

Padrões de projeto estruturais

Saulo Medeiros de Araujo

@sma

sma@cesar.school

Padrões estruturais

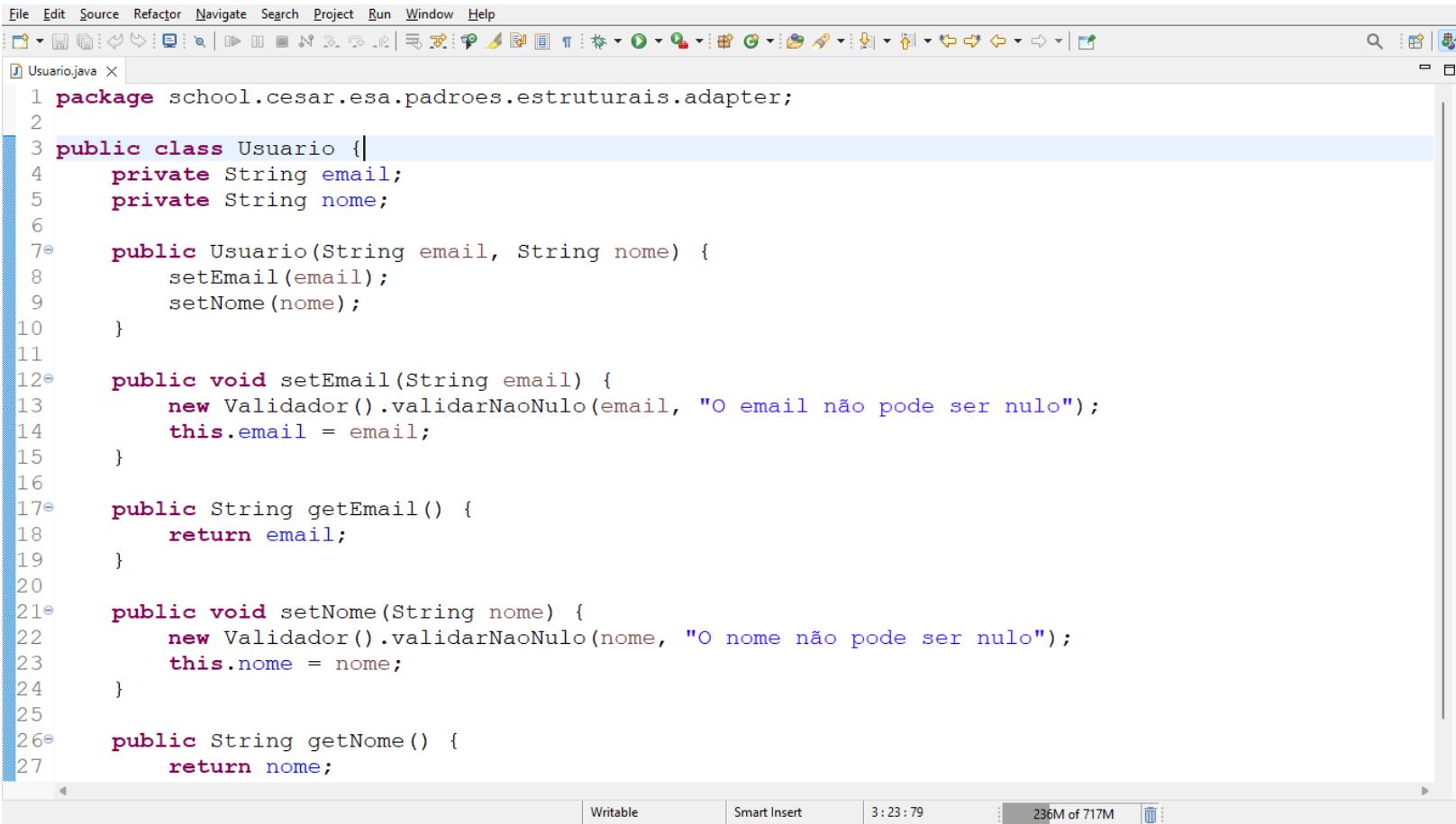
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Adapter

Adaptar-se é preciso



Adapter

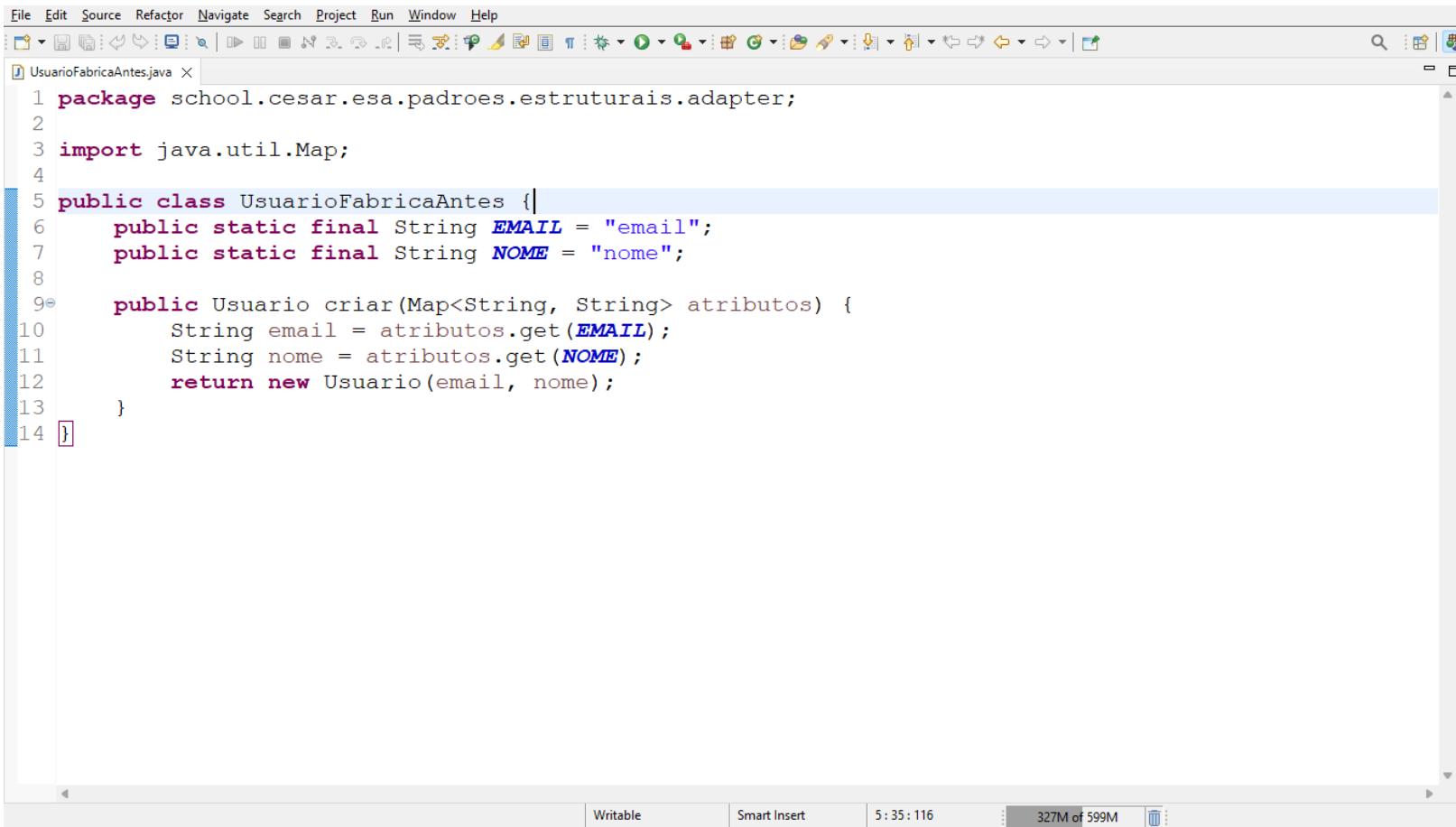


The screenshot shows a Java code editor with the file `Usuario.java` open. The code defines a class `Usuario` with private fields `email` and `nome`, a constructor, and methods for setting and getting these fields. Each setter includes a validation step using a `Validador` object.

```
File Edit Source Refactor Navigate Search Project Run Window Help
User.java X
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 public class Usuario {
4     private String email;
5     private String nome;
6
7     public Usuario(String email, String nome) {
8         setEmail(email);
9         setNome(nome);
10    }
11
12    public void setEmail(String email) {
13        new Validador().validarNaoNulo(email, "O email não pode ser nulo");
14        this.email = email;
15    }
16
17    public String getEmail() {
18        return email;
19    }
20
21    public void setNome(String nome) {
22        new Validador().validarNaoNulo(nome, "O nome não pode ser nulo");
23        this.nome = nome;
24    }
25
26    public String getNome() {
27        return nome;

