

Análise e desenvolvimento de aplicações orientadas a objeto com Java SE

Classe de Modelagem

Codificação da classe Usuario

Atributos

```
public class Usuario {
```

```
    // Atributos
    private String nome;
    private String email;
    private String login;
    private String senha;
```

Construtores

```
    // Construtores
    // Inicializa os atributos vazios
    public Usuario() {
        this("", "", "", "");
    }
    // Inicializa os atributos com valores passados por parametro
    public Usuario(String email, String login, String nome, String senha) {
        this.email = email;
        this.login = login;
        this.nome = nome;
        this.senha = senha;
    }
}
```

Getters e Setters

```
    // Getters e Setters
    public String getNome() {
        return nome;
    }

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

    // Implementação dos demais getters e setters

    // Métodos específicos da classe
    public void provarExistencia() {
        System.out.println("Oi, eu existo!");
    }
}
```

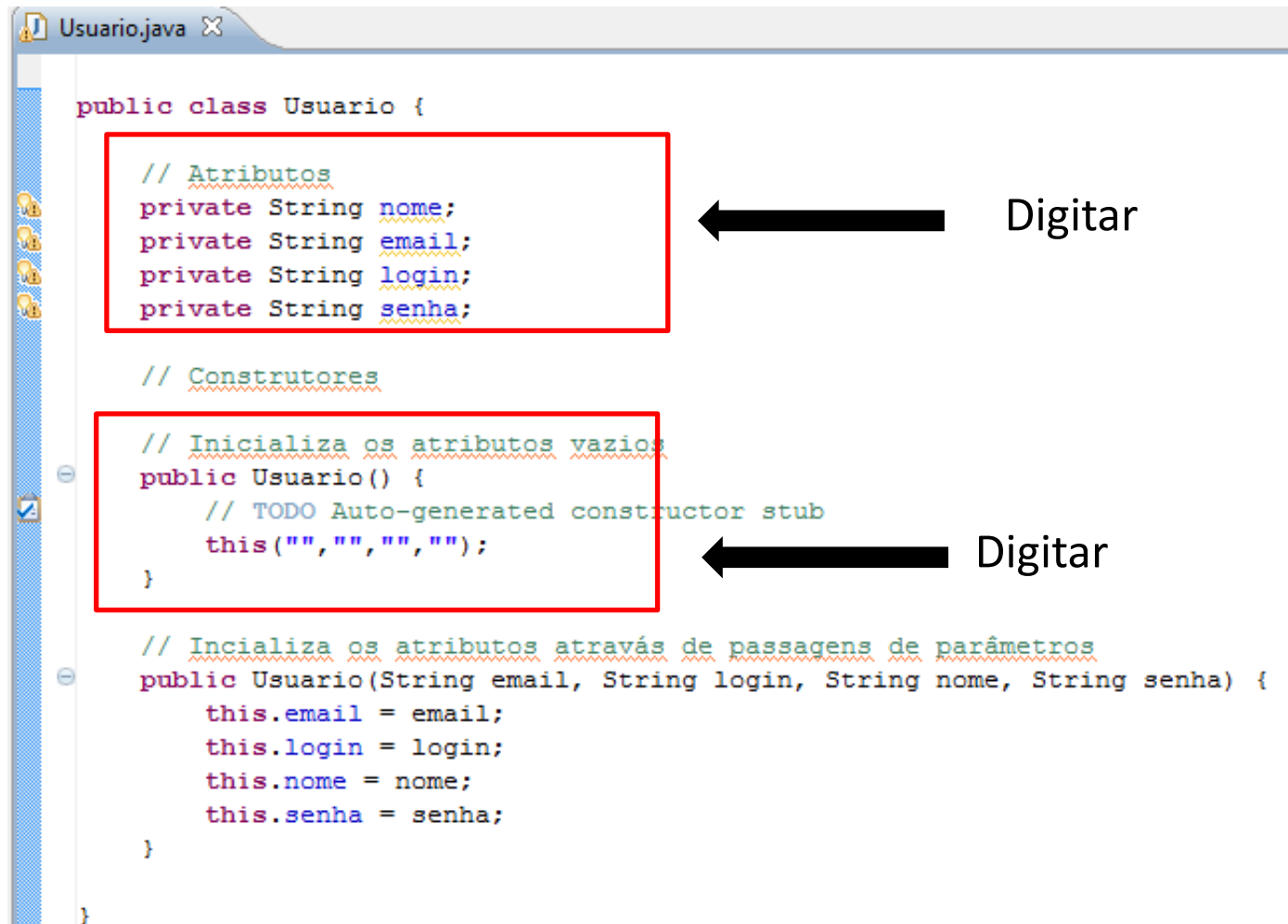


Estrutura básica de um projeto Java no NetBeans Apache

- Projeto (Estrutura de pastas e arquivos que compõem a aplicação)
 - Classe de modelagem (Definição de tipo)
 - Atributo
 - Construtores
 - Getters e setters
 - Métodos específicos da classe
 - Classe Principal
 - Método main()

Criação da estrutura da classe de modelagem

