# Módulo 7

Segurança


# Lição 9

Message Digest

***Versão 1.0 - Jan/2008***

**Autor**
Aldwin Lee
Cheryl Lorica

**Equipe**
Rommel Feria
John Paul Petines

## Necessidades para os Exercícios

### Sistemas Operacionais Suportados

**NetBeans IDE 5.5** para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

**NetBeans Enterprise Pack**, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

### Configuração Mínima de Hardware

**Nota:** IDE NetBeans com resolução de tela em 1024x768 pixel

| Sistema Operacional | Processador | Memória | HD Livre |
|---|---|---|---|
| Microsoft Windows | 500 MHz Intel Pentium III workstation ou equivalente | 512 MB | 850 MB |
| Linux | 500 MHz Intel Pentium III workstation ou equivalente | 512 MB | 450 MB |
| Solaris OS (SPARC) | UltraSPARC II 450 MHz | 512 MB | 450 MB |
| Solaris OS (x86/x64 Platform Edition) | AMD Opteron 100 Série 1.8 GHz | 512 MB | 450 MB |
| Mac OS X | PowerPC G4 | 512 MB | 450 MB |

### Configuração Recomendada de Hardware

| Sistema Operacional | Processador | Memória | HD Livre |
|---|---|---|---|
| Microsoft Windows | 1.4 GHz Intel Pentium III workstation ou equivalente | 1 GB | 1 GB |
| Linux | 1.4 GHz Intel Pentium III workstation ou equivalente | 1 GB | 850 MB |
| Solaris OS (SPARC) | UltraSPARC IIIi 1 GHz | 1 GB | 850 MB |
| Solaris OS (x86/x64 Platform Edition) | AMD Opteron 100 Series 1.8 GHz | 1 GB | 850 MB |
| Mac OS X | PowerPC G5 | 1 GB | 850 MB |

### Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

### *Colaboradores que auxiliaram no processo de tradução e revisão*

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Angelo de Oliveira
Bruno da Silva Bonfim
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Guilherme da Silveira Elias
Leandro Souza de Jesus
Lucas Vinícius Bibiano Thomé
Luiz Fernandes de Oliveira Junior
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Paulo Oliveira Sampaio Reis
Ronie Dotzlaw

## *Auxiliadores especiais*

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach e Vinícius G. Ribeiro (Especialista em Segurança)
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

## *Coordenação do DFJUG*

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

## Agradecimento Especial

**John Paul Petines –** Criador da Iniciativa JEDI™
**Rommel Feria** – Criador da Iniciativa JEDI™

# 1. Objetivos

Uma *message digest*, também conhecida como '*message fingerprint*' ou '*hash* seguro' é uma *String* de tamanho fixo que é o resultado da passagem dos dados por um algoritmo chamado *hash* – também conhecido por espalhamento, sentido único ou de mão única. A *message digest* é produzida através de uma função *hash*. A função é chamada de sentido único, uma vez que é impossível que a mensagem original seja extraída da *message digest*.

A função *hash* ideal é aquela que nunca produz uma mesma *message digest* para as possíveis *Strings* de entrada. Porém, essa 'perfeição teórica' de uma mensagem de entrada (sem tamanho fixo) é impossível com o tamanho curto e fixo para a *message digest* de saída. Para isso seria necessário que a *message digest* tivesse o mesmo tamanho da mensagem de entrada. Isto significa que podemos ter uma *String* qualquer de entrada e produzir uma *message digest* curta de tamanho fixo, mas esta mensagem não necessariamente corresponde SOMENTE à mensagem que o gerou. Se uma determinada *message digest* tenha for gerada a partir de duas mensagens de entrada diferente, teremos o que é chamado de colisão. (para maiores informações acesse http://www.cits.rub.de/MD5Collisions/)

Uma *message digest* é uma assinatura digital compacta de um fluxo de dados qualquer. Uma boa implementação de uma função *hash* deve produzir uma grande quantidade de mensagens diferentes, mesmo para uma pequena mudança na string de entrada, como por exemplo a alteração de um caractere da *String* de entrada de minúsculo para maiúsculo.