```

Adapter



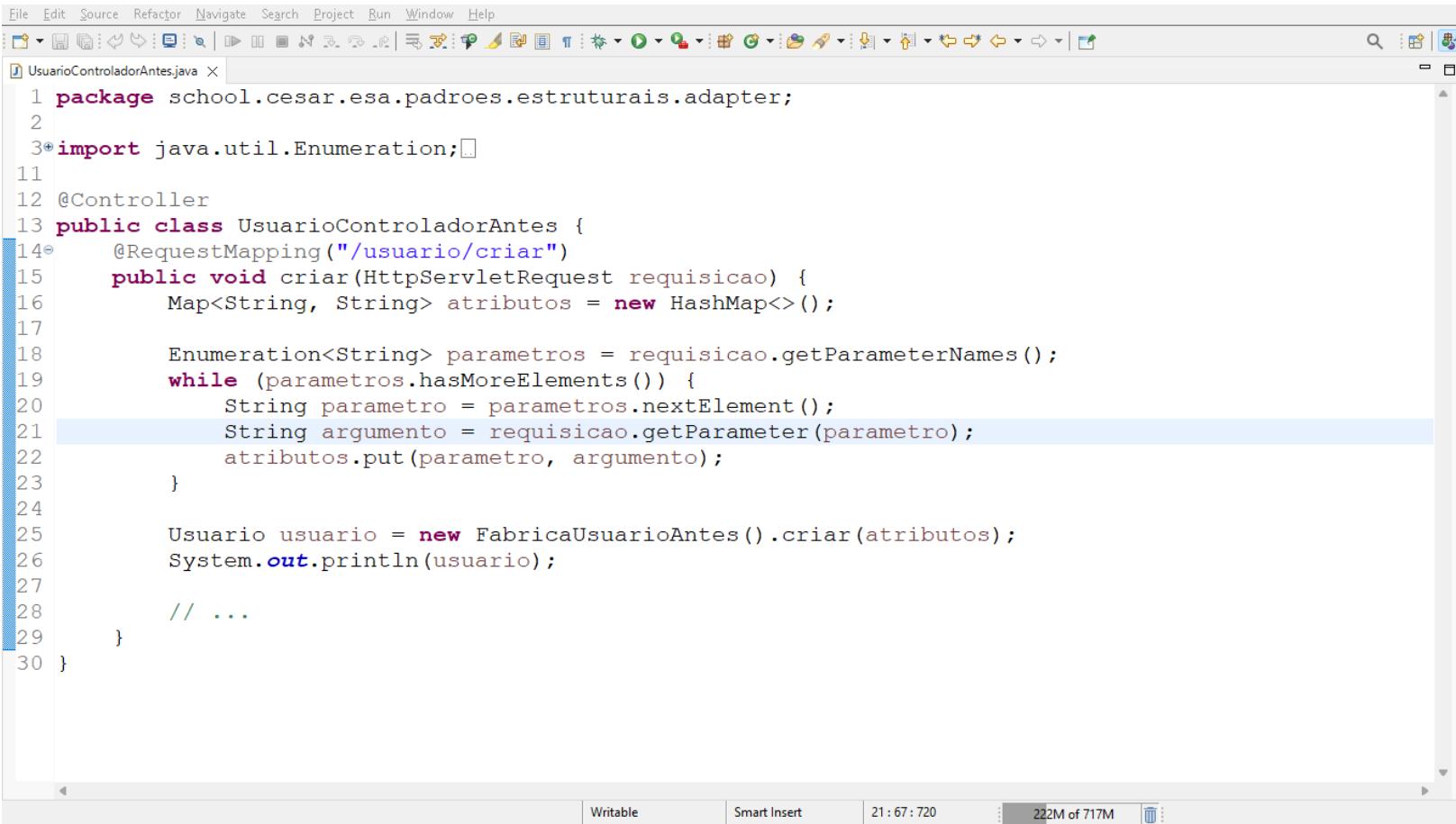
The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes various icons for file operations like Open, Save, Find, Copy, Paste, and others.
- Code Area:** The file `UsuarioFabricaAntes.java` is open. The code defines a class `UsuarioFabricaAntes` with static final fields `EMAIL` and `NOME`, and a method `criar` that takes a `Map<String, String>` and returns a new `Usuario` object.

```
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 import java.util.Map;
4
5 public class UsuarioFabricaAntes {
6     public static final String EMAIL = "email";
7     public static final String NOME = "nome";
8
9     public Usuario criar(Map<String, String> atributos) {
10        String email = atributos.get(EMAIL);
11        String nome = atributos.get(NOME);
12        return new Usuario(email, nome);
13    }
14 }
```

- Status Bar:** Shows Writable, Smart Insert, 5:35:116, 327M of 599M, and a trash bin icon.

