

Programação Orientada a Objetos - POOS3

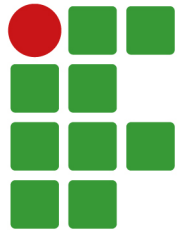
1

Tecnologia em Análise e Desenvolvimento de Sistemas

Aula 9

Collections, singleton, equals, compareTo,
hashCode, Map

2º semestre de 2018



INSTITUTO FEDERAL

São Paulo

Câmpus Araraquara

Situação problema

- Implementar um sistema de cadastro de usuários.
 - Cada usuário possui username (único), nome completo e senha (criptografada).
 - O sistema deve manter os dados em um List de forma que fiquem classificados pelo username. Não pode-se permitir o cadastro de dois usuários com o mesmo username.



Usuário é simples, basta definir uma classe para um objeto com os atributos do objeto.

Agora no conjunto como irei verificar se um username é o mesmo?

Como classificar o conjunto (coleção) de dados? Existe algo pronto?

Nenhuma novidade da implementação da classe Usuário.



Fixação
5!!!

```
public class Usuario {
    private String username;
    private String nome;
    private String senha;

    public Usuario(String username, String nome, String senha) {
        this.username = username;
        this.nome = nome;
        setSenha(senha);
    }

    public boolean autenticar(String username, String senha){
        boolean validado = false;
        if(this.username.equals(username)){
            if(this.senha.equals(getMD5(senha))){
                validado = true;
            }
        }
        return validado;
    }

    private String getMD5(String input){
        StringBuilder sb = new StringBuilder();
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(input.getBytes());
            byte[] digest = md.digest();
            for(byte b : digest){
                sb.append(String.format("%02x", b & 0xff));
            }
        } catch (Exception e){
            sb = new StringBuilder();
        }
        return sb.toString();
    }
}
```

```
public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = getMD5(senha);
}

public void setSenhaCript(String senhaCript){
    this.senha = senhaCript;
}
}
```

```

public class UsuarioDao {
    private final int INITIAL_SIZE = 1000;
    private static UsuarioDao instance;

    private List<Usuario> usuarios;

    private UsuarioDao() {
        usuarios = new LinkedList<>();
    }

    public static UsuarioDao getInstance(){
        if(instance == null){
            instance = new UsuarioDao();
        }
        return instance;
    }

    public int getSize(){
        return usuarios.size();
    }

    public boolean insert(Usuario user){
        boolean deuCerto = false;
        boolean repetido = false;
        for(Usuario u : usuarios){
            if(user.equals(u)){
                repetido = true;
                break;
            }
        }
        if(!repetido) {
            usuarios.add(user);
            Collections.sort(usuarios);
            deuCerto = true;
        }
        return deuCerto;
    }
}

```

Definiu-se um objeto usuários que é uma implementação da interface List, assim, é possível utilizar qualquer classe concreta que implemente List. No caso utilizou-se uma LinkedList.

```

public Usuario recupere(int posicao){
    Usuario user = null;
    if(posicao >= 0 && posicao < usuarios.size()){
        user = usuarios.get(posicao);
    }
    return user;
}

public int searchUsuarioPosition(String username){
    int position=-1;
    int i=0;
    for(Usuario u : usuarios){
        if(u.getUsername().equals(username)){
            position = i;
            break;
        }
        i++;
    }
    return position;
}

```

Essa é a implementação de um **Singleton**. Vejamos.



Singleton – padrão de projetos

- O padrão Singleton permite criar objetos únicos para os quais há apenas uma instância. Este padrão oferece um ponto de acesso global, assim como uma variável global, porém sem as desvantagens das variáveis globais.
- O Padrão Singleton tem como definição garantir que uma classe tenha apenas uma instância de si mesma e que forneça um ponto global de acesso a ela. Ou seja, uma classe gerencia a própria instância dela além de evitar que qualquer outra classe crie uma instância dela.

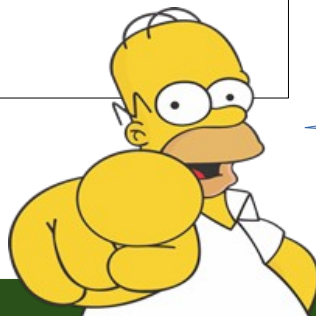
Singleton – padrão de projetos

```
public class UsuarioDao {  
    private final int INITIAL_SIZE = 1000;  
    private static UsuarioDao instance;  
    private List<Usuario> usuarios;  
  
    private UsuarioDao() {  
        usuarios = new LinkedList<>();  
    }  
  
    public static UsuarioDao getInstance(){  
        if(instance == null){  
            instance = new UsuarioDao();  
        }  
        return instance;  
    }  
}
```

instance é um atributo do mesmo tipo da classe, ele será o único objeto da classe `UsuarioDao` que existirá no sistema. Com isso garantimos que haverá apenas uma lista de usuários.

Um construtor **private** garante que o objeto só pode ser instanciado dentro dele mesmo.

O método *getInstance()* devolve uma instância do objeto. Caso o objeto não tenha sido instanciado o método instancia o objeto e retorna a instância.



Explique atributo estático !!!

```
public class UsuarioDao {

    public boolean insert(Usuario user){
        boolean deuCerto = false;
        boolean repetido = false;
        for(Usuario u : usuarios){
            if(user.equals(u)){
                repetido = true;
                break;
            }
        }
        if(!repetido) {
            usuarios.add(user);
            Collections.sort(usuarios);
            deuCerto = true;
        }
        return deuCerto;
    }
}
```

1. inserir usuários sem repetir o username. Implementou-se um foreach (**inadequado!!**) que percorre o List e verifica se o usuário novo (user) é igual ao usuário recuperado do List. O foreach encerra quando o usuário for repetido.

