1. Objetivo

Esta prática tem por objetivo o desenvolvimento de um limitado serviço de *Home Banking* valendo-se do conceito de Tabelas *Hash*. As definições oriundas desta estrutura de dados, quando aplicadas ao problema sugerido, permitem utilizar conceitos de otimização de acesso a dados, bem como prover um interessante sistema de criptografia das informações. Logo, a proposta da prática implica em criar um serviço de *Home Banking* dotado de um sistema de segurança. Sugere-se que o sistema seja criado em linguagem Java.

2. Definições Importantes

As definições fornecidas nesta seção irão lhe ajudar a entender melhor de que forma se pode utilizar Tabelas *Hash* como mecanismo de criptografia.

2.1. *Message-Digest Algorithm* (MD5)

O MD5 é um algoritmo baseado no conceito de *hashing* e é amplamente utilizado para criptografia e checagem da integridade de dados. Diversas linguagens de programação possuem classes e métodos capazes de criptografar cadeias de caracteres com base neste algoritmo. Veja um exemplo de como criptografar uma `String` em Java:

```
public static String md5(String stringToConvert)
{
    String hashtext="";
    stringToConvert +=salt;
    salt é uma string aleatória qualquer
    MessageDigest m;
    try
    {
      m = MessageDigest.getInstance("MD5");
  m.reset();
      m.update(stringToConvert.getBytes());
      byte[] digest = m.digest();
      BigInteger bigInt = new BigInteger(1,digest);
      hashtext = bigInt.toString(16);
    }
    catch (NoSuchAlgorithmException ex)
    {
      Logger.getLogger(SecurityProvider.class.getName()).
      ...
      log(Level.SEVERE, null, ex);
    }
```

```
    return hashtext;
}
```

No exemplo dado acima foram utilizadas as classe `MessageDigest` e `BigInteger`, nativas da *API* da Linguagem Java. Para maiores detalhes veja a documentação das mesmas, encontradas em, respectivamente:

- http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html
- http://www.docjar.org/docs/api/java/math/BigInteger.html

Um fato importante ser destacado sobre o algoritmo MD5 quando aplicado em cadeias de caracteres é que não importa quantos caracteres a cadeia possua. A saída sempre terá o mesmo tamanho (32 caracteres). Também vale destacar que somente haverá uma saída para cada cadeia de entrada, ou seja, só será gerado o mesmo MD5 quando as cadeias de caracteres forem as mesmas.

A saída do algoritmo corresponde um número hexadecimal na forma de cadeia de caracteres, o qual também pode ser convertido para inteiro ou outro sistema numérico, dependendo da necessidade. Isto é, o algoritmo MD5 é em sua essência é um algoritmo de *hashing*. Veja dois exemplos distintos em que o código mostrado é aplicado.

Exemplo 1

- Entrada: 'Chuck'
- Saída: 78193bb6ffe829b49981b494ab243a82

Exemplo 2

- Entrada: 'A picape de Chuck Norris não ostenta chifres sobre o capô para colocar medo. Ao invés disso, ele prefere acelerar o motor furiosamente, extraindo um rugido ensurdecedor de pura virilidade. O barulho é tão poderoso que faz todos os veículos a sua volta implodirem instantaneamente, matando todos aqueles que estiverem sentados dentro, finalizando com roundhouse kicks para tirar de seu caminho todos os carros implodidos'
- Saída: 7cb1efff6159b6092337434262aeac3d

2.2. O Conceito *Salt*

Salt, ou sal, é um conceito de segurança que sugere o uso de uma cadeia de caracteres aleatória quando realizada a operação de *hashing*. Por exemplo, se você criptografar a palavra 'teste' com o algoritmo MD5, o resultado será '698dc19d489c4e4db73e28a713eab07b'. No entanto, esta chave MD5 pode ser facilmente quebrada: suponha que se construa um banco

de dados com palavras de um dicionário, logo bastaria gerar o MD5 de cada uma das palavras e ao MD5 que se quer quebrar.