Adapter

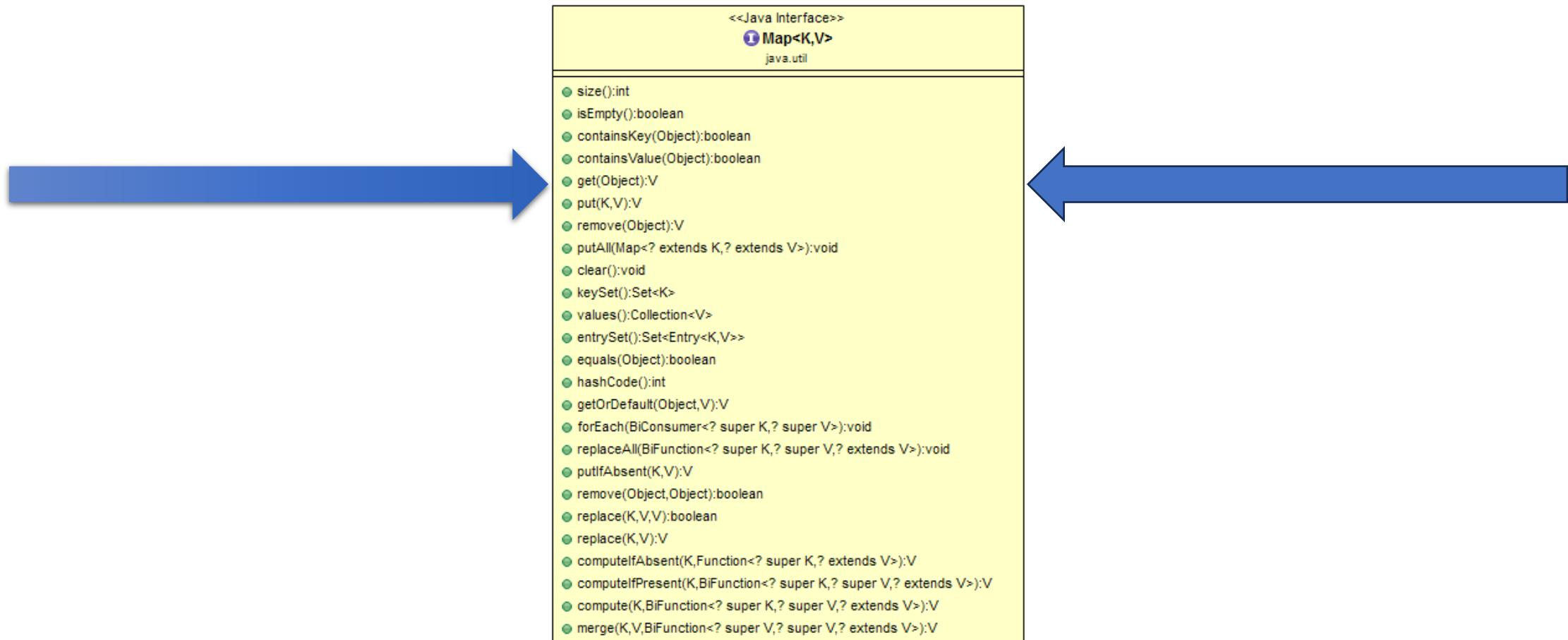


The screenshot shows a Java code editor with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
UserControladorAntes.java X
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 import java.util.Enumeration;
4
5 @Controller
6 public class UsuarioControladorAntes {
7     @RequestMapping("/usuario/criar")
8     public void criar(HttpServletRequest requisicao) {
9         Map<String, String> atributos = new HashMap<>();
10
11         Enumeration<String> parametros = requisicao.getParameterNames();
12         while (parametros.hasMoreElements()) {
13             String parametro = parametros.nextElement();
14             String argumento = requisicao.getParameter(parametro);
15             atributos.put(parametro, argumento);
16         }
17
18         Usuario usuario = new FabricaUsuarioAntes().criar(atributos);
19         System.out.println(usuario);
20
21         // ...
22     }
23
24 }
25
26 }
```

The code implements the Adapter pattern by reading parameters from an HttpServletRequest and creating a Usuario object using a FabricaUsuarioAntes factory.

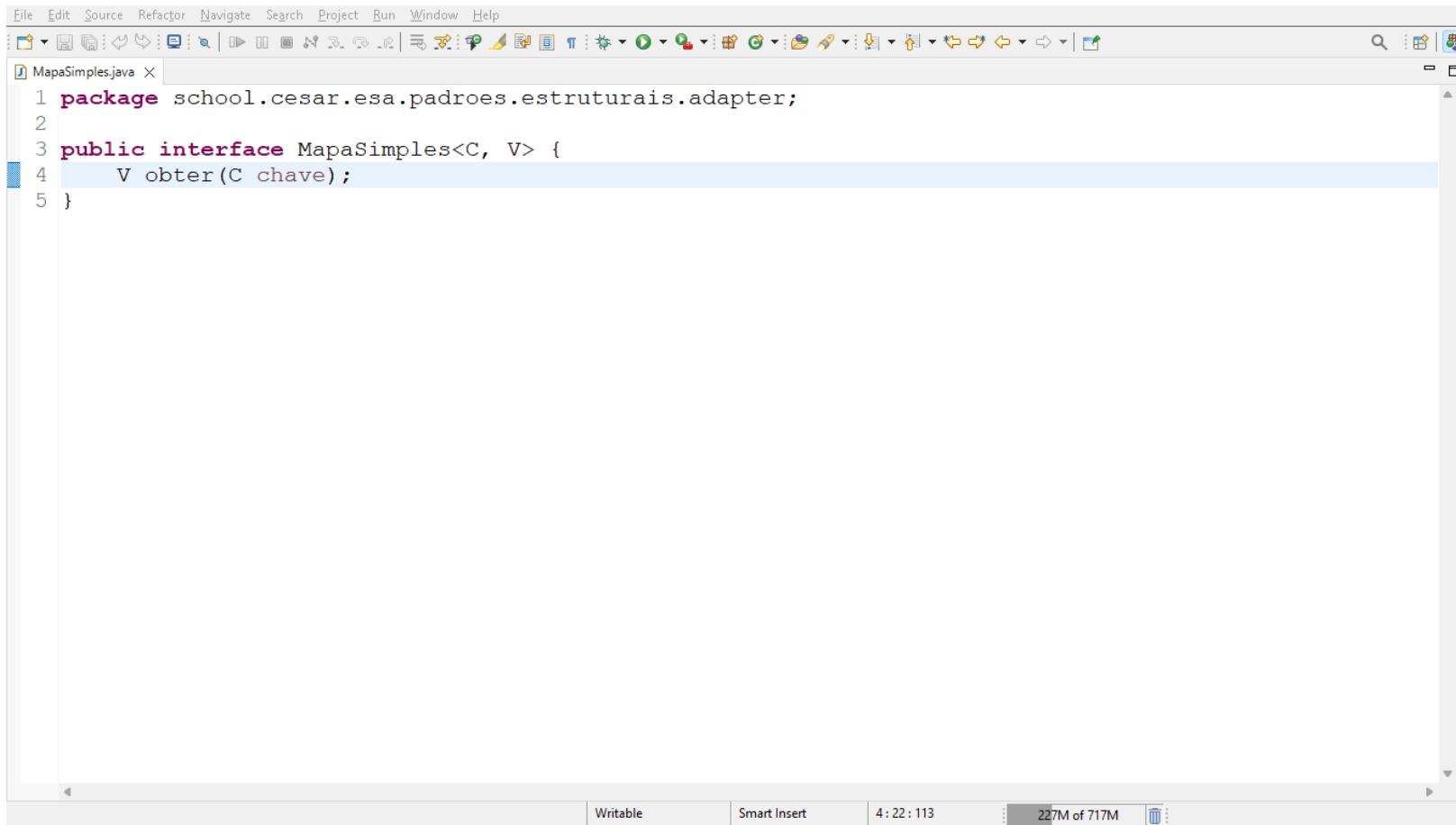
Adapter



Adaptadores



Adapter



A screenshot of a Java IDE interface, likely Eclipse or IntelliJ IDEA, showing a single file named "MapaSimple.java". The code defines a public interface with one method, "obter".

```
File Edit Source Refactor Navigate Search Project Run Window Help  
MapaSimple.java X  
1 package school.cesar.esa.padroes.estruturais.adapter;  
2  
3 public interface MapaSimple<C, V> {  
4     V obter(C chave);  
5 }
```

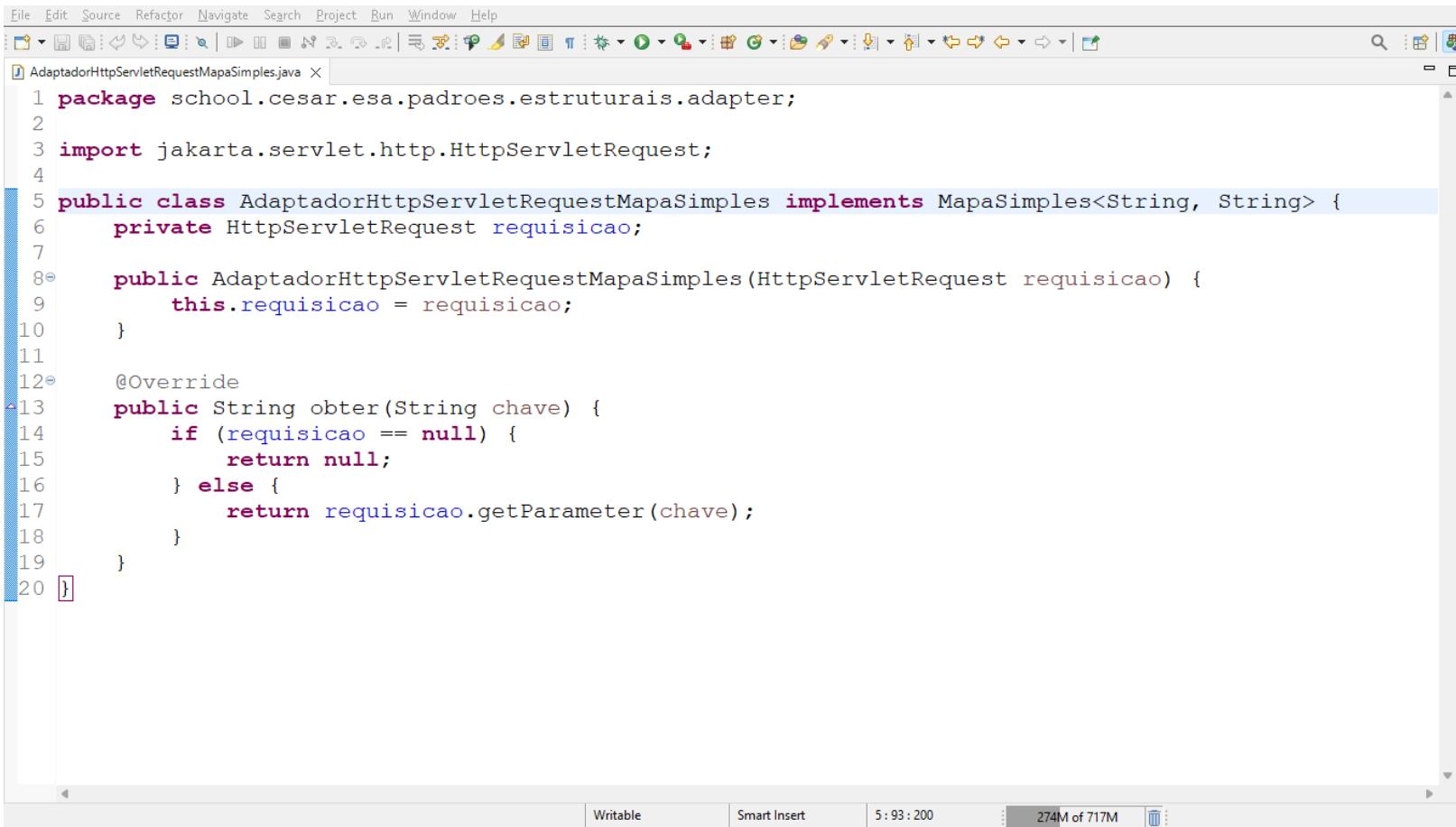
The interface is defined with a generic type parameter <C, V>. The method "obter" takes a parameter of type C and returns a value of type V.

Adapter

The screenshot shows a Java code editor with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
UsuarioFabricaDepois.java ×
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 public class UsuarioFabricaDepois {
4     public static final String EMAIL = "email";
5     public static final String NOME = "nome";
6
7     public Usuario criar(MapaSimples<String, String> atributos) {
8         String email = atributos.obter(EMAIL);
9         String nome = atributos.obter(NOME);
10        return new Usuario(email, nome);
11    }
12 }
```