Mas como funciona o equals() de um Usuário?

2. classificar por username. Após inserir o novo usuário (user) no List, basta mandar classificar. De forma mágica tudo acontece!

Nem tanto, é preciso permitir que a classe implemente a interface Comparable para isso. Vejamos.

```
public class Usuario
@Override
public boolean equals(Object obj) {
    boolean iguais = false;
    if(obj != null && obj instanceof Usuario){
        iguais = this.username.equals(((Usuario) obj).getUsername());
    }
    return iguais;
}
```

Todo objeto possui um método equals(), no caso, sobrescreveu-se o método para considerar igual qualquer objeto com o mesmo username.

```
public class Usuario implements Comparable<Usuario>{
    @Override
    public int compareTo(Usuario o) {
        int retorno = 0;
        if(o != null){
            retorno =
                this.username.compareTo(o.getUsername());
        }
        return retorno;
    }
}
```

Ao implementar a interface, deve-se implementar o método compareTo(). Esse retorna 0 se objetos são iguais; 1 se o objeto é maior que o outro; -1 se o objeto é menor que o outro. Sempre assim!!!

Resultado

```
public static void main(String[] args) {

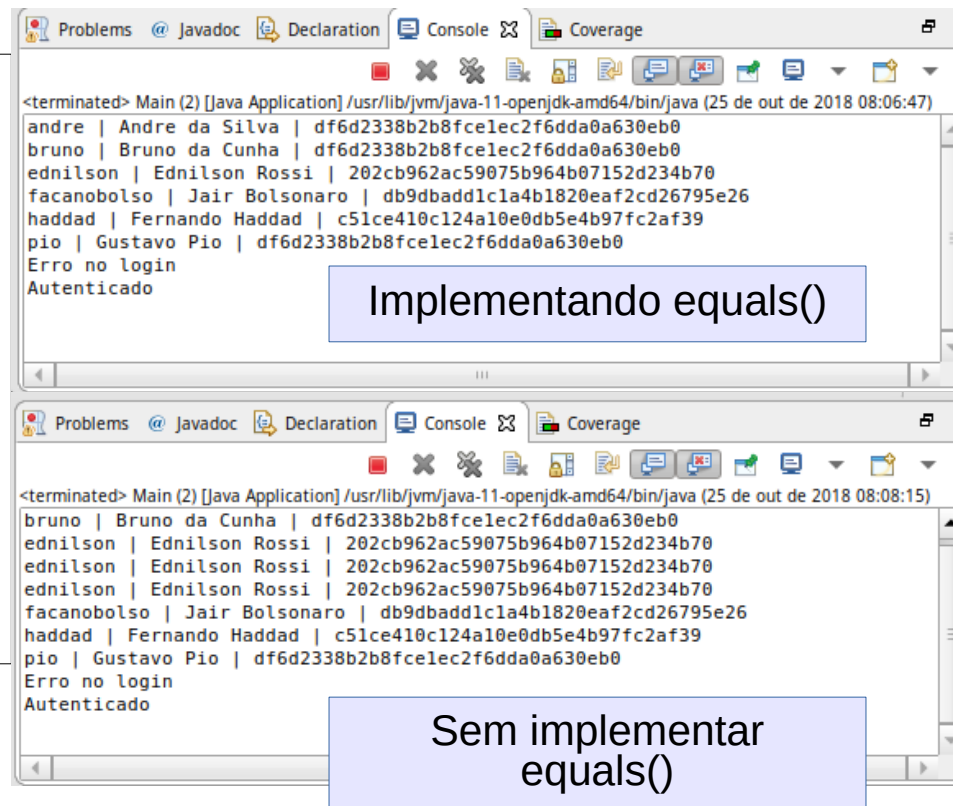
    UsuarioDao dao = UsuarioDao.getInstance();

    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("pio", "Gustavo Pio", "987"));
    dao.insert(new Usuario("andre", "Andre da Silva", "987"));
    dao.insert(new Usuario("bruno", "Bruno da Cunha", "987"));
    dao.insert(new Usuario("facanobolso", "Jair Bolsonaro", "#elenão"));
    dao.insert(new Usuario("haddad", "Fernando Haddad", "13"));

    listaTodos();

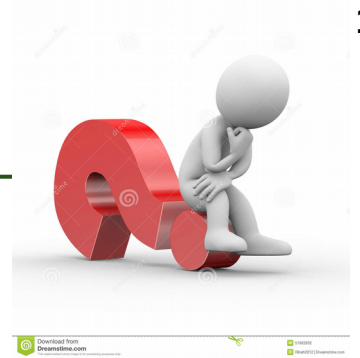
    login("ednilson", "professornota10");
    login("ednilson", "123");

}
```