Ao final desta lição, o estudante será capaz de:

- Conhecer os principais algoritmos de *Message Digest*
- Identificar os principais usos da utilização de *Message Digest*
- Empregar a classe *MessageDigest* em um aplicativo

# 2. Algoritmos de Message Digest

Os algoritmos *hash* mais comuns são o MD5 e o SHA. São suportados pelos provedores de segurança padrões.

**MD5 (*Message Digest* 5)**

O algoritmo de *message digest* mais utilizado atualmente é o algoritmo MD5 de 128 *bits*, desenvolvido por *Ron Rivest* do *MIT Laboratory for Computer Science and RSA Data Security*. Os algoritmos hash MD5 de 128 bits (16 bytes) são normalmente representados por uma seqüência de 32 dígitos hexadecimais. O MD2 e o MD4 foram algoritmos anteriores da família RSA. Ambos são atualmente considerados obsoletos. Nenhum deles ainda foi quebrado, mas ambos mostram-se potencialmente fracos graças ao tamanho da chave. Há estudos de implementação para uma versão que utilize chave com um tamanho maior – provavelmente, será denominado de MD6.

**SHA** (**Secure Hash Algorithm**)

Os algoritmos SHA foram projetados pela Agência Nacional de Segurança (*National Security Agency* - NSA) e foram publicados como padrão do governo americano. O SHA-1 foi considerado como o sucessor do MD5. Para mais informações sobre o SHA, por favor visite o endereço http://www.itl.nist.gov/fipspubs/fip180-1.htm (*Federal Information Processing Standards – Publication Secure Hash Standard*).

| ***Nome*** | ***Algoritmo*** |
|---|---|
| SHA-1 | Produz uma mensagem com 20 bytes (40 dígitos Hex); aplicável a documentos com menos de $2^{64}$ bits. |
| SHA-256 | Produz uma mensagem com 32 bytes (64 dígitos Hex); aplicável a documentos com menos de $2^{64}$ bits. |
| SHA-384 | Produz uma mensagem de 48 bytes (96 dígitos Hex); aplicável a documentos com menos de $2^{128}$ bits. |
| SHA-512 | Produz uma mensagem de 64 bytes (128 dígitos Hex); aplicável a documentos com menos de $2^{128}$ bits. |

Vejamos por exemplo a geração da *message digest* para a seguinte frase:

```
Java Security **cks!

MD5: cf93f2442e8e183d865d7fa9c251aa41
SHA-1: 331cc7479ab3571c96bf1c6ad30738ca0507d386
```

Ou seja, quando passamos a mensagem para o método *hash* produzimos uma *message digest* de tamanho fixo, dependendo do algoritmo utilizado.


*Figura 1: Comparação MD5 e SHA-1*

Substituindo os asteriscos, encontramos a seguinte frase:

```
Java Security rocks!

MD5: 6da66136e4d1d0af49d633edb7b53c12
SHA-1: f56f6f92804dfb8935b2185fc520044bba4c8b2a
```

Como podemos ver, duas simples alterações de caracteres, de '**' para 'ro', produziram grandes

diferenças nas *message digest*.

Isto serve como evidência para indicar que a mudança em alguns caracteres altera a *message digest* completamente. Na mensagem seguinte o 'r' na palavra 'rock' foi alterada para maiúscula. Isto produziu uma grande diferença nas *message digest* do MD5 e do SHA-1.

```
Java Security Rocks!

MD5: 79f131c274a82294177fbad8b977a707
SHA-1: f6e26ec898b69e563dd2bf2245ed12fb64780e30
```

Para comparar, lado a lado, as *message digest* geradas, veja a tabela abaixo.

| ***Mensagem*** | ***MD5*** | ***SHA-1*** |
|---|---|---|
| Java Security **cks! | `cf93f2442e8e183d865d7fa9c251aa41` | `331cc7479ab3571c96bf1c6ad30738ca0507d386` |
| Java Security rocks! | `6da66136e4d1d0af49d633edb7b53c12` | `f56f6f92804dfb8935b2185fc520044bba4c8b2a` |
| Java Security Rocks! | `79f131c274a82294177fbad8b977a707` | `f6e26ec898b69e563dd2bf2245ed12fb64780e30` |

# 3. Aplicações comuns

*Hashing* como sabemos é utilizado em muitas aplicações. Isto inclui aumento de desempenho, autenticação e verificação de dados.