Adapter

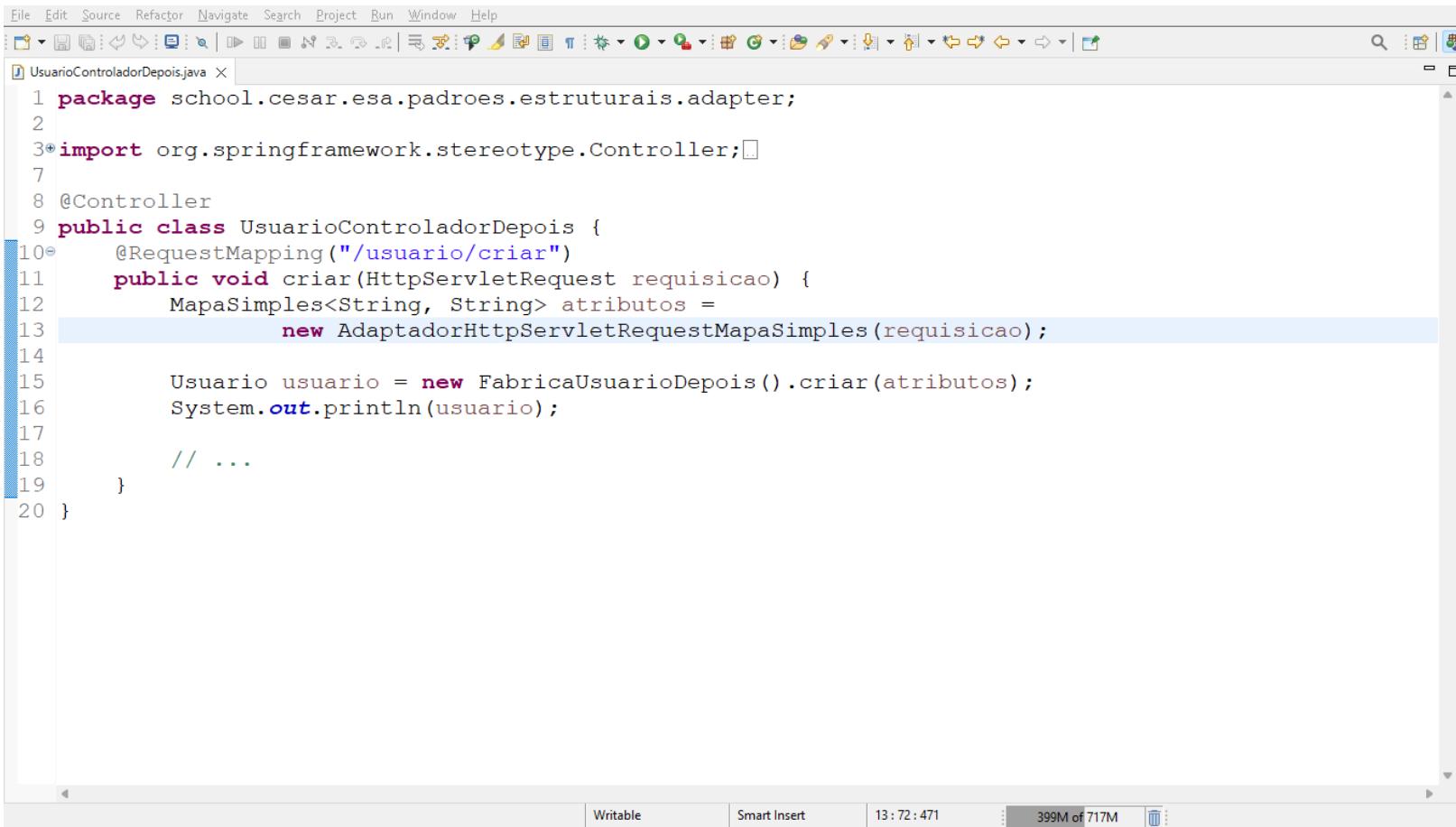


The screenshot shows a Java code editor with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
AdaptadorHttpServletRequestMapaSimples.java
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 import jakarta.servlet.http.HttpServletRequest;
4
5 public class AdaptadorHttpServletRequestMapaSimples implements MapaSimples<String, String> {
6     private HttpServletRequest requisicao;
7
8     public AdaptadorHttpServletRequestMapaSimples(HttpServletRequest requisicao) {
9         this.requisicao = requisicao;
10    }
11
12    @Override
13    public String obter(String chave) {
14        if (requisicao == null) {
15            return null;
16        } else {
17            return requisicao.getParameter(chave);
18        }
19    }
20 }
```

The code implements the `MapaSimples` interface and uses `HttpServletRequest` to map parameters from a request to a simple map.

Adapter



The screenshot shows a Java code editor with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
UserControladorDepois.java X
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller
6 public class UsuarioControladorDepois {
7     @RequestMapping("/usuario/criar")
8     public void criar(HttpServletRequest requisicao) {
9         MapaSimples<String, String> atributos =
10        new AdaptadorHttpServletRequestMapaSimples(requisicao);
11
12        Usuario usuario = new FabricaUsuarioDepois().criar(atributos);
13        System.out.println(usuario);
14
15        // ...
16    }
17
18 }
```

The code implements the Adapter pattern. It defines a controller class `UsuarioControladorDepois` that handles requests for creating a user. It uses a `MapaSimples` to store attributes and adapts an `HttpServletRequest` object using the `AdaptadorHttpServletRequestMapaSimples` adapter. The `FabricaUsuarioDepois` factory is used to create the `Usuario` object.