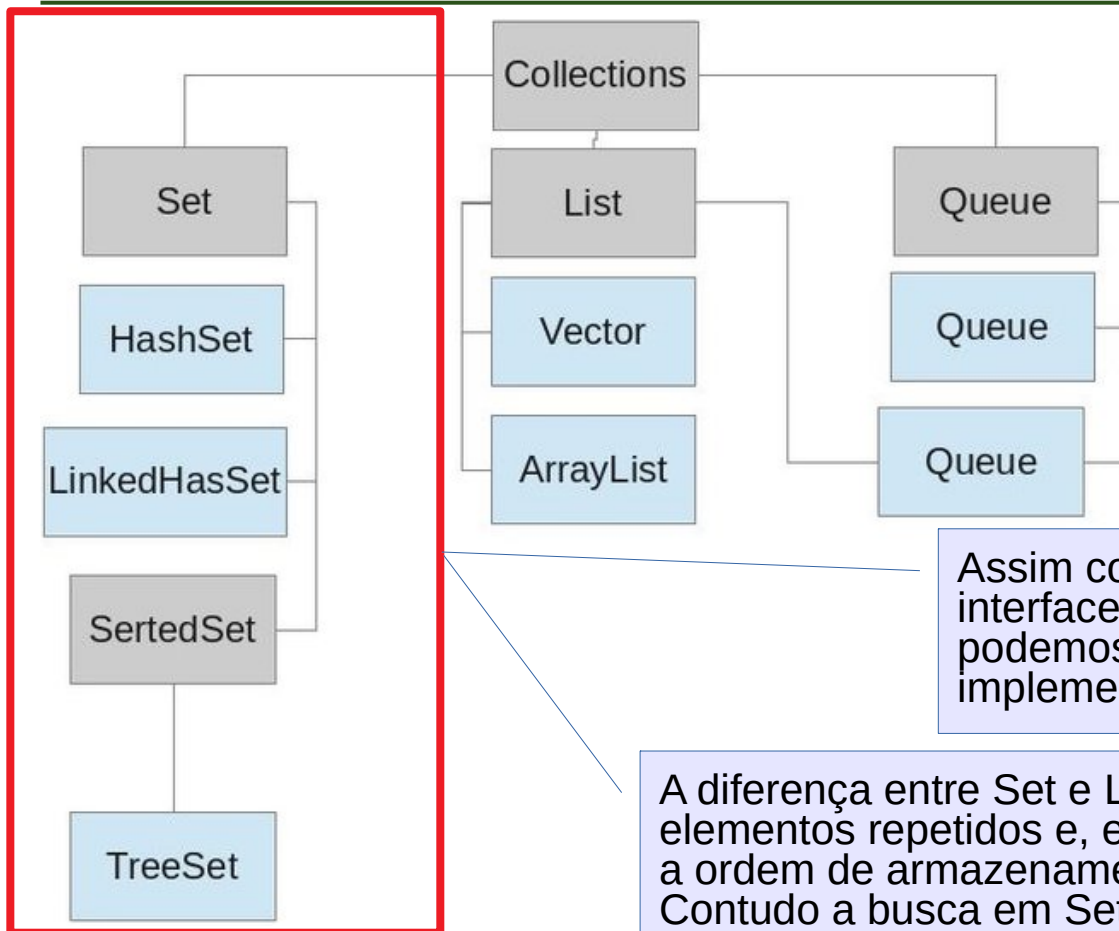


Momento de reflexão

- No exemplo utilizou-se LinkedList.
- Alternativas seriam ArrayList ou Vector.
- Quais são as vantagens e desvantagens?
 - Considere os casos de inserção
 - Pense nas buscas aplicadas



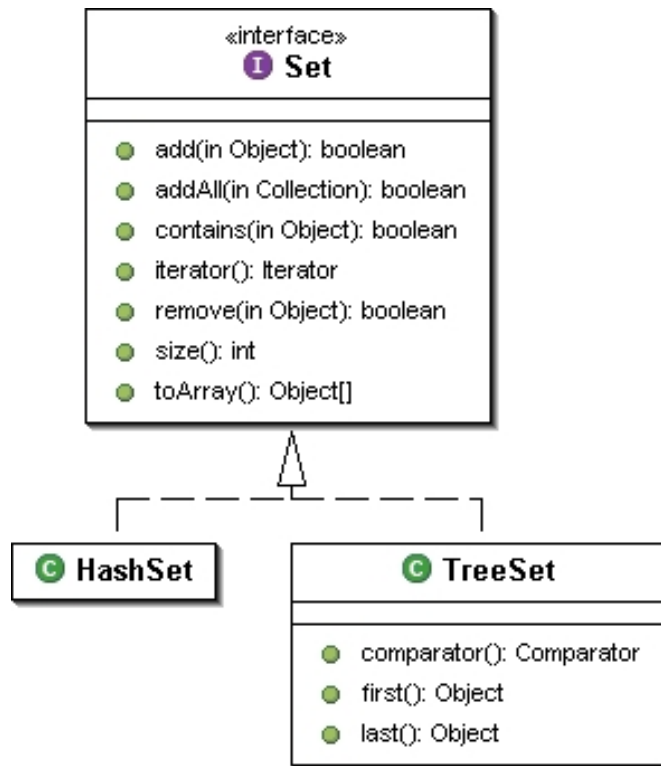
API Java Collection



Assim como não é possível utilizar um List por ser uma interface, um Set também não pode ser utilizado, podemos, contudo, utilizar classes concretas que implementam a interface Set.

A diferença entre Set e List é que Set não permite a inserção de elementos repetidos e, em algumas implementações não sabe-se a ordem de armazenamento dos dados. Contudo a busca em Set é mais eficiente que em List.