**Aumento de desempenho**

A tabela *hash* utiliza um método *hash* que indexa a chave dentro da posição correta na tabela *hash*. Veja a **lição 10 do Módulo 3 – Estruturas de Dados** para mais informações sobre Tabelas *Hash* e técnicas de *hashing*.

**Autenticação**

Senhas podem ser guardadas através do uso de métodos *hash*. No linux, senhas de usuários são armazenadas em um arquivo chamado /etc/password, se o *shadow* não estiver habilitado. Para guardar senhas reais do usuário, elas são criptografadas através das chaves *hash*.

```
root:x:0:1:Super-User:/:/sbin/sh
ozzie:6k/7KCFRPNVXg:508:10:& Ozzie:/usr2/ozzie:/bin/csh
```

A senha do Ozzie acima é '6k/7KCFRPNVXg'. Na prática existe a adição de um *bit* aleatório, chamado *salt*, adicionado à senha antes que ela seja criptografada.

**Assinatura digital de documentos**

'Assinar' digitalmente um documento é similar a assinar ou autografar um papel. Esta assinatura digital comprova que você concorda com a mensagem ou dado. O processo de assinatura de um documento envolve uma assinatura pública e particular e a chave *hash* da mensagem. A assinatura digital é gerada a partir da criptografia da chave *hash* da mensagem com a chave da assinatura privada.

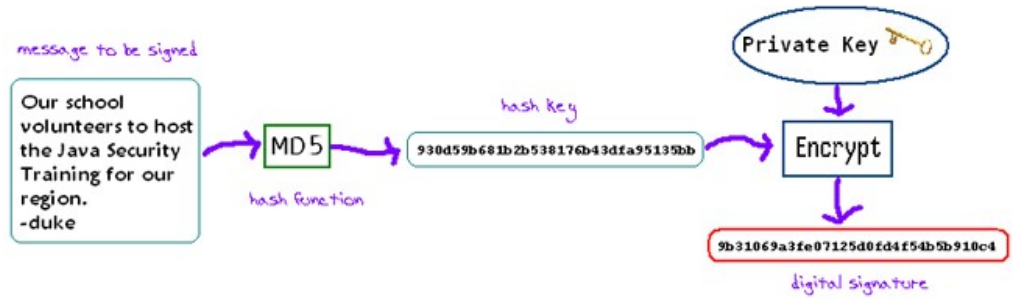


*Figura 2: Assinatura Digital de Documentos - Criptografia*

Para provar que uma assinatura pertence a um documento, a assinatura digital é utilizada para decriptar a chave pública do signatário. Isto resulta na função *hash* do documento. Se a chave *hash* decriptada combinar com a chave *hash* da mensagem original, então o proprietário da chave é o signatário do documento.

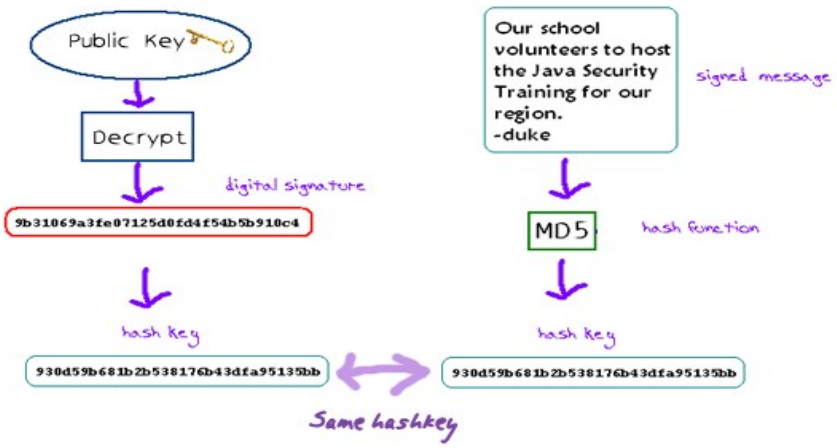


*Figura 3: Assinatura Digital de Documentos - Decriptar*

Cobriremos em detalhes este assunto no próximo capítulo, Assinaturas Digitais e Certificados.

