Módulo 7

Segurança

Lição 10

Listas de Controle de Acesso

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 Lista de Controle de Acesso (ACL) é uma estrutura de dados que armazena os recursos acessados. Uma ACL pode ser analisada de como uma estrutura de dados com múltiplos acessos. Cada acesso ACL é um conjunto de permissões associadas a um solicitante ou grupo de solicitantes. Além disso, cada ACL contém um sinal (uma flag), que indica se as permissões devem ou não ser concedidas ou negadas.

ACL promove uma maneira fácil de criar usuários e grupos, cada um combinado a um conjunto específico de permissões que controlarão como cada usuário ou grupo irá atuar na aplicação Java.

As Listas de Controle de Acesso respondem à pergunta: "Como autorizar um usuário a acessar dados de outros usuários ou outros dados, que não são permitidos?".

Ao final desta lição, o estudante será capaz de:

- Conhecer as características da ACL
- Obter maiores informações sobre Java e as Listas de Controle de Acesso

2. Características das ACL

Uma lista de controle de acesso independe do tipo de autenticação utilizado para verificar a validade do controlador. A ACL independe do tipo de criptografia utilizado para transmitir os dados através da rede. A ACL é consultada após a fase de autenticação. A própria ACL é independente das pesquisas que armazena.

Depois que o solicitante é autenticado para ser um usuário do sistema, pode acessar seus recursos. Para cada recurso, o solicitante pode ou não ter assegurado seu acesso, dependendo das permissões que são concedidas na ACL que guarda estes recursos.

A ACL pode ser consultada para encontrar a lista das permissões que um solicitante possui ou verificar se há ou não permissões especiais para este solicitante.

2.1. Modelo de Controle de Acesso

O modelo a seguir é extraído de *Lampson et al.: Authentication in Distributed Systems: Theory and Practice, ACM ToCS*, 1992

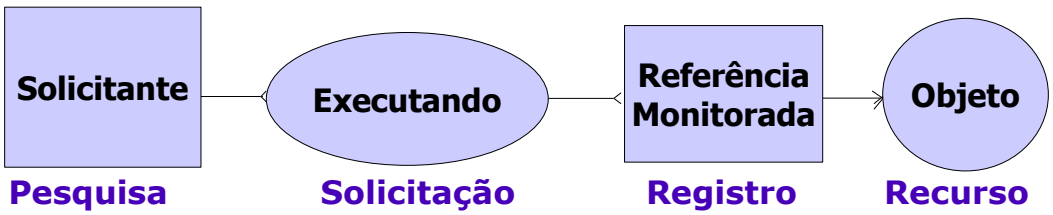

Figura 1: Modelo de ACL

Solicitante

Fonte dos pedidos. Um solicitante pode representar uma entidade, um usuário, um grupo ou um processo. Pode ser qualquer coisa que possua identificação e requer acesso a um recurso.

Solicitação

Realiza as operações com os objetos. Podendo, inclusive, ler, escrever ou executar.

Referência monitorada

Guarda cada objeto e avalia cada pedido para os objetos, decidindo se deve ou não conceder acesso.

Objeto

Recursos tais como: arquivos, dispositivos ou processos.

A referência monitorada baseia sua decisão na apresentação do pedido principal, na operação de solicitação, bem como nas regras de acesso que controlam quais solicitantes poderão realizar essa operação sobre o objeto.

Para executar seu trabalho, o monitor necessita de um meio confiável para reconhecer tanto a origem do pedido, quanto a regra de acesso. A obtenção do código fonte do pedido é conhecido por "autenticação", interpretar a regra de acesso é chamado "autorização".

Assim, a autenticação responde à pergunta ''Quem disse isso?'', e a autorização responde à pergunta ''Quem é confiável para acessar isso?''. Normalmente, a regra de acesso é anexada ao objeto; tal regra é chamada lista de controle de acesso ou ACL. Para cada operação a ACL especifica um conjunto de indivíduos autorizados e o monitor concede um pedido se o indivíduo for tão confiável quanto o solicitante que está autorizado a fazer a operação no pedido.

2.2. Cálculo de Permissões Concedidas

Cada acesso na ACL é especificado como positivo ou negativo. Um acesso positivo significa que seus recursos devem estar disponíveis quando um usuário ou processo tentar acessá-los. Uma

entrada negativa significa que o recurso deve ser negado.