API Java Collection



```
public class UsuarioDaoSet {
    private static UsuarioDaoSet instance;

    private Set<Usuario> usuarios;
```

```
private UsuarioDaoSet() {
```

```
    usuarios = new HashSet<>();
```

```
}
```

```
public static UsuarioDaoSet getInstance(){
    if(instance == null){
        instance = new UsuarioDaoSet();
    }
    return instance;
}
```

```
public int getSize(){
    return usuarios.size();
}
```

```
public boolean insert(Usuario user){
    return usuarios.add(user);
}
```

Diferença muito pequena na troca do List pelo Set. Neste exemplo utilizamos o objeto concreto HashSet. Como o Set não permite a inclusão de objetos iguais, não precisamos realizar a validação. Como teste, após exibir o JOptionPane com a mensagem de sucesso, imprime-se o usuário inserido. Observe o resultado.

Foi pego na mentira!!!



```
<terminated> Main2 (2) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (25 de o
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
haddad | Fernando Haddad | c51ce410c124a10e0db5e4b97fc2af39
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
andre | Andre da Silva | df6d2338b2b8fcec2f6dda0a630eb0
pio | Gustavo Pio | df6d2338b2b8fcec2f6dda0a630eb0
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
bruno | Bruno da Cunha | df6d2338b2b8fcec2f6dda0a630eb0
facanobolso | Jair Bolsonaro | db9dbadd1c1a4b1820eaf2cd26795e26
```

```
public static void main(String[] args) {
    UsuarioDaoSet dao = UsuarioDaoSet.getInstance();
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("pio", "Gustavo Pio", "987"));
    dao.insert(new Usuario("andre", "Andre da Silva", "987"));
    dao.insert(new Usuario("bruno", "Bruno da Cunha", "987"));
    dao.insert(new Usuario("facanobolso", "Jair Bolsonaro", "#elenão"));
    dao.insert(new Usuario("haddad", "Fernando Haddad", "13"));
    listaTodos();
}
```

Esclarecendo...

```
public class Usuario implements Comparable<Usuario>{
```

```
    private String username;
```

```
    private String nome;
```

```
    private String senha;
```

```
    public Usuario(String username, String nome, String senha) {
```

```
        this.username = username;
```

```
        this.nome = nome;
```

```
        setSenha(senha);
```

```
    }
```

```
    public boolean autenticar(String username, String senha) {
```

```
        boolean valido = false;
```

```
        //suprimido
```

```
        return valido;
```

```
    }
```

```
    private String getMD5(String senha) {
```

```
        //suprimido
```

```
    }
```

```
//gets e sets
```

Classe Usuário contém os métodos equals() e compareTo()

```
@Override
```

```
public boolean equals(Object obj) {
```

```
    boolean iguais = false;
```

```
    if(obj != null && obj instanceof Usuario){
```

```
        iguais = this.username.equals(((Usuario) obj).getUsername());
```

```
    }
```

```
    return iguais;
```

```
}
```

```
@Override
```

```
public int compareTo(Usuario o) {
```

```
    int retorno = 0;
```

```
    if(o != null){
```

```
        retorno = this.username.compareTo(o.getUsername());
```

```
    }
```

```
    return retorno;
```

```
}
```

```
}
```



Como!??

- Se um objeto é igual ao outro, como o Set está permitindo a inserção de objetos iguais? Ele funciona de verdade?
- O Set faz primeiro uma busca pelo Hash dos objetos para depois verificar se são iguais.

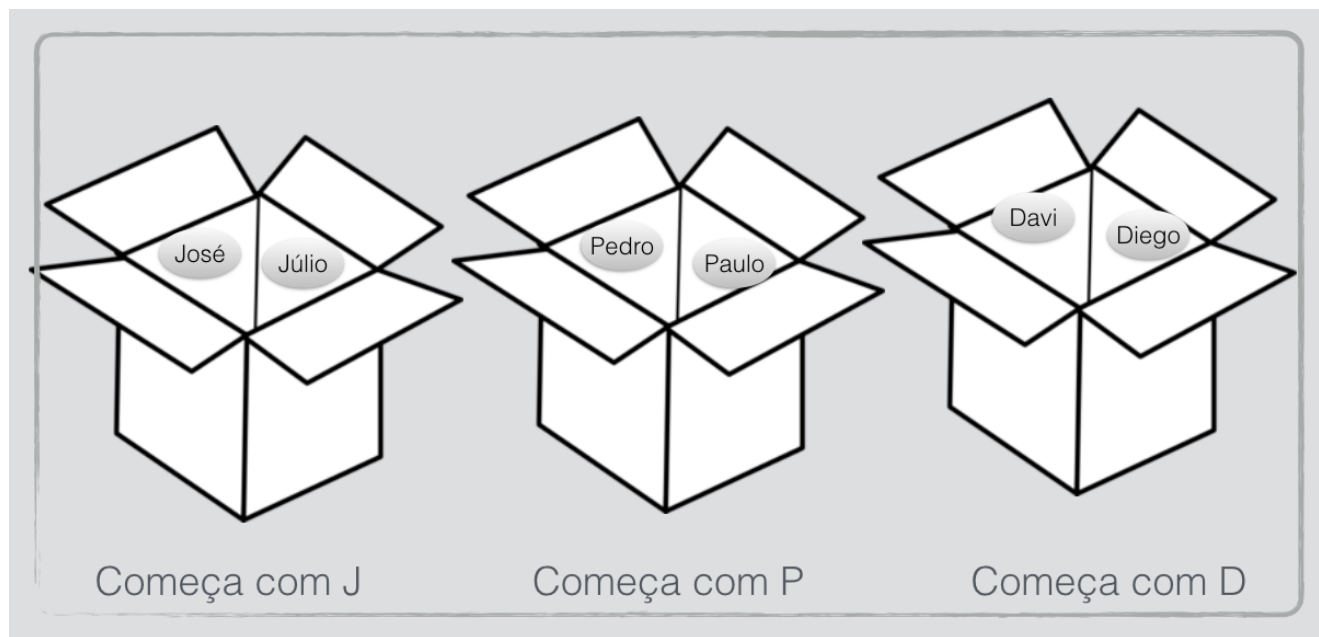
```
@Override  
public int hashCode() {  
    return 1;  
}
```

Colega,
sobrescreva o
método hashCode()
de Usuário.