Adapter

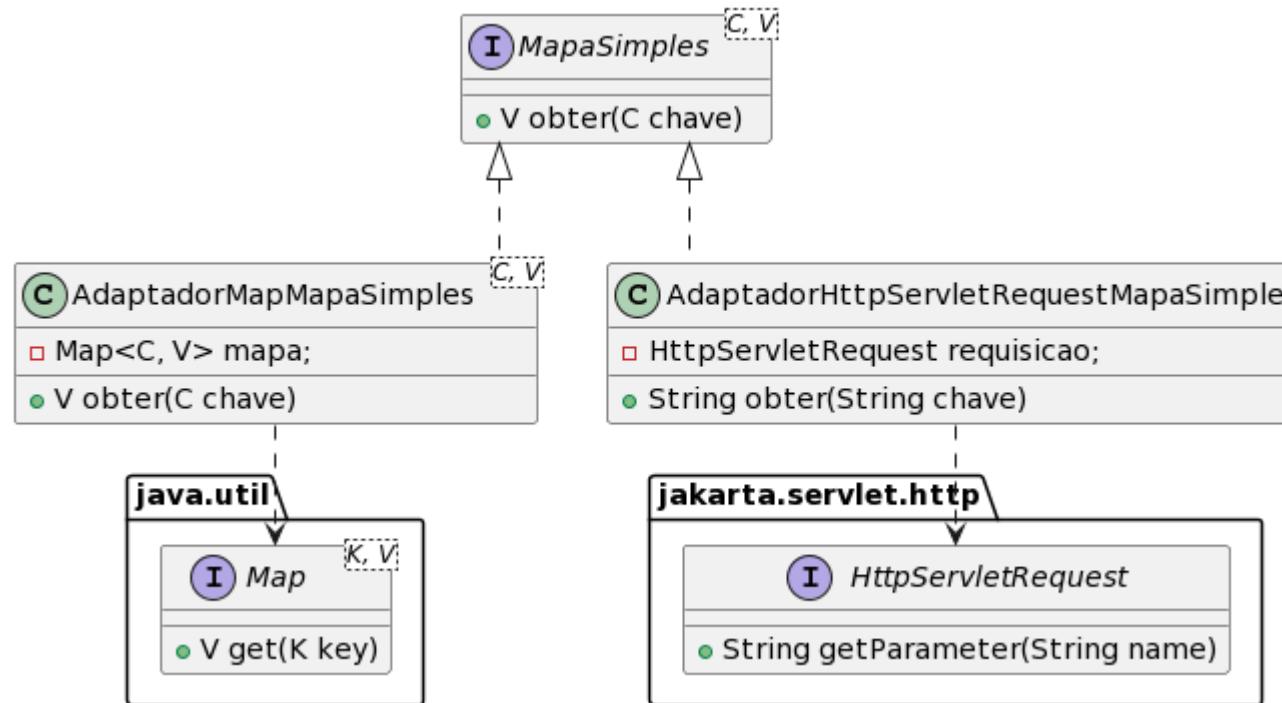
The screenshot shows a Java IDE interface with the following details:

- Toolbar:** Standard Java IDE toolbar with icons for file operations, search, and navigation.
- Project Explorer:** Shows two files: `UsuarioControladorDepois.java` and `AdaptadorMapMapaSimples.java`.
- Code Editor:** Displays the source code for `AdaptadorMapMapaSimples.java`. The code implements the `MapaSimples` interface and adapts a `Map` object.

```
File Edit Source Refactor Navigate Search Project Run Window Help
UserControladorDepois.java AdaptadorMapMapaSimples.java
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 import java.util.Map;
4
5 public class AdaptadorMapMapaSimples<C, V> implements MapaSimples<C, V> {
6     private Map<C, V> mapa;
7
8     public AdaptadorMapMapaSimples(Map<C, V> mapa) {
9         this.mapa = mapa;
10    }
11
12    @Override
13    public V obter(C chave) {
14        if (mapa == null) {
15            return null;
16        } else {
17            return mapa.get(chave);
18        }
19    }
20 }
```

- Status Bar:** Shows the following information: Writable, Smart Insert, 5:74:155, 243M of 717M.

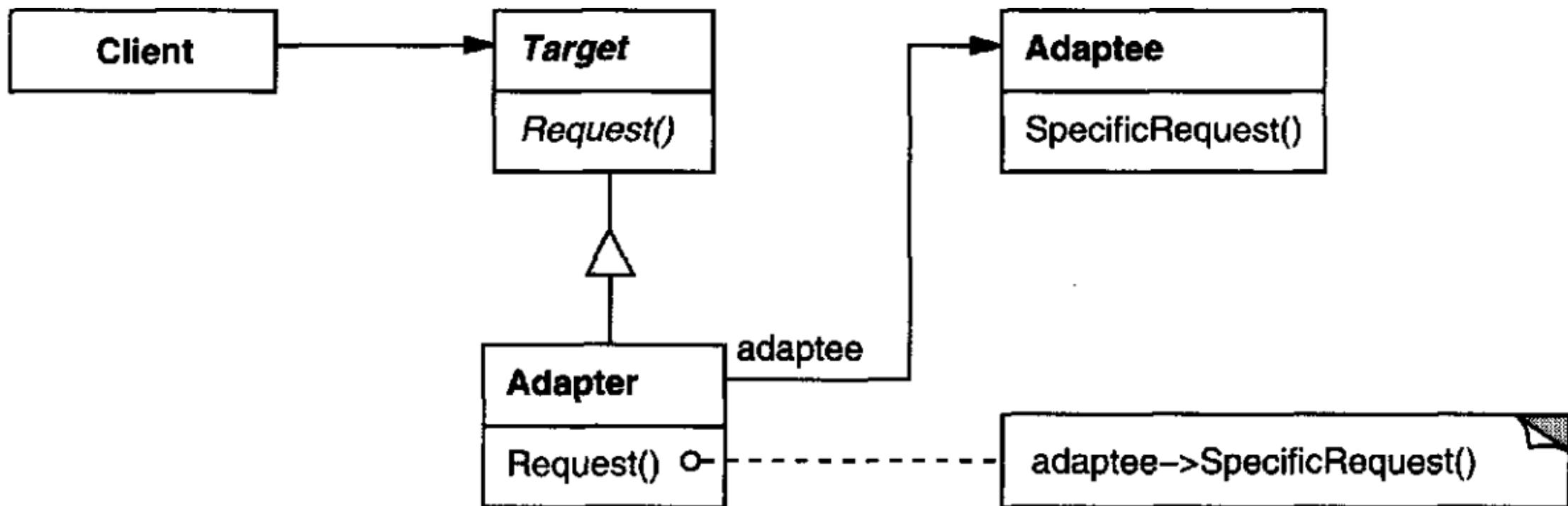
Adapter



Adapter

- “Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.”

Adapter



Adapter

- SOLID
 - Responsabilidade única (**S**ingle responsibility)
 - Aberto-fechado (**O**pen-closed)
 - Substituição de Liskov (**L**iskob substitution)
 - Segregação de interfaces (**I**nterface segregation)
 - Inversão de dependências (**D**evelopment dependency inversion)
- Prefira composição à herança
- Demeter