**Verificação de Dados**

*Message digests* são freqüentemente utilizadas para verificar quais dados não foram alterados desde que a assinatura foi publicada. Alterações de dados podem ocorrer durante a transmissão dos dados, por exemplo, dano acidental (transmissão realizada durante uma tempestade) ou por acesso indevido (ataque por um programa que possa danificar esta transmissão)

Resumos MD5 ou *checksums* (verificação de somatório) MD5 têm sido amplamente utilizados para fornecer segurança para que arquivos particulares não sejam alterados durante a transferência. Isto pode ocorrer devido a programas maliciosos ou simplesmente por corrompimento durante o seu envio ou recebimento. O publicador do arquivo ou classe disponibiliza a soma MD5 em seu site. Depois de ter baixado o arquivo deve-se recalcular o *checksum* MD5 para verificar a integridade do arquivo. Compara-se então o *checksum* MD5 publicado com o *checksum* MD5 calculado. Se os *checksum* MD5 calculados não coincidirem então a transmissão terá falhado ou o arquivo foi alterado.

Um exemplo poderia ser o *download* do OpenOffice. Uma vez baixado o software OpenOffice, podemos verificar a lista de *checksum* MD5 para as diferentes plataformas no endereço http://download.openoffice.org/1.1.5/md5sums.html. Para o *download* do OpenOffice Solaris x86, a checagem da soma (*checksum*) é a exibida abaixo.

**91cbb9d5bda04c9c93d1475b6b806928** Ooo_1.1.5_Solarisx86_install.tar.gz

Calcula-se então o *checksum* de verificação do arquivo baixado. Sistemas baseados no Linux têm uma ferramenta para cálculo do *checksum* MD5 de um arquivo. Sistemas baseados no Windows podem baixar ferramentas para o cálculo do *checksum* MD5 de terceiros.

```
bash-2.05b$ md5sum OOo_1.1.5_Solarisx86_install.tar.gz
91cbb9d5bda04c9c93d1475b6b806928
```

O MD5 publicado e o *checksum* MD5 calculado geram um mesmo resultado MD5.

# 4. A classe *MessageDigest*

Em Java, *message digests* são armazenadas em *arrays* de bytes que são calculados utilizando a classe java.security.MessageDigest.

```
java.lang.Object
     ∟ java.security.MessageDigestSpi
          ∟ java.security.MessageDigest
```

| Construtor | |
|---|---|
| protected | **MessageDigest**(String algorithm)<br>Cria uma *message digest* com o algoritmo especificado. |

| Métodos | |
|---|---|
| Object | **clone**()<br>Retorna um clone se a implementação for *clonable*. |
| byte[] | **digest**()<br>Completa o cálculo *hash* através da execução de operações finais como *padding*. |
| byte[] | **digest**(byte[] input)<br>Executa uma atualização final sobre o resumo utilizando um vetor de *bytes* que completa o cálculo. |
| int | **digest**(byte[] buf, int offset, int len)<br>Completa o cálculo *hash* através da execução de operações finais como *padding*. |
| String | **getAlgorithm**()<br>Retorna uma *String* que identifica o algoritmo, independente dos detalhes da implementação. |
| int | **getDigestLength**()<br>Retorna o comprimento da mensagem em bytes ou 0 se esta operação não for suportada pelo provedor de segurança e a implementação não for *clonable*. |
| static MessageDigest | **getInstance**(String algorithm)<br>Gera uma instância *MessageDigest* que implementa o algoritmo de resumo (digest) especificado. |
| static MessageDigest | **getInstance**(String algorithm, Provider provider)<br>Gera uma instância *MessageDigest* que implementa um algoritmo específico, como fornecido pelo provedor de segurança, se possuir um. |
| static MessageDigest | **getInstance**(String algorithm, String provider)<br>Gera uma instância *MessageDigest* implementando um algoritmo específico, como fornecido pelo provedor de segurança, se possuir um. |
| Provider | **getProvider**()<br>Retorna o provedor de segurança desta instância da *message digest*. |
| static boolean | **isEqual**(byte[] digesta, byte[] digestb)<br>Compara duas *message digest* em relação ao seu conteúdo. |
| void | **reset**()<br>Reinicia uma mensagem para uso futuro. |
| String | **toString**()<br>Retorna uma *String* de representação da mensagem de uma instância *message digest*. |
| void | **update**(byte input)<br>Atualiza uma *message digest* utilizando o *byte* especificado. |
| void | **update**(byte[] input)<br>Atualiza *message digest* utilizando um *array* de bytes especificado. |
| void | **update**(byte[] input, int offset, int len)<br>Atualiza um *message digest* utilizando um array de bytes especificado, a partir da posição *offset* especificada. O número de bytes utilizados, iniciando do *offset*, é especificado |