HashCode

- Região dentro da coleção onde os objetos ficam agrupados por semelhança, facilitando assim os encontrar.



HashCode

- É usado para comparar objetos dentro de coleções?
 - Não é usado para comparar, mas é usado para encontrar objetos dentro da coleção, pois primeiro procura-se o grupo ao qual o objeto pertence em seguida procura-se o objeto fazendo a comparação de objetos, logo, se não for possível encontrar o grupo não será possível encontrar o objeto. A comparação de objetos é feita através do resultado do método equals().
 - A implementação correta do hashCode() é aquela que sempre retorna o mesmo valor quando chamado para um mesmo objeto, de acordo com o **contrato do hashCode()**.

HashCode - Javadoc

```
public int hashCode()
```

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by HashMap.

The general contract of hashCode is:

Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.

It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)

Returns:

a hash code value for this object.

HashCode

- Além de implementar corretamente deve-se sempre procurar implementar o hashCode() eficientemente.
- Quando implementado de uma maneira eficiente ele ajuda as coleções a eliminarem vários objetos que com certeza não são o que está sendo procurado, ou seja, ele descarta os grupos de objetos que não produzem determinado hashCode().
- Um exemplo de implementação de hashCode() ineficiente:

```
public int hashCode() { return 42; } //é válido, porém ineficiente
```
- É ineficiente pois todos os objetos estarão em um mesmo grupo, dificultando o trabalho da coleção ao se procurar um objeto. A forma mais eficiente de implementação é se você conseguir garantir uma forma de gerar hashCode() únicos para cada objeto, assim você estará garantindo que haverá apenas um objeto por grupo de hashCode() dentro da coleção.

```
public class Usuario implements Comparable<Usuario>{

    private String username;
    private String nome;
    private String senha;

    public Usuario(String username, String nome, String s) {
        this.username = username;
        this.nome = nome;
        setSenha(senha);
    }

    public boolean autenticar(String username, String s) {
        /*suprimido*/
    }

    private String getMD5(String input){
        /*Suprimido*/
    }

    /*gets e sets*/
}
```

```
@Override
public String toString() {

    return username + " | " + nome + " | " + senha;

}

@Override
public boolean equals(Object obj) {
    boolean iguais = false;
    if(obj != null && obj instanceof Usuario){
        iguais = this.username.equals(((Usuario) obj).getUsername());
    }
    return iguais;
}

@Override
public int compareTo(Usuario o) {
    int retorno = 0;
    if(o != null){
        retorno = this.username.compareTo(o.getUsername());
    }
    return retorno;
}

@Override
public int hashCode() {
    return username.hashCode();
}
}
```

O Set não possui o método get() como no List, então para acessar os dados do Set deve-se utilizar um Iterator e a partir deste acessar os objetos do Set.

A princípio não parece ser interessante esse uso, contudo o Set é mais eficiente que o List na recuperação de dados e isso pode justificar seu uso.



```
public class UsuarioDao {
    private static UsuarioDao instance;
    private Set<Usuario> usuarios;

    private UsuarioDao() {
        usuarios = new HashSet<>();
    }
    public static UsuarioDao getInstance(){
        if(instance == null){
            instance = new UsuarioDao();
        }
        return instance;
    }
    public int getSize(){
        return usuarios.size();
    }

    public boolean insert(Usuario user){
        return usuarios.add(user);
    }

    public Usuario recupere(String username){
        Usuario usuario;
        Usuario retorno = null;
        Iterator<Usuario> iterator = usuarios.iterator();
        while (iterator.hasNext() && retorno == null){
            usuario = iterator.next();
            if(usuario.getUsername().equals(username)){
                retorno = usuario;
            }
        }
        return retorno;
    }
}
```

```

public class UsuarioDaoSet {
    private static UsuarioDaoSet instance;
    private Set<Usuario> usuarios;

    private UsuarioDaoSet() {
        usuarios = new HashSet<>();
    }

    public static UsuarioDaoSet getInstance(){
        if(instance == null){
            instance = new UsuarioDaoSet();
        }
        return instance;
    }

    public int getSize(){
        return usuarios.size();
    }

    public boolean insert(Usuario user){
        return usuarios.add(user);
    }

    public Usuario recuperar(String username){
        Usuario usuario;
        Usuario retorno = null;
        Iterator<Usuario> iterator = usuarios.iterator();
        while (iterator.hasNext() && retorno == null){
            usuario = iterator.next();
            if(usuario.getUsername().equals(username)){
                retorno = usuario;
            }
        }
        return retorno;
    }
}