Quando calculamos as permissões que um solicitante terá, as seguintes regras devem ser usadas:

1. Cada indivíduo ou grupo pode ter uma entrada positiva e outra negativa. Isto quer dizer que entradas ACL duplicadas não são permitidas;
2. Se não houver entrada de um determinado indivíduo ou de um grupo, significa que o indivíduo ou o grupo tem permissão nula;
3. O conjunto de grupos positivos que permite definir um indivíduo é a união de todas as permissões positivas de cada grupo a que o indivíduo pertence;
4. O conjunto de grupos negativos que permite definir um indivíduo é a união de todos as permissões negativas de cada grupo que o indivíduo pertence;
5. Se existe uma entrada positiva, que concede a um indivíduo ou grupo uma permissão especial, e uma entrada negativa, que nega alguma destas permissões, então todas as permissões são removidas, tanto positivas, quanto negativas;
6. Permissões individuais (permissões concedidas ou negadas para um solicitante específico) sempre tem prioridade sobre as permissões de um grupo. Ou seja, indivíduos com permissões negativas (*specific denial of permissions*) tem prioridade sobre os grupos com permissões positivas. E indivíduos com permissões positivas tem prioridade sobre grupos com permissões negativas;
7. Presumindo que todo o grupo (g1),que o indivíduo pertença, tem permissão positiva e que todo grupo (g2), que o indivíduo pertença, tem permissão negativa. Assumimos, também, que o indivíduo de permissão positiva é definido por (p1) e o indivíduo de permissão negativa é definido por (p2). Então a resultante das permissões, que os indivíduos possuem é (p1 + (g1 - p2)) - (p2 + (g2 - p1)).

2.3. Exemplo de Cálculo das Permissões

Presumindo que o solicitante P pertença ao grupos G1 e G2. A tabela abaixo mostra 5 colunas com alguns exemplos de permissões dadas a G1, G2 e P. O resultado das permissões concedidas a P é verificado na última coluna.

	Permissões do Grupo G1	**Permissões do Grupo G2**	**Permissões da União (G1, G2)**	**Permissões Individuais**	**Resultado das Permissões**
Positivo	A	B	A+B	C	A+B+C
Negativo	nula	nula	nula	nula	
Positivo	A	B	B	C	B+C
Negativo	-C	-A	-C	nula	
Positivo	A	B	A+B	C	B+C
Negativo	nula	nula	nula	-A	
Positivo	A	C	A	B	B
Negativo	-C	-B	-B	-A	

Para o primeiro exemplo de permissões, temos:

	Permissões do Grupo G1	**Permissões do Grupo G2**	**Permissões da União (G1, G2)**	**Permissões Individuais**	**Resultado das Permissões**
Positivo	A	B	A+B	C	A+B+C
Negativo	nula	nula	nula	nula	

O grupo positivo tem permissões: (A+B)

O grupo negativo tem permissões: (nula)
Portanto, a união dos 2 grupos de permissões é: (A e B)

Além disso, a autorização individual é: (C)

Portanto, o resultado das permissões é: (A+B+C)

Para o segundo exemplo de permissões, temos:

	Permissões do Grupo G1	Permissões do Grupo G2	Permissões da União (G1, G2)	Permissões Individuais	Resultado das Permissões
Positivo	A	B	B	C	B+C
Negativo	-C	-A	-C	nula	

O grupo positivo tem permissões: (A+B)
O grupo negativo tem permissões: (A+C)
Portanto, a união dos 2 grupos de permissões é: (B e -C)

Além disso, a autorização individual é: (C)

A regra 6 indica que permissões individuais terão sempre prioridade sobre permissões de um grupo, as permissões resultantes são (B + C), em vez de apenas (B).

No terceiro exemplo de permissões, temos:

	Permissões do Grupo G1	Permissões do Grupo G2	Permissões da União (G1, G2)	Permissões Individuais	Resultado das Permissões
Positivo	A	B	A+B	C	B+C
Negativo	nula	nula	nula	-A	

O grupo positivo tem permissões: (A+B)
O grupo negativo tem permissões: (nula)
Portanto, a união dos 2 grupos de permissões é: (A e B)

Além disso, a autorização individual é: (-A+C)

As permissões resultantes são (B + C), uma vez que a permissão do indivíduo negativo (A), se sobrepõe à permissão (A) do grupo.

Para finalizar, no quarto exemplo permissões, temos:

	Permissões do Grupo G1	Permissões do Grupo G2	Permissões da União (G1, G2)	Permissões Individuais	Resultado das Permissões
Positivo	A	C	A	B	B
Negativo	-C	-B	-B	-A	

O grupo positivo tem permissões: (A+C)
O grupo negativo tem permissões: (B+C)
Portanto, a união dos 2 grupos de permissões é: (A e -B)

Além disso, a autorização individual é: (-A + B)

O resultado das permissões é (B+C), quando a permissão do individuo negativo for A, este irá sobrescrever a permissão do grupo (positivo), quando esta for também A.