Atributos digitados e Construtores gerados



```
public class Usuario {  
  
    // Atributos  
    private String nome;  
    private String email;  
    private String login;  
    private String senha;  
  
    // Construtores  
  
    // Inicializa os atributos vazios  
    public Usuario() {  
        // TODO Auto-generated constructor stub  
        this("", "", "", "");  
    }  
  
    // Inicializa os atributos através de passagens de parâmetros  
    public Usuario(String email, String login, String nome, String senha) {  
        this.email = email;  
        this.login = login;  
        this.nome = nome;  
        this.senha = senha;  
    }  
}
```

Criação do construtor e dos gets e sets automaticamente

* * * * para criar construtores (com passagem de valores) automaticamente::

- alt + insert
- no submenu escolher construtor , escolher campos e ok....

* * * * para criar getters e setter automaticamente:

- Menu refatorar
- encapsular campos
- seleccionar tudo

(visualizar) - fazer refatoração.

Criação da estrutura da classe de modelagem

getters e setters gerados

```
Usuario.java X
public String getNome() {
    return nome;
}

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

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getSenha() {
    return senha;
}

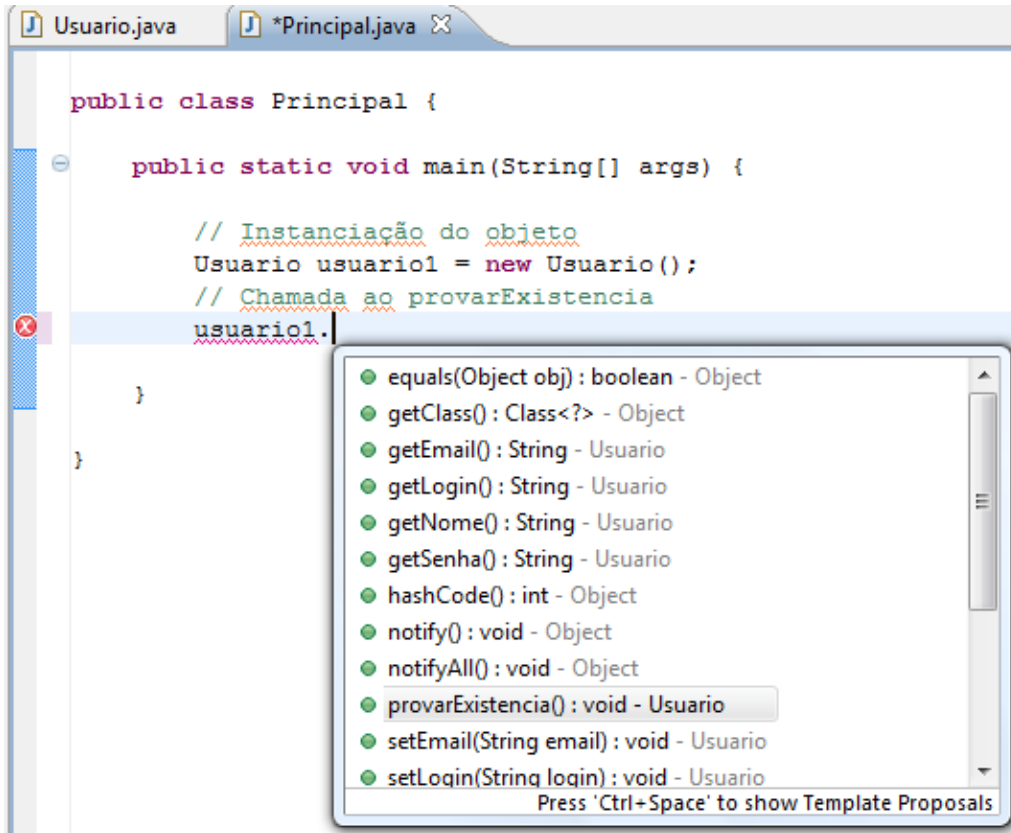
public void setSenha(String senha) {
    this.senha = senha;
}
```