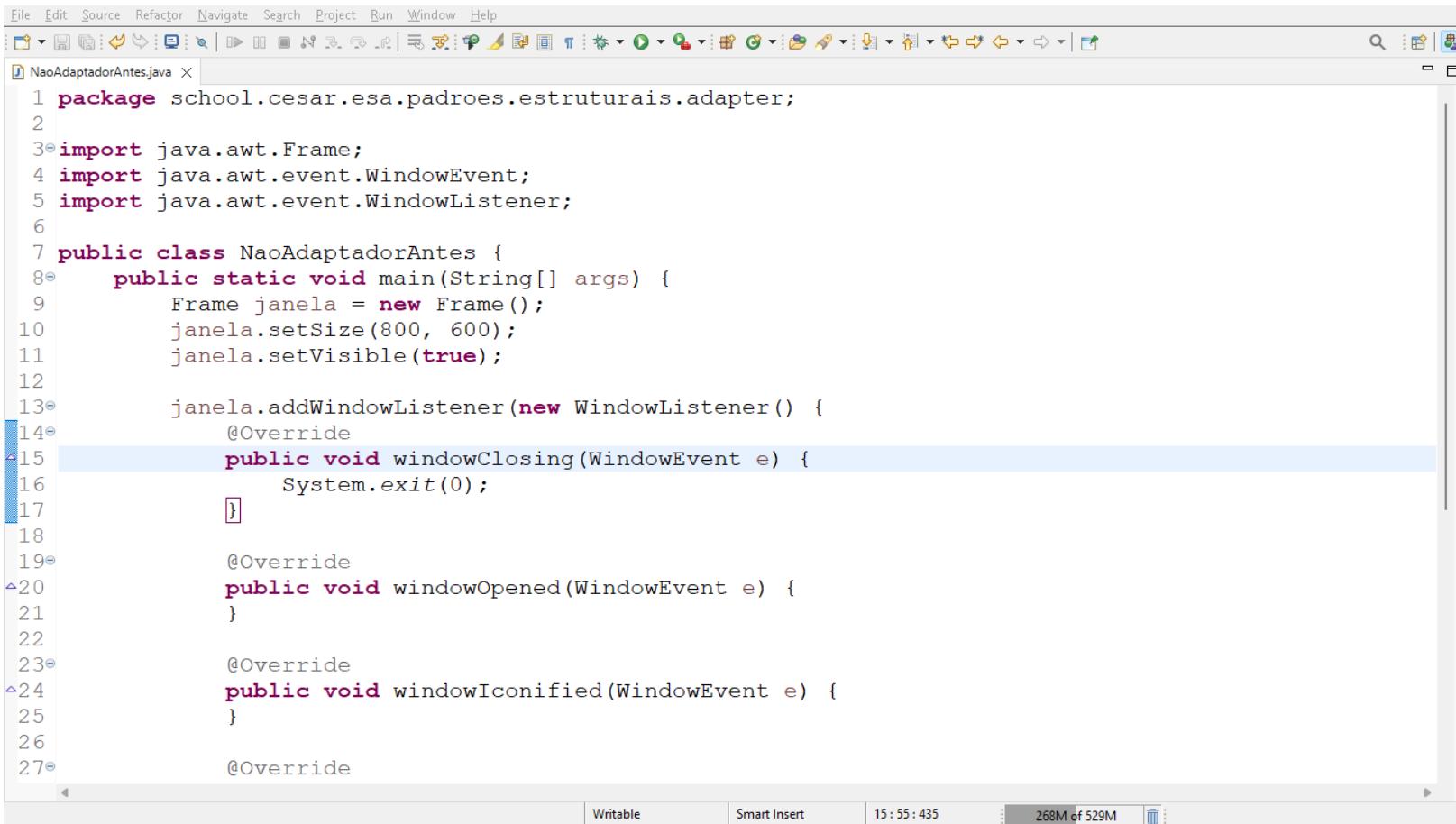
Adapter

- Integridade conceitual
- (Alta) Coesão
- (Baixo) Acoplamento
- Ocultamento de informações

Adapter



Adapter

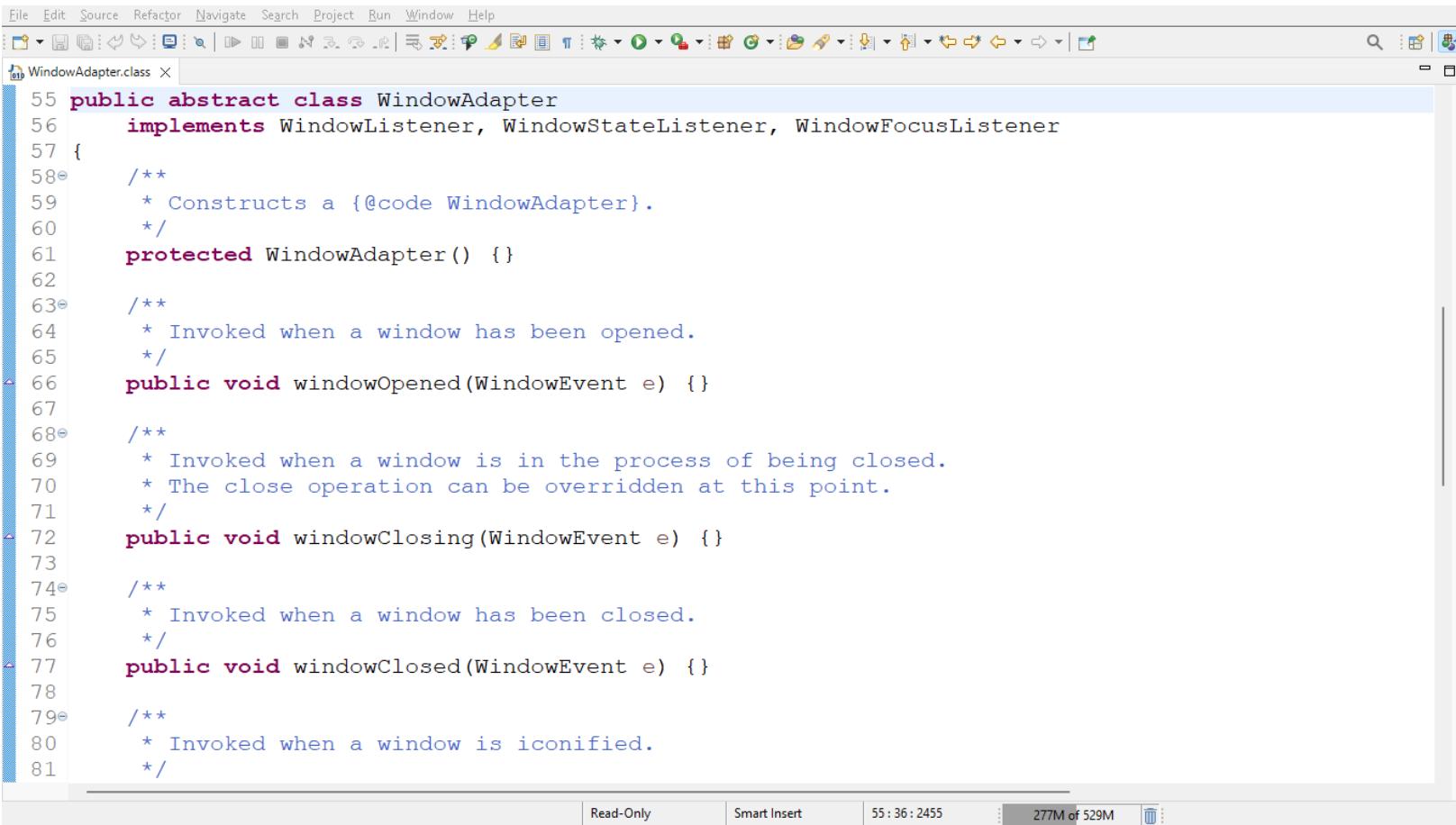


The screenshot shows a Java code editor window with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Code Editor:** The file `NaoAdaptadorAntes.java` is open. The code implements the `WindowListener` interface for a `Frame`.
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 import java.awt.Frame;
4 import java.awt.event.WindowEvent;
5 import java.awt.event.WindowListener;
6
7 public class NaoAdaptadorAntes {
8     public static void main(String[] args) {
9         Frame janela = new Frame();
10        janela.setSize(800, 600);
11        janela.setVisible(true);
12
13        janela.addWindowListener(new WindowListener() {
14            @Override
15            public void windowClosing(WindowEvent e) {
16                System.exit(0);
17            }
18
19            @Override
20            public void windowOpened(WindowEvent e) {
21            }
22
23            @Override
24            public void windowIconified(WindowEvent e) {
25            }
26
27            @Override
```
- Status Bar:** Writable, Smart Insert, 15:55:435, 268M of 529M.

Adapter

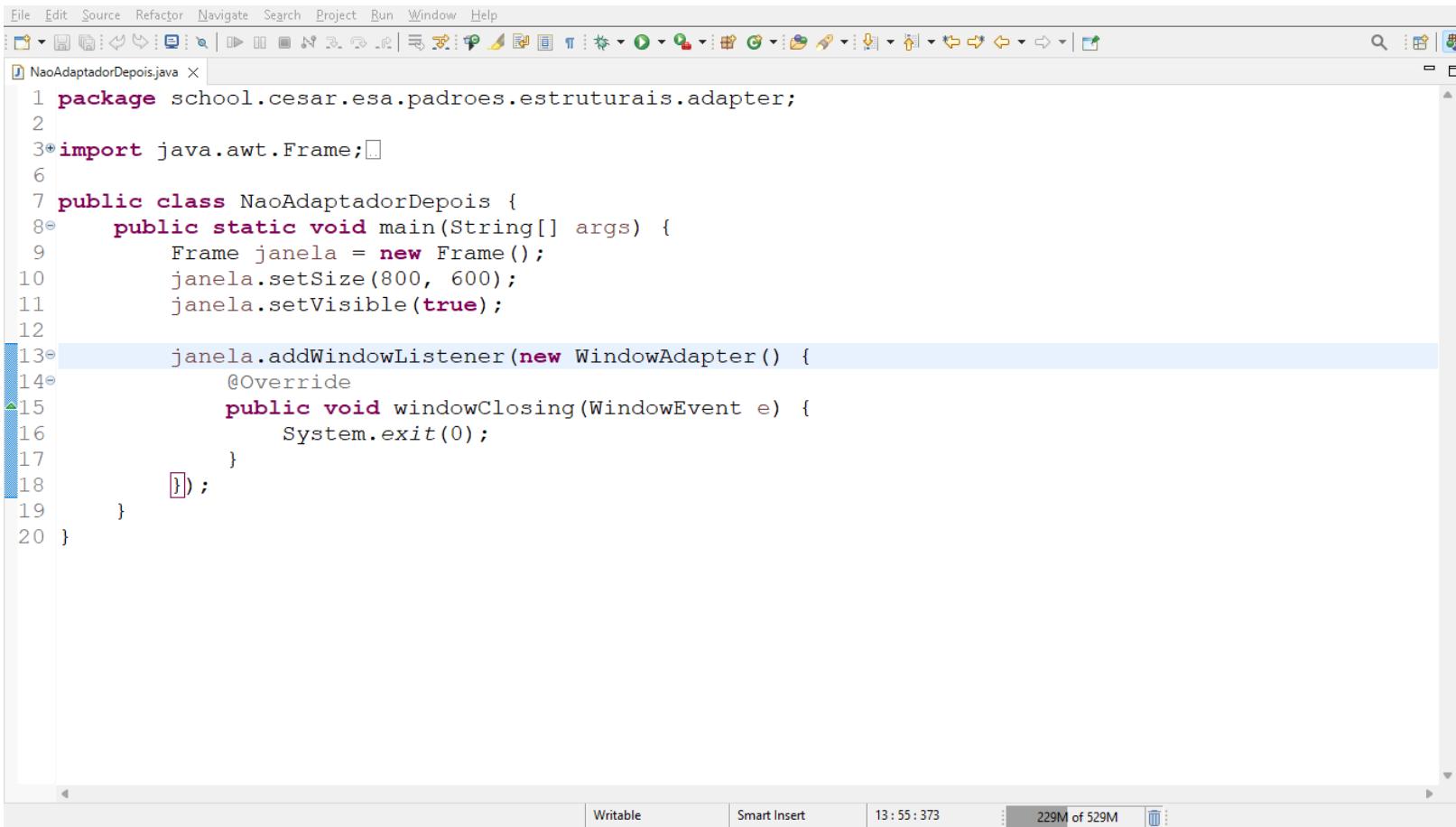


The screenshot shows a Java code editor window with the title bar "File Edit Source Refactor Navigate Search Project Run Window Help". The tab bar shows "WindowAdapter.java X". The code is as follows:

```
55 public abstract class WindowAdapter
56     implements WindowListener, WindowStateListener, WindowFocusListener
57 {
58     /**
59      * Constructs a {@code WindowAdapter}.
60      */
61     protected WindowAdapter() {}
62
63     /**
64      * Invoked when a window has been opened.
65      */
66     public void windowOpened(WindowEvent e) {}
67
68     /**
69      * Invoked when a window is in the process of being closed.
70      * The close operation can be overridden at this point.
71      */
72     public void windowClosing(WindowEvent e) {}
73
74     /**
75      * Invoked when a window has been closed.
76      */
77     public void windowClosed(WindowEvent e) {}
78
79     /**
80      * Invoked when a window is iconified.
81      */
```

The status bar at the bottom shows "Read-Only", "Smart Insert", "55 : 36 : 2455", "277M of 529M", and a trash bin icon.

Adapter



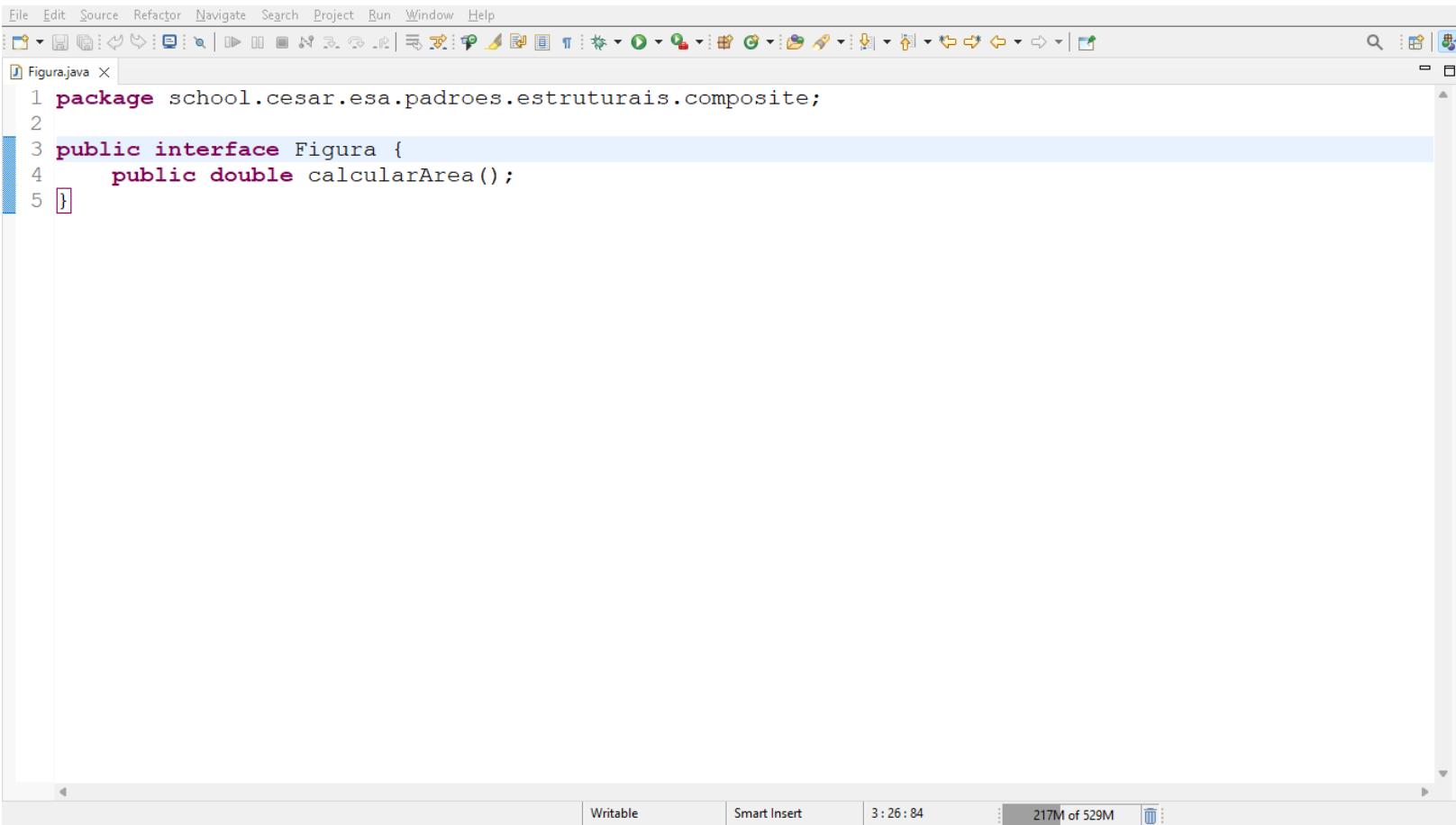
The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Code Editor:** The file `NaoAdaptadorDepois.java` is open. The code implements the Adapter pattern to handle window closing events.
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.adapter;
2
3 import java.awt.Frame;
4
5 public class NaoAdaptadorDepois {
6     public static void main(String[] args) {
7         Frame janela = new Frame();
8         janela.setSize(800, 600);
9         janela.setVisible(true);
10
11         janela.addWindowListener(new WindowAdapter() {
12             @Override
13             public void windowClosing(WindowEvent e) {
14                 System.exit(0);
15             }
16         });
17     }
18 }
19 }
```
- Status Bar:** Writable Smart Insert 13 : 55 : 373 229M of 529M