3. Java e as Listas de Controle de Acesso

O pacote *java.security.acl* fornece a interface para essa estrutura de dados que guarda o acesso aos recursos. O *sun.security.acl* fornece uma estrutura padrão para implementação das interfaces especificadas no pacote *java.security.acl.*

3.1. Estrutura da ACL

Em Java, uma ACL é um objeto que implementa a interface *java.security.acl.Acl*. Cada *Acl* é uma lista de objetos *AclEntry*. Cada *AclEntry* está associada a um objeto *Indivíduo* ou *Grupo* de uma lista de objetos *Permissões*. Cada *AclEntry* também pode ser associada a uma entrada positiva ou negativa. Uma entrada positiva concede a lista de permissões na entrada do indivíduo, ou grupo, e uma entrada negativa nega a lista de permissões para o indivíduo ou grupo.

3.2. Pacote java.security.acl

Este pacote contém classes e interfaces usadas durante a execução das Listas de Controle de Acesso para assegurar ou negar permissões a indivíduos ou grupos de indivíduos.

3.2.1. Interfaces

Interface Acl

Interface representando uma Lista de Controle de Acesso (ACL). Uma ACL é uma estrutura de dados usada para guardar acessos aos recursos.

Uma ACL pode ser considerada como uma estrutura de dados com múltiplas entradas ACL. Cada entrada ACL, de interface *AclEntry*, contém um conjunto de permissões associadas a um determinado solicitante (Um solicitante representa uma entidade, como um usuário individual ou de grupo). Além disso, cada entrada ACL é especificada como positiva ou negativa. Caso seja positiva, as permissões serão concedidos aos respectivos solicitantes. Caso seja negativa, as permissões serão negadas.

As solicitações ACL em cada ACL devem observar as seguintes regras:

- Cada solicitante pode ter, no máximo, uma entrada ACL positiva e uma negativa. Isto é, múltiplas entradas ACL, positivas ou negativas, não são permitidas para nenhum indivíduo. Cada entrada especifica o conjunto de permissões que serão concedidas (se positiva) ou negadas (se negativas)
- Se não houver entrada de nenhum solicitante, então considera-se que o solicitante tem um conjunto vazio de permissões
- Se houver uma entrada positiva, que concede uma determinada permissão a um indivíduo e uma entrada negativa, que nega esta entrada na mesma permissão, o resultado é como se a permissão nunca fosse concedida ou recusada
- Permissões individuais sempre terão prioridade sobre permissões de grupo ao qual pertence o indivíduo. Isto é, permissões individuais negativas substituirão os grupos com permissões positivas. E permissões individuais positivas substituirão as permissões dos grupos negativos.

Interface AclEntry

Esta é a interface usada para representar uma entrada na Lista de Controle de Acesso (ACL).

Uma ACL pode ser comparada a uma estrutura de dados com múltiplos objetos de entrada ACL. Cada objeto de entrada ACL contém um conjunto de permissões associadas com um *Principal* em particular. (Um *Principal* representa uma entidade como um usuário individual ou um grupo). Adicionalmente, cada entrada ACL é especificada como sendo tanto positiva como negativa. Se positiva, as permissões devem ser concedidas para o principal associado. Se negativa, as permissões serão negadas. Cada *Principal* deve ter pelo menos uma entrada positiva e uma negativa; assim sendo, múltiplas entradas positivas ou negativas não são permitidas para

qualquer *Principal*. As entradas ACL por padrão são positivas. Um entrada se torna negativa apenas se o método *setNegativePermissions* for chamado.

Interface Group

Essa interface é usada para representar um grupo de *Principals*. Um *Principal* representa uma entidade como um usuário individual ou um empresa.

Observe que *Group* estende *Principal*. Então, tanto um *Principal* ou um *Group* podem ser passados como um argumento para métodos que contém um parâmetro *Principal*. Por exemplo, podemos adicionar um *Principal* ou um *Group* para um objeto *Group* chamando o método *addMember* desse objeto, passando o *Principal* ou *Group*.

Interface Owner

Interface para gerenciamento de *Owners* de Listas de Controle de Acesso ou configurações *ACL*. (Observe que a interface *Acl* no pacote *java.security.acl* estende a interface *Owner*). O *Owner* inicial de um Principal deve ser especificado como um argumento para o construtor da classe que implementa essa interface.