| | |
|---|---|
| | pelo argumento len. |

A MessageDigest é uma classe abstrata que fornece um meio de implementar classes de algoritmos de *message digest*. Estes algoritmos são fornecidos por vários provedores de pacotes de segurança. A versão da Sun de seu JRE vem com um provedor de segurança padrão: o provedor "SUN". Outros ambientes de execução Java não necessariamente fornecem o provedor "SUN". O pacote do provedor "SUN" inclui as seguintes implementações dos algoritmos para a classe *MessageDigest*:

| Algoritmo | Tamanho da saída *hash* |
|---|---|
| • MD2 | 128 bits |
| • MD5 | 128 bits |
| • SHA-1 | 160 bits |
| • SHA-256 | 256 bits |
| • SHA-384 | 384 bits |
| • SHA-512 | 512 bits |

Quando qualquer um dos algoritmos listados acima for requisitado, o provedor "SUN" é utilizado se nenhum provedor de serviço for especificado ou se o provedor "SUN" for explicitamente indicado. Isto significa que o provedor "SUN" tem maior prioridade entre os provedores de segurança. Esta preferência é configurável. (Para uma revisão dos pacotes de provedores de segurança, por favor veja a classe *Provider*).

Métodos importantes da classe *MessageDigest*:

***getInstance*()** - O método *getInstance*() é um método estático polimórfico por overload que retorna uma instância da *message digest* de um determinado algoritmo e provedor de segurança (caso existam estas especifidades). Há três assinaturas para o *getInstance*:

- *getInstance*(*String algorithm*) – Esta assinatura aceita uma *String* que especifica o algoritmo *hash* a ser utilizado. Caso o provedor não seja especificado a implementação padrão será utilizada
- *getInstance*(*String algorithm, Provider provider*) – Esta assinatura aceita uma *String* que especifica o algoritmo *hash* e também aceita uma instância Provider que especifica qual a implementação a ser utilizada
- *getInstance*(*String algorithm, String provider*) – Esta assinatura aceita uma *String* que especifica o algoritmo *hash* e também aceita uma *String* que especifica qual provedor, que, por sua vez, indica que implementação deve ser utilizada

***update*()** - Este método adiciona dados à mensagem de entrada para o processamento

- *update*(*byte input*) – Atualiza o *message digest* utilizando o *byte* especificado
- *update*(*byte[] input*) – Atualiza o *message digest* utilizando um array de *bytes* especificado
- *update*(*byte[] input, int offset, int len*) – Atualiza o *message digest* utilizando um *array* de *bytes* especificado a partir de uma posição *offset* indicada. O número de *bytes* utilizados, iniciando do *offset*, é especificado pelo argumento *len*.

***reset*()** – Este método redefine a *message digest* ao seu estado inicial para uso futuro. Utilize este método para executar outro cálculo sem precisar instanciar o *MessageDigest*.

***digest*()** – Retorna os dados da mensagem a partir dos dados de entrada da instância *message digest* corrente. A *message digest* é reiniciada após cada chamada. Teoricamente, esta redefinição da *message digest* é de responsabilidade do provedor de segurança, e não é possível garantir que a *message digest* sempre será redefinida.