```

```

public class UsuarioDaoSet2 {
    private static UsuarioDaoSet2 instance;
    private Set<Usuario> usuarios;

    private UsuarioDaoSet2() {
        usuarios = new LinkedHashSet<>();
    }

    public static UsuarioDaoSet2 getInstance(){
        if(instance == null){
            instance = new UsuarioDaoSet2();
        }
        return instance;
    }

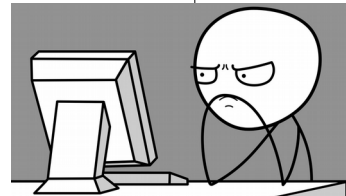
    public int getSize(){
        return usuarios.size();
    }

    public boolean insert(Usuario user){
        return usuarios.add(user);
    }

    public Usuario recuperar(String username){
        Usuario usuario;
        Usuario retorno = null;
        Iterator<Usuario> iterator = usuarios.iterator();
        while (iterator.hasNext() && retorno == null){
            usuario = iterator.next();
            if(usuario.getUsername().equals(username)){
                retorno = usuario;
            }
        }
        return retorno;
    }
}

```

Qual a
diferença?
Alguém
explica?



```

<terminated> Main2 (2) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (25 de out de 2018 09:07:44)
andre | Andre da Silva | df6d2338b2b8fcec2f6dda0a630eb0
haddad | Fernando Haddad | c51ce410c124a10e0db5e4b97fc2af39
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
pio | Gustavo Pio | df6d2338b2b8fcec2f6dda0a630eb0
bruno | Bruno da Cunha | df6d2338b2b8fcec2f6dda0a630eb0
facanobolso | Jair Bolsonaro | db9dbadd1c1a4b1820eaf2cd26795e26

```

```

<terminated> Main3 [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (25 de out de 2018 09:08:32)
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
pio | Gustavo Pio | df6d2338b2b8fcec2f6dda0a630eb0
andre | Andre da Silva | df6d2338b2b8fcec2f6dda0a630eb0
bruno | Bruno da Cunha | df6d2338b2b8fcec2f6dda0a630eb0
facanobolso | Jair Bolsonaro | db9dbadd1c1a4b1820eaf2cd26795e26
haddad | Fernando Haddad | c51ce410c124a10e0db5e4b97fc2af39

```

```

public static void main(String[] args) {
    UsuarioDaoSet2 dao = UsuarioDaoSet2.getInstance();

    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
    dao.insert(new Usuario("pio", "Gustavo Pio", "987"));
    dao.insert(new Usuario("andre", "Andre da Silva", "987"));
    dao.insert(new Usuario("bruno", "Bruno da Cunha", "987"));
    dao.insert(new Usuario("facanobolso", "Jair Bolsonaro", "#elenão"));
    dao.insert(new Usuario("haddad", "Fernando Haddad", "13"));

    listaTodos();
}

```

```

public class UsuarioDaoSetTodos {
    private static UsuarioDaoSetTodos instance;
    private Set<Usuario> usuariosHashSet;
    private Set<Usuario> usuariosLinkedHashSet;
    private Set<Usuario> usuariosTreeSet;
    private UsuarioDaoSetTodos() {
        usuariosHashSet = new HashSet<>();
        usuariosLinkedHashSet = new LinkedHashSet<>();
        usuariosTreeSet = new TreeSet<>();
    }
    public boolean insert(Usuario user){
        return usuariosHashSet.add(user) && usuariosLinkedHashSet.add(user) && usuariosTreeSet.add(user);
    }
    public String getTodos(){
        Iterator<Usuario> iterator;
        StringBuilder sb = new StringBuilder();
        sb.append("HashSet\n");
        iterator = usuariosHashSet.iterator();
        while(iterator.hasNext()){
            sb.append(iterator.next().toString());
            sb.append("\n");
        }
        sb.append("\nLinkedHashSet\n");
        iterator = usuariosLinkedHashSet.iterator();
        while(iterator.hasNext()){
            sb.append(iterator.next().toString());
            sb.append("\n");
        }
        sb.append("\nTreeSet\n");
        iterator = usuariosTreeSet.iterator();
        while (iterator.hasNext()){
            sb.append(iterator.next().toString());
            sb.append("\n");
        }
        return sb.toString();
    }
}

```


Não existe lógica na organização dos dados.

```
public static void main(String[] args) {
    UsuarioDaoSetTodos dao = UsuarioDaoSetTodos.getInstance();

    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "202cb962ac59075b964b07152d234b70"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "202cb962ac59075b964b07152d234b70"));
    dao.insert(new Usuario("ednilson", "Ednilson Rossi", "202cb962ac59075b964b07152d234b70"));
    dao.insert(new Usuario("pio", "Gustavo Pio", "987"));
    dao.insert(new Usuario("andre", "Andre da Silva", "df6d2338b2b8fcec2f6dda0a630eb0"));
    dao.insert(new Usuario("bruno", "Bruno da Cunha", "df6d2338b2b8fcec2f6dda0a630eb0"));
    dao.insert(new Usuario("facanobolso", "Jair Bolsonaro", "db9dbadd1c1a4b1820eaf2cd26795e26"));
    dao.insert(new Usuario("haddad", "Fernando Haddad", "c51ce410c124a10e0db5e4b97fc2af39"));

    System.out.println(dao.getTodos());
}
```

Por ser uma lista os dados são inseridos sequencialmente no final da lista.