Implementar (digitar) o método
provarExistencia

```
public void provarExistencia() {
    System.out.println("Oi, eu existo!");
}

}
```

Criação da classe Principal

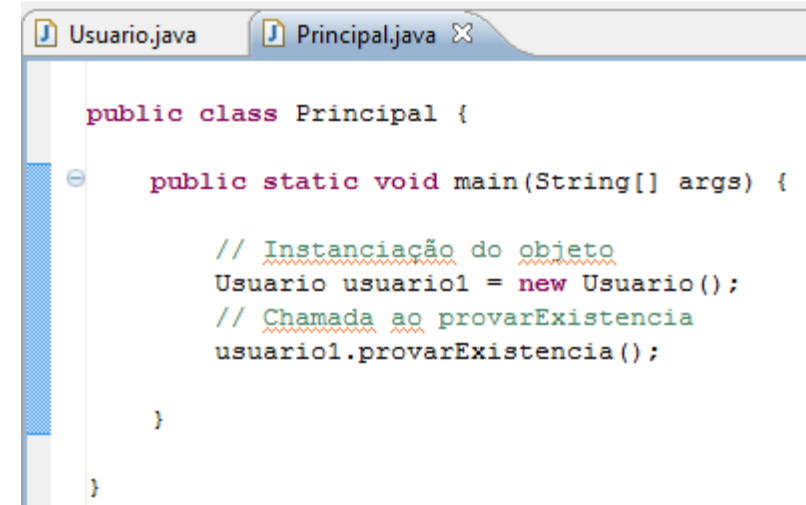


```
public class Principal {  
    public static void main(String[] args) {  
        // Instanciação do objeto  
        Usuario usuario1 = new Usuario();  
        // Chamada ao provarExistencia  
        usuario1.  
    }  
}
```

- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- getEmail() : String - Usuario
- getLogin() : String - Usuario
- getNome() : String - Usuario
- getSenha() : String - Usuario
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- **provarExistencia() : void - Usuario**
- setEmail(String email) : void - Usuario
- setLogin(String login) : void - Usuario

Press 'Ctrl+Space' to show Template Proposals

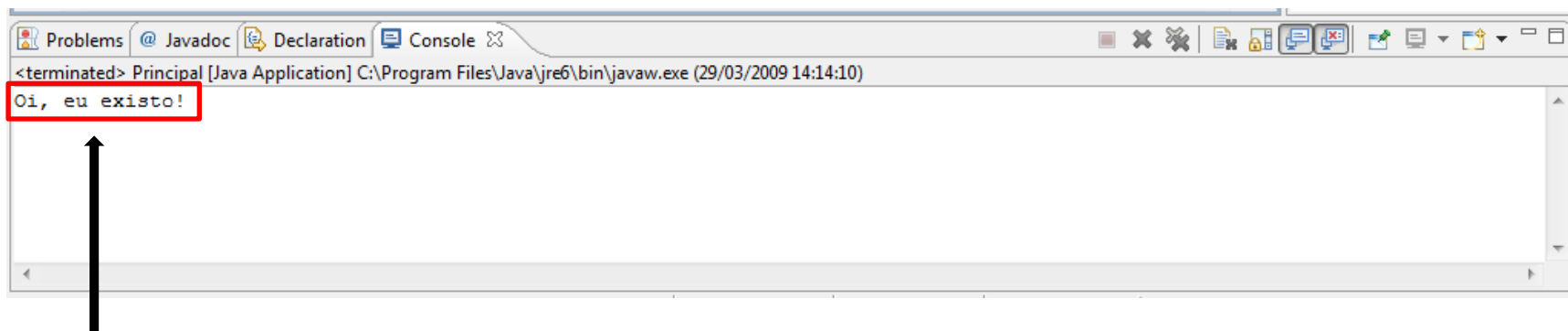
O objeto usuario1 possui todos os métodos definidos na classe Usuario e mais alguns pré-definidos por default



```
public class Principal {  
    public static void main(String[] args) {  
        // Instanciação do objeto  
        Usuario usuario1 = new Usuario();  
        // Chamada ao provarExistencia  
        usuario1.provarExistencia();  
    }  
}
```