- *byte[] digest*() - Completa o cálculo do *hash* pela execução de uma operação final como o padding e então retorna a *message digest* em um *array* de *bytes*.
- *byte[] digest*(*byte[] input*) – Executa a atualização final sobre a *message digest* utilizando um *array* de *bytes* específico, e então conclui o cálculo da *message digest*. Ou seja,

primeiro chama o método *update(input)* e o método *digest*().

- *int digest*(*byte[] buf, int offset, int len*) – Conclui o cálculo do *hash* pela execução de operações como o *padding*.

# 5. Como implementar uma message digest

Os passos a seguir ilustram como criar uma *message digest*.

1. Obter uma instância da classe *MessageDigest* para o algoritmo apropriado

O JDK fornece implementações de SHA-1 e MD5. Se o pacote do fornecedor padrão fornece uma implementação do algoritmo com uma *message digest* específico, uma instância da *MessageDigest* contendo essa implementação é retornada. Se o algoritmo não estiver disponível no pacote padrão, é procurado em outros pacotes. Se for especificado um algoritmo que nenhum fornecedor implemente, *NoSuchAlgorithmException* é lançada. O algoritmo SHA-1 resulta em uma *message digest* de 20 *bytes*, enquanto que o MD5 possui um de tamanho de 16 bytes.

```
MessageDigest md5 = MessageDigest.getInstance("MD5");
```

ou

```
MessageDigest sha1 = MessageDigest.getInstance("SHA-1");
```

2. Obter a mensagem como um array de *bytes*

A mensagem pode vir de um *InputStream* carregado, de um *InputStream* da rede, de um arquivo, ou de uma simples *String*. Para o nosso propósito, vamos pegar a mensagem de um arquivo.

```
BufferedInputStream message = new BufferedInputStream(
  new FileInputStream(filename));
// Pegar a mensagem de um input stream através de uma linha de comando
BufferedInputStream message = new BufferedInputStream(
  new InputStreamReader(System.in));
// Criar um ByteArrayOutputStream
ByteArrayOutputStream baos = new ByteArrayOutputStream();
// Ler o arquivo por caracteres individuais
int ch;
while ((ch = bis.read()) != -1) {
  baos.write(ch);
}
// Obter um array de byte do ByteArrayOutputStream
byte[] buffer = baos.toByteArray();
```

3. Adicionar a mensagem ao carregador da *message digest* ao chamar o método *update*

Lembre-se que ao chamar este método adiciona-se o método carregado do *input* ao final da *message digest*. Então, se o *update* já houver sido chamado na *message digest*, e desejamos limpar o carregador, chamamos o método *reset*() para limpar o carregador da *message digest*.

```
/* chame md5.reset() para reiniciar o carregador se você já houver utilizado
essa instância previamente */
md5.update(message);
```

4. Gerar o sumário

Alguns algoritmos requerem que um *padding* seja adicionado a *message digest*. O método *digest*() cria automaticamente o *padding* requerido pelo algoritmo para que não seja necessário se preocupar com isso. O método *digest*() retorna um *array* de *bytes* que contém o *checksum* ou a *message digest*.

```
byte[] digest = md5.digest();
```

5. Converter para *String*

Para converter o código *hash* em um objeto do tipo *String*, lembre-se de utilizar *Integer.toHexString*(), este método não insere zeros à esquerda, então deve-se adicioná-los por conta própria.

```
StringBuffer md5Hex = new StringBuffer();
for (int i=0; i<digest.length; i++) {
```

```
    // Inserir 0xFF para adicionar um zero condutor para cada elemento do array
    md5Hex.append(Integer.toHexString(0xFF & digest[i]));
  }
```

6. Finalizar

Imprimir a *message digest*, salve-o em um arquivo e insira-o no banco de dados.

```
System.out.println(md5Hex.toString());
```

A *message digest* pode estar qualquer um dos 2 status, '*initialized*'(iniciado) ou '*in progress*'(em progresso). Uma vez obtida uma instância do *MessageDigest*, esse objeto inicia como '*initialized*'. Os dados são processados pela *message digest*, utilizando qualquer um dos métodos *update*(). Uma vez chamado este método, o status da *message digest* muda para '*in progress*'. O método *reset*() pode se chamado a qualquer momento para reiniciar a *message digest*, revertendo seu status para '*initialized*'.