Como pode-se esperar de uma árvore, os dados são retornados em percurso "em ordem".

```

terminated> MainSetTodos [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (25 de
HashSet
andre | Andre da Silva | df6d2338b2b8fcec2f6dda0a630eb0
haddad | Fernando Haddad | c51ce410c124a10e0db5e4b97fc2af39
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
pio | Gustavo Pio | df6d2338b2b8fcec2f6dda0a630eb0
bruno | Bruno da Cunha | df6d2338b2b8fcec2f6dda0a630eb0
facanobolso | Jair Bolsonaro | db9dbadd1c1a4b1820eaf2cd26795e26

LinkedHashSet
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
pio | Gustavo Pio | df6d2338b2b8fcec2f6dda0a630eb0
andre | Andre da Silva | df6d2338b2b8fcec2f6dda0a630eb0
bruno | Bruno da Cunha | df6d2338b2b8fcec2f6dda0a630eb0
facanobolso | Jair Bolsonaro | db9dbadd1c1a4b1820eaf2cd26795e26
haddad | Fernando Haddad | c51ce410c124a10e0db5e4b97fc2af39

TreeSet
andre | Andre da Silva | df6d2338b2b8fcec2f6dda0a630eb0
bruno | Bruno da Cunha | df6d2338b2b8fcec2f6dda0a630eb0
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
facanobolso | Jair Bolsonaro | db9dbadd1c1a4b1820eaf2cd26795e26
haddad | Fernando Haddad | c51ce410c124a10e0db5e4b97fc2af39
pio | Gustavo Pio | df6d2338b2b8fcec2f6dda0a630eb0
  
```

```

public class UsuarioDaoQueue {

    private static UsuarioDaoQueue instance;
    private Queue<Usuario> usuarios;

    private UsuarioDaoQueue() {
        usuarios = new LinkedList<>();
    }

    public static UsuarioDaoQueue getInstance(){
        /* */
    }

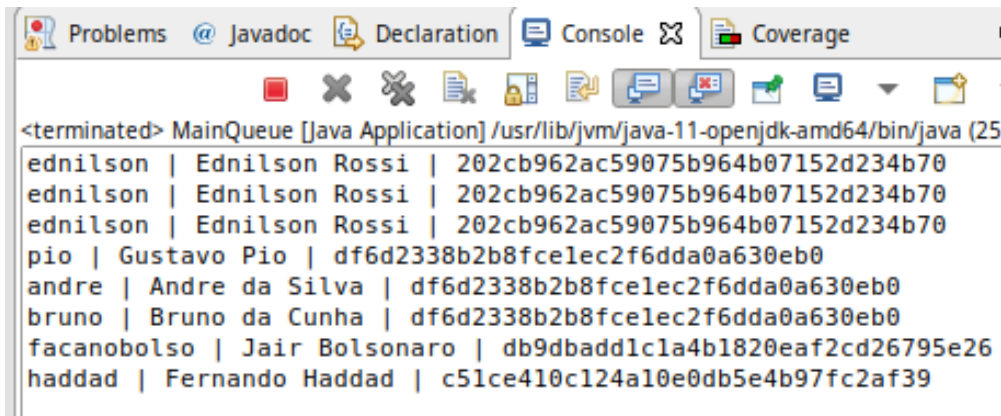
    public int getSize(){
        return usuarios.size();
    }

    public void insert(Usuario user){
        usuarios.add(user);
    }

    public String getAll(){
        StringBuilder sb = new StringBuilder();
        while (!usuarios.isEmpty()){
            sb.append(usuarios.remove().toString());
            sb.append("\n");
        }
        return sb.toString();
    }
}

```

Queue



```

<terminated> MainQueue [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (25
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
ednilson | Ednilson Rossi | 202cb962ac59075b964b07152d234b70
pio | Gustavo Pio | df6d2338b2b8fceleec2f6dda0a630eb0
andre | Andre da Silva | df6d2338b2b8fceleec2f6dda0a630eb0
bruno | Bruno da Cunha | df6d2338b2b8fceleec2f6dda0a630eb0
facanobolso | Jair Bolsonaro | db9dbadd1c1a4b1820eaf2cd26795e26
haddad | Fernando Haddad | c51ce410c124a10e0db5e4b97fc2af39