Interface Permission

Essa interface representa uma permissão, como às usadas para conceder um tipo particular de acesso para um recurso.

3.2.2. Exceções

AclNotFoundException

Essa é uma exceção que é lançada sempre que uma referência é feita a um ACL que não existe.

LastOwnerException

Essa é uma exceção que é lançada sempre que é feita uma tentativa de apagar o último dono de uma Lista de Controle de Acesso.

NotOwnerException

Essa é uma exceção que é lançada sempre que a modificação de um objeto (como uma Lista de Controle de Acesso) é permitida ser feita apenas pelo dono do objeto, mas o Principal tentando modificar não é o dono.

3.3. Implementando uma ACL

O foco do pacote *java.security.acl* é a interface *Acl*, que representa uma lista de controle de acesso. Uma ACL tem um grupo de donos associados com ela, representado pelos objetos da classe *Principal*. *Principal* é um termo usado no meio de segurança para se referir a um usuário agindo como uma parte em uma transação de segurança. Como ambas classes *Identity* e *Signer* são subclasses de *Principal*, podemos usar instâncias de qualquer uma onde um *Principal* estiver sendo chamado. Apenas *Owners* da ACL devem ser capazes de modificá-la. Implementações da interface *Acl* devem reforçar isso checando as chaves e certificados dos *Owners* iniciais, para assegurar que o agente criando ou modificando a *ACL* tem acesso aos elementos certificados da identidade de um dos *Owners* da *ACL*.

Para definirmos um conjunto de tipos de permissões, um objeto da classe *Permission* tem que ser criado. Isso pode ser feito instanciando-se um *PermissionImpl* e associando-o a um objeto *Permission*.

```
Permission create = new PermissionImpl("CREATE");
Permission read = new PermissionImpl("READ");
Permission update = new PermissionImpl("UPDATE");
Permission destroy = new PermissionImpl("DELETE");
```

Para criarmos *Principals*, instanciamos um *PrincipalImpl* e o associamos a um objeto *Principal*. O pacote *sun.security.acl* provê uma implementação de *Permission* chamada *PermissionImpl*, uma subclasse da interface *Principal*, que utiliza *Strings* para identificar tipos de permissões (ex.,

"READ", "WRITE").

Em uma aplicação real, poderíamos usar um objeto *Identity* ou *Signer* para representar um *Principal* na ACL. Isso nos permitiria verificar uma assinatura digital de um cliente remoto antes de permitir o acesso do cliente remoto a recursos protegidos pela ACL.

```
Principal hunny = new PrincipalImpl("Hunny");
Principal ozzie = new PrincipalImpl("Ozzie");
```

Cada entrada na lista de controle de acesso é representada como um objeto *AclEntry*, o qual associa identidades específicas com permissões existentes para um recurso sendo controlado.

```
AclEntry aclEntry1 = new AclEntryImpl(hunny);
AclEntry aclEntry2 = new AclEntryImpl(ozzie);
```

Uma entrada é adicionada ao Acl usando o método *addEntry*(), o qual pega o *Principal* da entidade e seus *AclEntry* como argumentos.

Cada *AclEntry* define um conjunto de permissões dadas para o *Principal* sobre o recurso sendo protegido. Tipos específicos de permissões são representados usando a interface *Permission*, a qual não implementa qualquer comportamento, mas age como *placeholder* para subclasses que distinguem permissões de maneiras específicas da aplicação (nomes de permissões, tipos binários, entre outros).

Para associar permissões para cada *Principal*, o método *addPermission*() em um *AclEntry* é usado. Esse método recebe objetos de permissão.

Para *Hunny*, o qual é um elemento de *aclEntry1*, apenas permissões *read* e *update* são dadas.

```
aclEntry1.addPermission(read);
aclEntry1.addPermission(update);
```

Para *Ozzie*, o qual é elemento de *aclEntry2*, todas as permissões *create*, *read*, *update* e *delete* são concedidas.

```
aclEntry2.addPermission(create);
aclEntry2.addPermission(read);
aclEntry2.addPermission(update);
aclEntry2.addPermission(delete);
```

Uma vez que o *AclEntrys* for criado, podem ser adicionados à *ACL* através do método *addEntry*(). O método recebe dois argumentos: um *Principal* que corresponde ao dono da *ACL* que realiza a entrada, e o *AclEntry*. Por exemplo:

```
Acl myAcl = new AclImpl(hunny, "SampleACL1");
myAcl.addEntry(hunny, aclEntry1);
myAcl.addEntry(hunny, aclEntry2);
```

O exemplo a seguir cria um grupo com 2 usuários *Principals*, *user1* e *user2*. Inicialmente, as permissões dos grupos são configuradas para *read* e *write*. A permissão individual para *user1* é então configurada para não escrever (permissão negativa para *write*). Cada passo será descrito a seguir.