Uma vez que todos os dados foram processados, um dos métodos *digest*() deve ser chamado para completar o processo da junção. O método *digest*() pode ser chamado apenas uma vez para um dado número de alterações. Depois que o método *digest*() foi chamado, o objeto *MessageDigest* é reiniciado para seu status *initialized*.

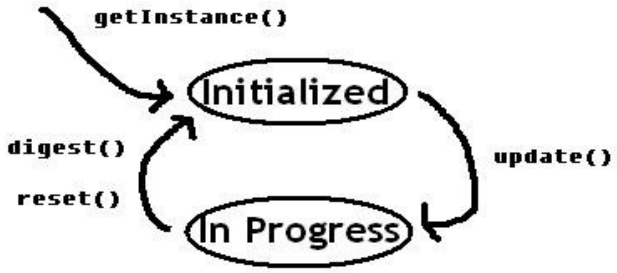


*Figura 4: Modelo de criação de uma Message Digest*

Exemplo:

O programa a seguir processa a união de assinaturas do MD5 de um dado arquivo. Para utilizar o programa, utilize o nome do arquivo como um argumento. Esse arquivo será lido e o sumário da mensagem MD5 correspondente será exibido.

```
bash-2.05b$ java MD5 /home/jedi/input.txt
73692f161ea854bf739147dfbbb5f8
```

```
/* MD5.java */

package jedi.security.messagedigest.MD5;

import java.io.BufferedInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class MD5 {
  public static void main(String[] args) {
    if (args.length < 1){
      System.out.println("Usage: MD5 <file>");
      return;
    }
    MD5 md = new MD5();
    System.out.println("MD5: " + md.getHashSignature(args[0]));
  }
```

```
  public String getHashSignature(String filename){
    StringBuffer md5Hex = new StringBuffer();
    try {
      MessageDigest md5 = MessageDigest.getInstance("MD5");
      BufferedInputStream message = new BufferedInputStream(
        new FileInputStream(filename));
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      int ch;
      while ((ch = message.read()) != -1) {
        baos.write(ch);
      }
      byte[] buffer = baos.toByteArray();
      message.close();
      baos.close();
      md5.update(buffer);
      byte[] digest = md5.digest();
      for (int i=0;i<digest.length;i++) {
        md5Hex.append(Integer.toHexString(0xFF & digest[i]));
      }
    } catch (NoSuchAlgorithmException ex) {
      System.out.println(
        "Sorry, there's no such algorithm on the default provider");
    } catch (FileNotFoundException ex) {
      System.out.println("Sorry, file " + filename + " was not found");
    } catch (IOException ioe){
      System.out.println("Sorry, I/O Exception occured.");
    } finally {
      return md5Hex.toString();
    }
  }
}
```

# 6. Exercícios

## 6.1. Aplicações do Mundo Real

1. Vamos supor uma tabela no banco de dados de usuários que podem acessar seu sistema. O sistema foi implementado de maneira que as senhas dos usuários são armazenadas como assinaturas de *hash* SHA-1. Recebemos uma solicitação de mudança para implementar uma funcionalidade de modificação da senha. Nos requisitos de negócio constam:
   a) Solicitar a senha atual e a nova senha
   b) Se a senha atual corresponde à senha do banco de dados, proceder à mudança da senha atual para a nova senha
2. Recentemente foi feito o download de um arquivo da internet. Não existe a certeza se o arquivo foi corrompido ou modificado durante a transmissão. O *hash* MD5 do arquivo é fornecido no site. Desconhecendo a existência alguns geradores MD5 de código livre, devemos criar nossa própria aplicação Java para calcular o *checksum* MD5.

# Parceiros que tornaram JEDI™ possível


***Instituto CTS***
Patrocinador do DFJUG.

***Sun Microsystems***
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

***Java Research and Development Center da Universidade das Filipinas***
Criador da Iniciativa JEDI™.

***DFJUG***
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

***Banco do Brasil***
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

***Politec***
Suporte e apoio financeiro e logístico a todo o processo.

***Borland***
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

***Instituto Gaudium/CNBB***
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.