,ele dá a garantia de a(s) thread(s) utilizar corretamente a variável *uniquelInstance* que cria a única instância. Esta solução consegue reduzir a sincronização do método getInstance(), ela diminui também o overhead que é imposto pela sincronização entre as threads (synchronized).