Executando um programa em java

O painel console simula um prompt em ambiente texto



Enfim, a emocionante manifestação de vida do objeto usuario1

Incrementando nossa aplicação

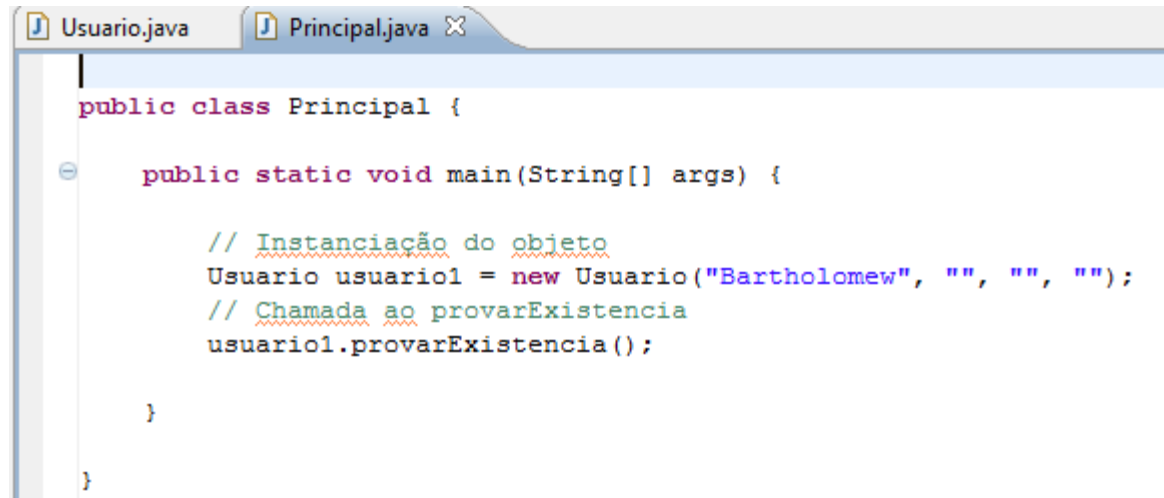
Inserindo o conteúdo do atributo um nome a mensagem

Classe Usuario: Alterar o método provarExistencia

```
public void provarExistencia() {  
    System.out.println("Oi, meu nome é " + this.getNome() + ", e eu existo!");  
}  
}
```

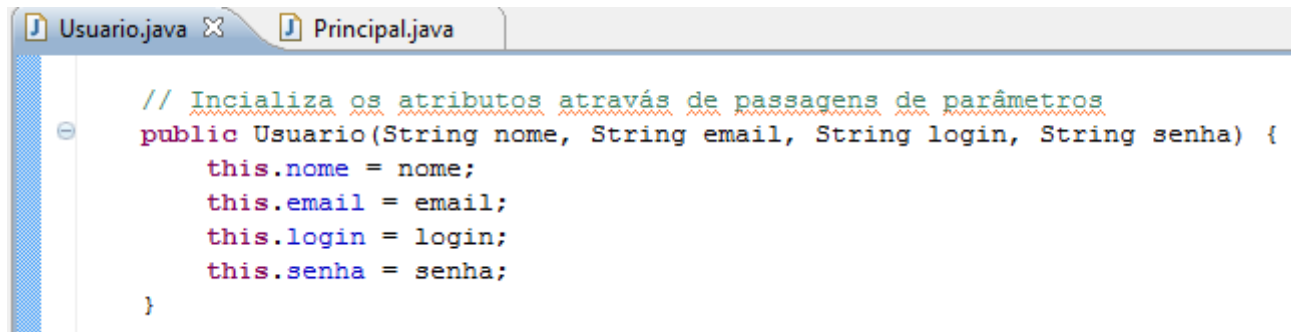
Incrementando nossa aplicação

Inicializar o atributo nome do objeto usuario1 através do construtor

A screenshot of an IDE window showing the file 'Principal.java'. The code defines a 'Principal' class with a 'main' method. Inside 'main', a 'Usuario' object named 'usuario1' is instantiated with the name 'Bartholomew' and empty strings for email, login, and password. Then, the 'provarExistencia()' method of 'usuario1' is called. The 'Usuario.java' file is also visible in the background.