Para tornar mais robusto o método de criptografia podemos adicionar uma cadeia de caracteres aleatória ao texto original e então criptografamos esta nova 'palavra'. Esta cadeia de caracteres que adicionamos ao nosso texto original é chamada de *salt*. Melhorando o exemplo anterior poderíamos utilizar a sequencia '838ns923n' como *salt*. Logo, aplicaríamos o algoritmo à 'teste838ns923n'. A saída seria equivalente a '9f63eb0e2f0f832d156e354a4df9e82'. Esta sequencia é muito mais difícil de ser quebrada, pois além de se adivihar qual é a palavra que foi criptografada há também a necessidade de se saber qual foi a senha, ou *salt*, utilizada.

3. Sua Tarefa

Você deverá implementar um sistema de *Home Banking* utilizando criptografia MD5. Seu sistema apenas irá executar a tarefa de *login* do cliente, retornando seu nome e saldo. Esta operação será descrita nas subseções a seguir, compondo-se de algumas subrotinas.

Passo 1

Crie uma classe chamada `Conta` com os seguintes atributos:

- `nomeCliente`
- `saldo`
- `agencia`
- `numero`
- `senha`
- `md5`

Esta classe deverá ter os seguintes construtores:

- `Conta(String agencia, String numero, String senha)`
- ```
  Conta(String agencia, String numero, String senha,
    String saldo, String nomeCliente)
  ```
- `Conta(String nomeCliente,String saldo)`

Adicione também à classe *getters* e *setters* e sobrescreva o método nativo *toString*, de forma que todos os dados da classe sejam impressos na tela. Para testar seu código, use o seguinte método:
  ```

```
public static void test()
{
    Conta c1 = new Conta("124", "333", "1234","10", "john doe");
    System.out.println(c1);
    Conta c2 = new Conta("John Doe", "10");
    System.out.println(c2);
    Conta c3 = new Conta("123", "321","666");
    System.out.println(c3);
}
```

Este teste deverá imprimir na tela o seguinte conteúdo:

```
AGENCIA: 124
CONTA: 333
SENHA: 1234
NOME CLIENTE: john doe
SALDO: 10
MD5: null

AGENCIA: null
CONTA: null
SENHA: null
NOME CLIENTE: John Doe
SALDO: 10
MD5: null

AGENCIA: 123
CONTA: 321
SENHA: 666
NOME CLIENTE: null
SALDO: null
MD5: null
```

Passo 2

Crie a classe `SecurityProvider`. Esta classe será responsável pela segurança da aplicação, criptografando as informações. Utilize o seguinte trecho de código para acelerar a implementação:

```
public class SecurityProvider
{
    public static String salt = "5a1t";
    public static String md5(String stringToConvert)
    {
```

```
         String hashtext="";
         stringToConvert +=salt;
         MessageDigest m;
         try
         {
             m = MessageDigest.getInstance("MD5");
             m.reset();
             m.update(stringToConvert.getBytes());
             byte[] digest = m.digest();
             BigInteger bigInt = new BigInteger(1,digest);
             hashtext = bigInt.toString(16);
         }
         catch (NoSuchAlgorithmException ex)
        {
            Logger.getLogger(SecurityProvider.class.getName()).
            ...
            log(Level.SEVERE, null, ex);
        }
        return hashtext;
    }
}
```

Para testar este código, use o seguinte método:

```
public static void test1()
{
    System.out.println(SecurityProvider.md5("teste"));
}
```

A saída deverá ser:

```
66e23b3b8413d2e219536d909eb320dc
```

Note que a `String` *salt* já está pré-definida na classe. Mantenha-a, para fins de facilitar os testes em seu código. Agora adicione à classe o seguinte método:

```
public static String md5ToServer(Conta conta)
{
}
```

Este método deverá receber um objeto do tipo conta e retornar uma chave md5 relativa a este objeto. O método irá aplicar o método md5 da classe `SecurityProvider` para fazer a criptografia, sendo a entrada para o método uma `String` resultante da concatenação dos atributos agencia, numero e senha, da classe `Conta`, mais a `String` *salt*, isto é:

```
    String cat = conta.agencia+conta.numero+
    ...
    conta.senha+SecurityProvider.salt;
```

Teste seu método:

```
public static void test2()
{
    Conta c = new Conta("1234", "2222","1245");
    System.out.println(SecurityProvider.md5ToServer(c));
}
```

A saída deverá ser:

```
245e43ed3a0d09deac1c8beb52c5e049
```

Por fim, complete os construtores da classe `Conta`, fazendo com que o campo md5 de cada conta seja inicializado por meio do método criado. Teste novamente os construtores:

```
public static void test()
{
    Conta c1 = new Conta("124", "333", "1234","10", "john doe");
    System.out.println(c1);
    Conta c2 = new Conta("John Doe", "10");
    System.out.println(c2);
    Conta c3 = new Conta("123", "321","666");
    System.out.println(c3);
}
```

Veja o retorno:

```
AGENCIA: 124
CONTA: 333
SENHA: 1234
NOME CLIENTE: john doe
SALDO: 10
MD5: d8868a6f9e369703ca7c235292360842

AGENCIA: null
CONTA: null
SENHA: null
NOME CLIENTE: John Doe
SALDO: 10
MD5: null
```

```
AGENCIA: 123
CONTA: 321
SENHA: 666
NOME CLIENTE: null
SALDO: null
MD5: 270263d637e5a0f430ce7f605301851a
```

Passo 3

Agora crie a classe `ServerDatabase`, a qual irá armazenar as contas dos clientes por meio de uma tabela *hash*. Use a seguinte estrutura:

```
public class ServerDatabase extends Database
{
    public static final ArrayList<ArrayList<Conta>> contas;
    public static final int N = 100;
    static
    {
        contas = new ArrayList<ArrayList<Conta>>();
        for(int i=0;i<N;i++)
        {
            contas.add(new ArrayList<Conta>());
        }
    }
}
```

Adicione à classe o método estático chamado *hashCode* com a seguinte estrutura:

```
public static int hashCode(String md5)
{
    BigInteger bi = new BigInteger(md5, 16);
    BigInteger m = new BigInteger(Integer.toString(N), 10);
    int pos;
    pos = bi.mod(m).intValue();
    return pos;
}
```

Este método deverá receber uma `String` md5 e dizer em qual posição da tabela *hash* contas um novo objeto deverá ser inserido. Note que para isto o método utiliza o módulo do valor inteiro da chave fornecida. Agora, usando este método, crie o método `insereConta`:

```
public static void insereConta(Conta conta)
{
}
```

O método irá receber uma conta, obter seu md5 e então calcular em qual posição da tabela *hash* a conta deverá ser inserida. Para resolver o problema de colisões, considere que cada linha da tabela é um vetor dinâmico, do tipo `ArrayList`, como indicado anteriormente. Você poderá utilizar o código md5 do objeto Conta para verificar se dois objetos são iguais. Adicione também à classe o método `getConta`:

```
public static Conta getConta(String md5)
{
}
```

Este método irá receber uma `String` md5 de uma conta e irá busca-la na tabela *hash* contas, usando o método `hashCode`. O método irá retornar nulo ou a conta em questão. Teste seu código com o seguinte método:

```
public static void test3()
{
    Conta c = new Conta("1234", "2222","1245");
    ServerDatabase.insereConta(c);
    String chave = SecurityProvider.md5ToServer(c);
    System.out.println(chave);
    Conta conta = ServerDatabase.getConta(chave);
    System.out.println(conta);
}
```

Este teste deverá imprimir na tela

```
245e43ed3a0d09deac1c8beb52c5e049
AGENCIA: 1234
CONTA: 2222
SENHA: 1245
NOME CLIENTE: null
SALDO: null
MD5: 245e43ed3a0d09deac1c8beb52c5e049
```

Passo 4

Adicione à classe `SecurityProvider` um método para criptografar em md5 as informações da conta do cliente, as quais serão enviadas de volta após o login, isto é, seu nome e seu saldo. O método deverá ter a estrutura mostrada abaixo. Note que o retorno do método é um *array* de chaves md5. Desta forma, as informações de retorno deverão ser codificadas caractere a caractere. Cada caractere terá um código md5 correspondente, o qual deverá ser gerado pelo método md5 da classe em questão. Veja o exemplo da `String` a ser criptografada:

```
String toCrypt = conta.getNomeCliente()+" "+conta.getSaldo();
```

Assinatura do método:

```
public static String[] md5ToClient(Conta conta)
{
}
```

Para testar seu método, use o código:

```
public static void test4()
{
    Conta c = new Conta("124", "333", "1234","10", "john doe");
    ServerDatabase.insereConta(c);
    String chave = SecurityProvider.md5ToServer(c);
    Conta conta = ServerDatabase.getConta(chave);
    String chars[];
    chars = SecurityProvider.md5ToClient(conta);
    for(int i=0;i<chars.length;i++)
        {
            System.out.println(chars[i]);
        }
}
```

A saída observada será

```
9b4ff28f48bdb2ac725221a3b9e4e81e
cb159a41f5b94562450b57555d8bf18a
16b9f1dbe5c97ed57c48cc0d55e0a920
44bc04868555d05106534ebb4e25a349
67cf64b8ee2d78139a1879edd920656c
4e29004bd2a0080076877a6a929d26fb
cb159a41f5b94562450b57555d8bf18a
9d4ccb0d7fb7dd3ae6a3ce429640db17
67cf64b8ee2d78139a1879edd920656c
340631132fa5d19c96a14df8daa0bea
82e1863e6ffcada8ef3e7de421e422cd
```

Passo 5

Crie a classe `Letra` com os seguintes atributos:

- `String caractere`
- `String md5Code`

Crie também o seguinte construtor:

`Letra(String caractere)`

Este método irá receber um caractere e inicializar o campo md5Code por meio da classe `SecurityProvider`. Crie também *getters and setters*.

Passo 6

Crie a classe `Database` a partir da estrutura que segue

```
public class Database
{
    public static final ArrayList<Letra> caracteres;
    static
    {
        caracteres = new ArrayList<Letra>();
    }
}
```

Sua tarefa agora é inicializar o `ArrayList` caracteres com todos os caracteres alfanuméricos, incluindo o caractere de espaço. Feito isso, adicione a esta classe o método `public static Letra getLetra(String md5)`, que irá pesquisar na lista um caractere a partir da chave md5 dada.

Para testar sua classe, use o código a seguir:

```
public static void test6()
{
    Letra l = new Letra("a");
    String md5 = l.getMd5Code();
    System.out.println(md5);
    Letra ll = Database.getLetra(md5);
    System.out.println(ll.getCaractere());
}
```

A saída mostrada deverá ser

```
8fb2e27c72a8b8302d0d9b99fcc58b57
a
```

Passo 7 - final

Crie agora na classe `Database` o método `getConta`, que receberá o vetor de chaves md5 gerado pelo método `md5ToClient` (classe `SecurityProvider`) e retorna um objeto do tipo conta. A conta retornada apenas terá as informações relativas ao nome do cliente e ao saldo, isto é, os dados retornados do `ServerDatabase` quando do login. Você não poderá acessar o `ServerDatabase` neste método, pois deverá ser feita a descriptografia da mensagem recebida.

Teste seu código

```
public static void test5()
{
Conta c = new Conta("124", "333", "1234","10", "john doe");
ServerDatabase.insereConta(c);
String chave = SecurityProvider.md5ToServer(c);
Conta conta = ServerDatabase.getConta(chave);
String chars[];
chars = SecurityProvider.md5ToClient(conta);
System.out.println(Database.getConta(chars));
}
```

Saída gerada

```
AGENCIA: null
CONTA: null
SENHA: null
NOME CLIENTE: john doe
SALDO: 10
MD5: null
```

*Este trabalho prático é de autoria de Éverton Santi (UFRN, Brasil)