Criar Principals

```
Principal p1 = new PrincipalImpl("user1");
Principal p2 = new PrincipalImpl("user2");
Principal owner = new PrincipalImpl("owner");
```

Criar Permissions

```
Permission read = new PermissionImpl("READ");
Permission write = new PermissionImpl("WRITE");
```

Criar um grupo que contém p1 (user1) e p2 (user2)

```
Group g = new GroupImpl("group1");
g.addMember(p1);
```

```
g.addMember(p2);
```

Criar um novo ACL com o nome exampleAcl

```
Acl acl = new AclImpl(owner, "exampleAcl");
```

Configurar permissão do grupo para *read* e *write*

```
AclEntry entry1 = new AclEntryImpl(g);
entry1.addPermission(read);
entry1.addPermission(write);
acl.addEntry(owner, entry1);
```

Remover a permissão *write* apenas para o *user1* (permissão individual negativa)

```
AclEntry entry2 = new AclEntryImpl(p1);
entry2.addPermission(write);
entry2.setNegativePermissions();
acl.addEntry(owner, entry2);
```

Testes são criados para verificar os resultados do exemplo ACL

Como p1 tem permissão de *read* e *write*, e uma permissão individual de negativo *write*, uma enumeration de permissões acessíveis para p1 deverá listar apenas permissões *read*.

```
Enumeration e1 = acl.getPermissions(p1);
```

Como p2 tem permissões de grupo *read* e *write*, e permissão individual de null, uma enumeração das permissões acessíveis para p1 deverá listar permissões *read* e *write*.

```
Enumeration e2 = acl.getPermissions(p2);
```

Para checar a permissão de um *Principal* em particular, o *checkPermission*(*Principal prin, Permission perm*) pode ser usado. O método irá retornar verdadeiro se *Principal* puder executar a permissão ou falso, caso contrário.

Executando *checkPermission* de *write* no *Principal p1* retornará falso pois a permissão individual de *write* para *p1* é configurada para negativo (*write* não é permitido).

```
boolean b1 = acl.checkPermission(p1, write);
```

Executando *checkPermission* de *read* no *Principal p1* irá retornar verdadeiro pois a permissão do grupo permiti acesso de leitura para *p1*. Adicionalmente, executando *read* e *write checkPermissions* no *p2* irá retornar verdadeiro pois *read* e *write* é permitido para *p2*. Nos casos abaixo, nenhuma permissão individual sobrescreve a permissão do grupo, então todos os comandos abaixo irão retornar verdadeiro.

```
boolean b2 = acl.checkPermission(p1, read);
boolean b3 = acl.checkPermission(p2, read);
boolean b4 = acl.checkPermission(p2, write);
```

Um exemplo de classe ACL é definida abaixo

```
import java.security.acl.*;
import sun.security.acl.*;
import java.util.Enumeration;
import java.security.Principal;

public class ExampleACL {

    public static void main(String[] args) throws NotOwnerException {
        Principal p1 = new PrincipalImpl("user1");
        Principal p2 = new PrincipalImpl("user2");
        Principal owner = new PrincipalImpl("owner");
        Permission read = new PermissionImpl("READ");
        Permission write = new PermissionImpl("WRITE");
        Group g = new GroupImpl("group1");
```

```
            g.addMember(p1);
            g.addMember(p2);
            Acl acl = new AclImpl(owner, "exampleAcl");
            AclEntry entry1 = new AclEntryImpl(g);
            entry1.addPermission(read);
            entry1.addPermission(write);
            acl.addEntry(owner, entry1);
            AclEntry entry2 = new AclEntryImpl(p1);
            entry2.addPermission(write);
            entry2.setNegativePermissions();
            acl.addEntry(owner, entry2);
            Enumeration e1 = acl.getPermissions(p1);
            Enumeration e2 = acl.getPermissions(p2);
            boolean b1 = acl.checkPermission(p1, write);
            boolean b2 = acl.checkPermission(p1, read);
            boolean b3 = acl.checkPermission(p2, read);
            boolean b4 = acl.checkPermission(p2, write);
            System.out.println(b1 + " " + b2 + " " + " " + b3 + " " + b4);
        }
    }
```

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.