```

```

    public static void main(String[] args) {
        UsuarioDaoQueue dao = UsuarioDaoQueue.getInstance();

        insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
        insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
        insert(new Usuario("ednilson", "Ednilson Rossi", "123"));
        insert(new Usuario("pio", "Gustavo Pio", "987"));
        insert(new Usuario("andre", "Andre da Silva", "987"));
        insert(new Usuario("bruno", "Bruno da Cunha", "987"));
        insert(new Usuario("facanobolso", "Jair Bolsonaro", "#elenão"));
        insert(new Usuario("haddad", "Fernando Haddad", "13"));

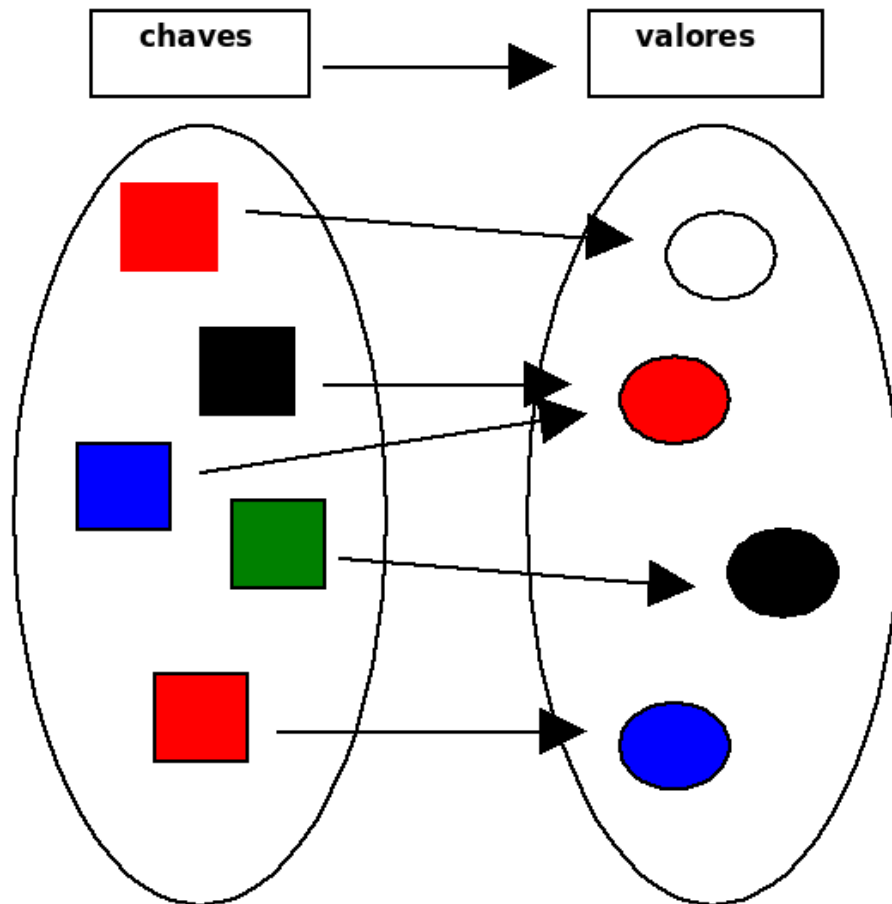
        System.out.println(dao.getAll());
    }
}

```

Map

- Mudando um pouco de assunto, mas nem tanto, vejamos o Map. Esse tipo de não faz parte da API Collection porém pode auxiliar na busca de dados.
 - Muitas vezes queremos buscar rapidamente um objeto dado alguma informação sobre ele. Um exemplo seria, dada a placa do carro, obter todos os dados do carro. Poderíamos utilizar uma lista para isso e percorrer todos os seus elementos, mas isso pode ser péssimo para a performance, mesmo para listas não muito grandes.
 - Um mapa é composto por um conjunto de associações entre um objeto chave a um objeto valor. É equivalente ao conceito de dicionário, usado em várias linguagens. Algumas linguagens, como Perl ou PHP, possuem um suporte mais direto a mapas, onde são conhecidos como matrizes/arrays associativas.
 - `java.util.Map` é um mapa, pois é possível usá-lo para mapear uma chave a um valor, por exemplo: mapeie à chave "empresa" o valor "Sun Microsystems", ou então mapeie à chave "rua" ao valor "Bento de Abreu". Semelhante a associações de palavras que podemos fazer em um dicionário.

Map



Possíveis ações em um mapa:

Mapeie uma chave a um valor
O que está mapeado na chave X?
Remapeie uma certa chave
Quero o conjunto de chaves.
Quero o conjunto de valores.
Desmapeie a chave X.

Map - Exemplo

```
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {

        Map<String, Usuario> mapaDeUsuario = new HashMap<>();

        mapaDeUsuario.put("ednilson", new Usuario("ednilson", "Ednilson", "123"));
        mapaDeUsuario.put("george", new Usuario("george", "George", "123"));
        mapaDeUsuario.put("caio", new Usuario("caio", "Caio", "123"));
        mapaDeUsuario.put("igor", new Usuario("igor", "Igor", "123"));

        Usuario usuario = mapaDeUsuario.get("caio");
        System.out.println(usuario);

    }
}
```

Trabalhando

- Exercício Avaliativo:
 - 10, 11 e 12



Material Adicional

31



- **Singleton**

- <https://www.devmedia.com.br/padrao-de-projeto-singleton-em-java/26392>
- <https://www.devmedia.com.br/trabalhando-com-singleton-java/23632>

- **Java Collection (List, Set, Queue e Map)**

- <https://www.devmedia.com.br/linkedlists-o-que-acontece-por-tras-da-interface/24613> **(Obrigatório)**
- <https://docs.oracle.com/javase/8/docs/api/java/util/AbstractSet.html>
- <https://www.devmedia.com.br/java-collections-set-list-e-iterator/29637>
- <http://blog.caelum.com.br/performance-hashset-em-vez-de-arraylist/>
- <https://pt.stackoverflow.com/questions/40196/em-que-ordem-uma-set-%C3%A9-armazenada-aleat%C3%B3ria>
- <https://www.caelum.com.br/apostila-java-orientacao-objetos/collections-framework/#mapas---javautlmap>
- <https://www.devmedia.com.br/api-collections-em-java-fundamentos-e-implementacao-basica/28445>
- <https://www.devmedia.com.br/diferenca-entre-arraylist-vector-e-linkedlist-em-java/29162>
- <http://tutorials.jenkov.com/java-collections/queue.html>

- **HashCode e Equals**

- <http://blog.caelum.com.br/ensinando-que-e-o-hashcode/>
- <http://www.matera.com.br/2015/01/15/desvendando-os-metodos-equals-e-hashcode/>
- <https://pt.stackoverflow.com/questions/11108/qual-a-import%C3%A2ncia-de-implementar-o-m%C3%A9todo-hashcode-em-java#11114>
- <http://www.guj.com.br/t/resolvido-real-necessidade-do-hashcode/138573>