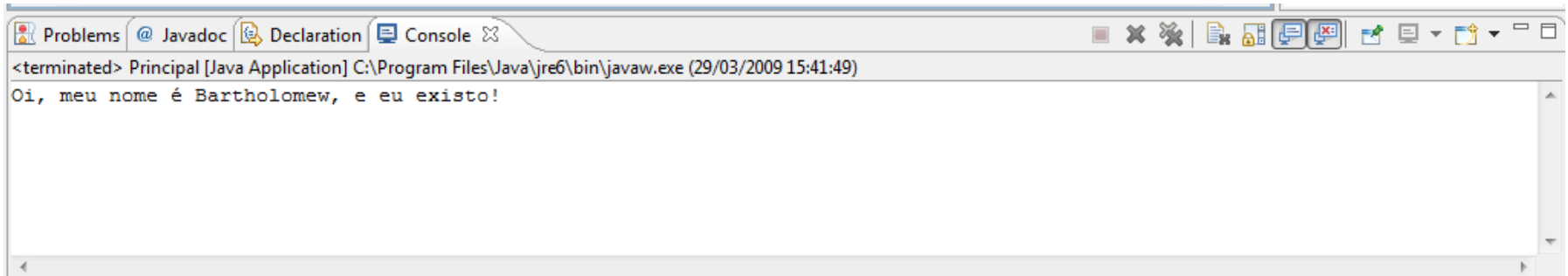
```
public class Principal {  
  
    public static void main(String[] args) {  
  
        // Instanciação do objeto  
        Usuario usuario1 = new Usuario("Bartholomew", "", "", "");  
        // Chamada ao provarExistencia  
        usuario1.provarExistencia();  
  
    }  
  
}
```

Construtor na classe Usuario

A screenshot of an IDE window showing the file 'Usuario.java'. The code defines a 'Usuario' class with a constructor that takes four parameters: 'nome', 'email', 'login', and 'senha'. The constructor assigns these parameters to the instance variables 'this.nome', 'this.email', 'this.login', and 'this.senha'. The 'Principal.java' file is also visible in the background.


```
// Inicializa os atributos através de passagens de parâmetros  
public Usuario(String nome, String email, String login, String senha) {  
    this.nome = nome;  
    this.email = email;  
    this.login = login;  
    this.senha = senha;  
}
```

Executando novamente a aplicação aplicação




The screenshot shows a console window from an IDE. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console text reads: '<terminated> Principal [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (29/03/2009 15:41:49)' followed by the output 'Oi, meu nome é Bartholomew, e eu existo!'. The window has standard OS controls (minimize, maximize, close) and a toolbar with icons for running, debugging, and other IDE functions.

```
<terminated> Principal [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (29/03/2009 15:41:49)
Oi, meu nome é Bartholomew, e eu existo!
```



Recapitulando nosso “HelloWorld” orientado a objeto

- Em uma aplicação orientada a objetos quem manipula dados e ações são os objetos.
- Através de análise e abstração definimos as entidades a serem representadas na aplicação e criamos uma classe (um tipo) com todos os dados e ações especificados (classe de modelagem).
- Definido o tipo, criamos (instanciamos) objetos a partir dos tipos pré-existentes (por enquanto, sempre no método main).



Recapitulando nosso “HelloWorld” orientado a objeto

- Criado o objeto, basta manipulá-lo.
- Uma classe de modelagem da origem a quantos objetos forem necessários na aplicação.
- Qualquer alteração e/ou inserção nos atributos ou métodos são realizados na classe de modelagem e automaticamente todos os objetos instanciados a partir dessa classe serão atualizados automaticamente.