Composite

Composite

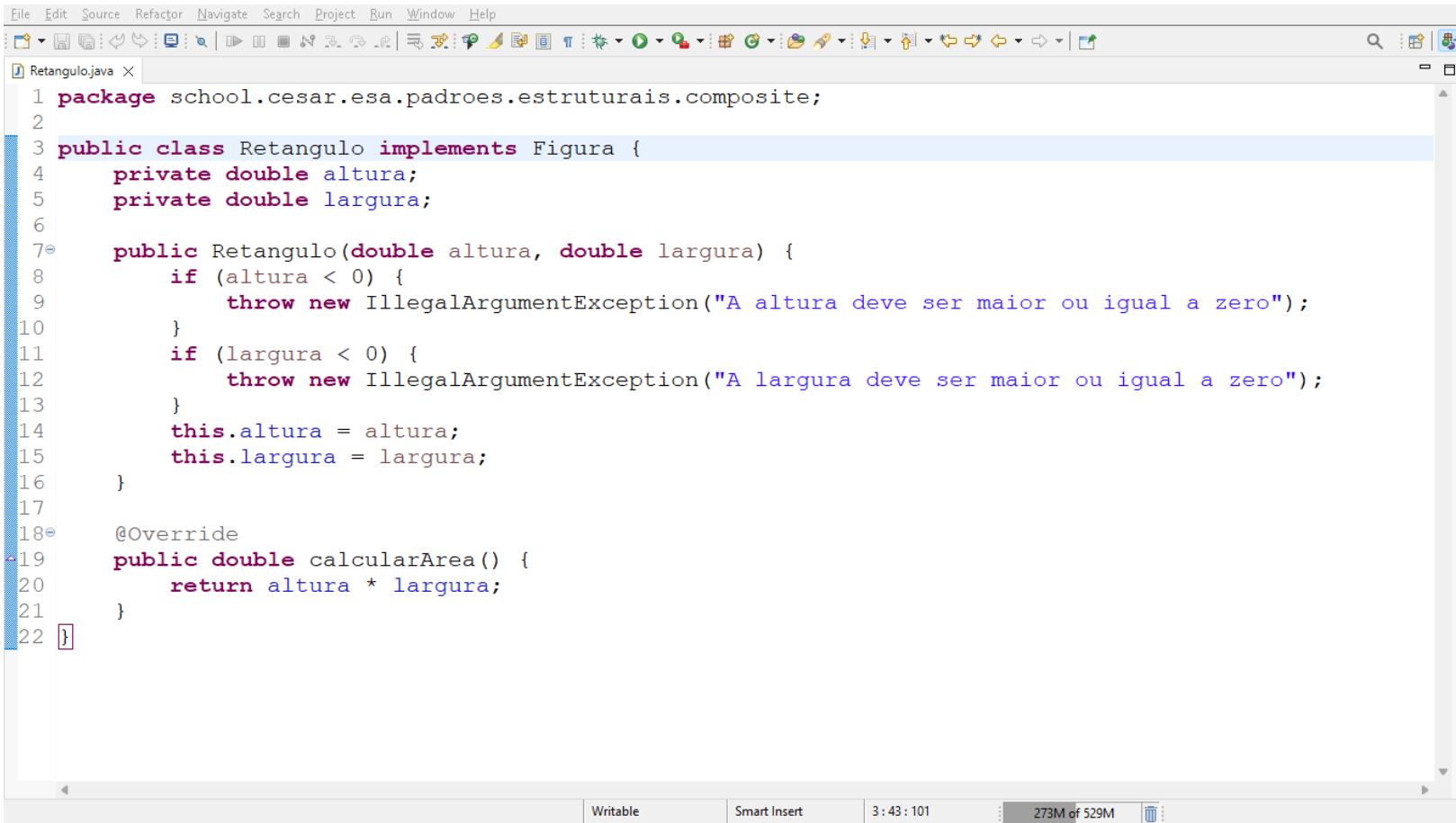


A screenshot of a Java Integrated Development Environment (IDE) showing a single file named `Figura.java`. The code defines a public interface `Figura` with a single method `calcularArea()`. The interface is located in the package `school.cesar.esa.padroes.estruturais.composite`.

```
File Edit Source Refactor Navigate Search Project Run Window Help
Figura.java X
1 package school.cesar.esa.padroes.estruturais.composite;
2
3 public interface Figura {
4     public double calcularArea();
5 }
```

The IDE interface includes a toolbar with various icons, a menu bar with options like File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help, and a status bar at the bottom showing the file path, file type (Writable), insertion point (Smart Insert), current time (3:26:84), and disk usage (217M of 529M).

Composite

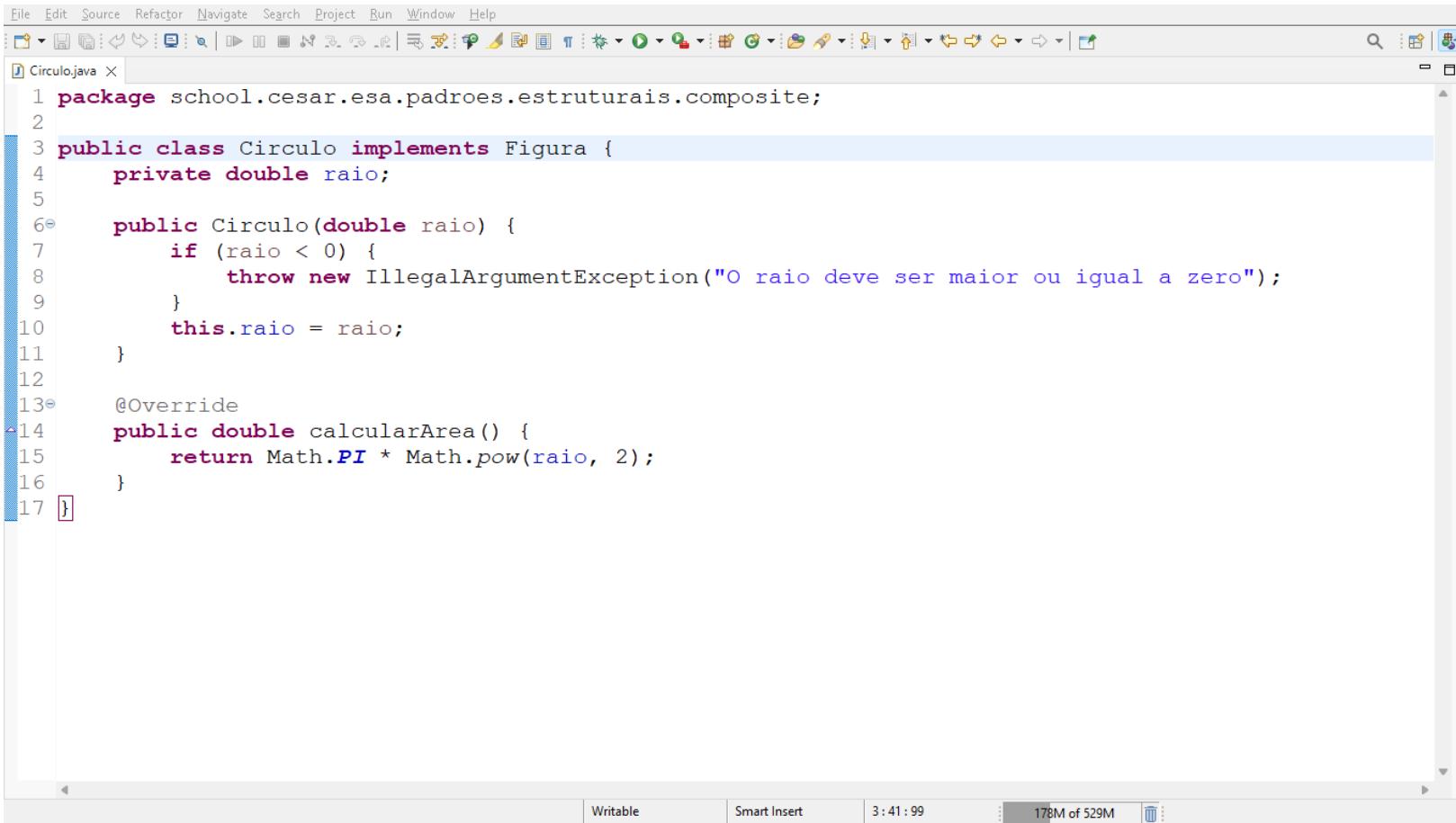


The screenshot shows a Java code editor with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
Retangulo.java ×
1 package school.cesar.esa.padroes.estruturais.composite;
2
3 public class Retangulo implements Figura {
4     private double altura;
5     private double largura;
6
7     public Retangulo(double altura, double largura) {
8         if (altura < 0) {
9             throw new IllegalArgumentException("A altura deve ser maior ou igual a zero");
10        }
11        if (largura < 0) {
12            throw new IllegalArgumentException("A largura deve ser maior ou igual a zero");
13        }
14        this.altura = altura;
15        this.largura = largura;
16    }
17
18    @Override
19    public double calcularArea() {
20        return altura * largura;
21    }
22 }
```

The code defines a class `Retangulo` that implements the `Figura` interface. It has private fields `altura` and `largura`. The constructor checks if both dimensions are non-negative. The `@Override` method `calcularArea()` returns the product of `altura` and `largura`.

Composite



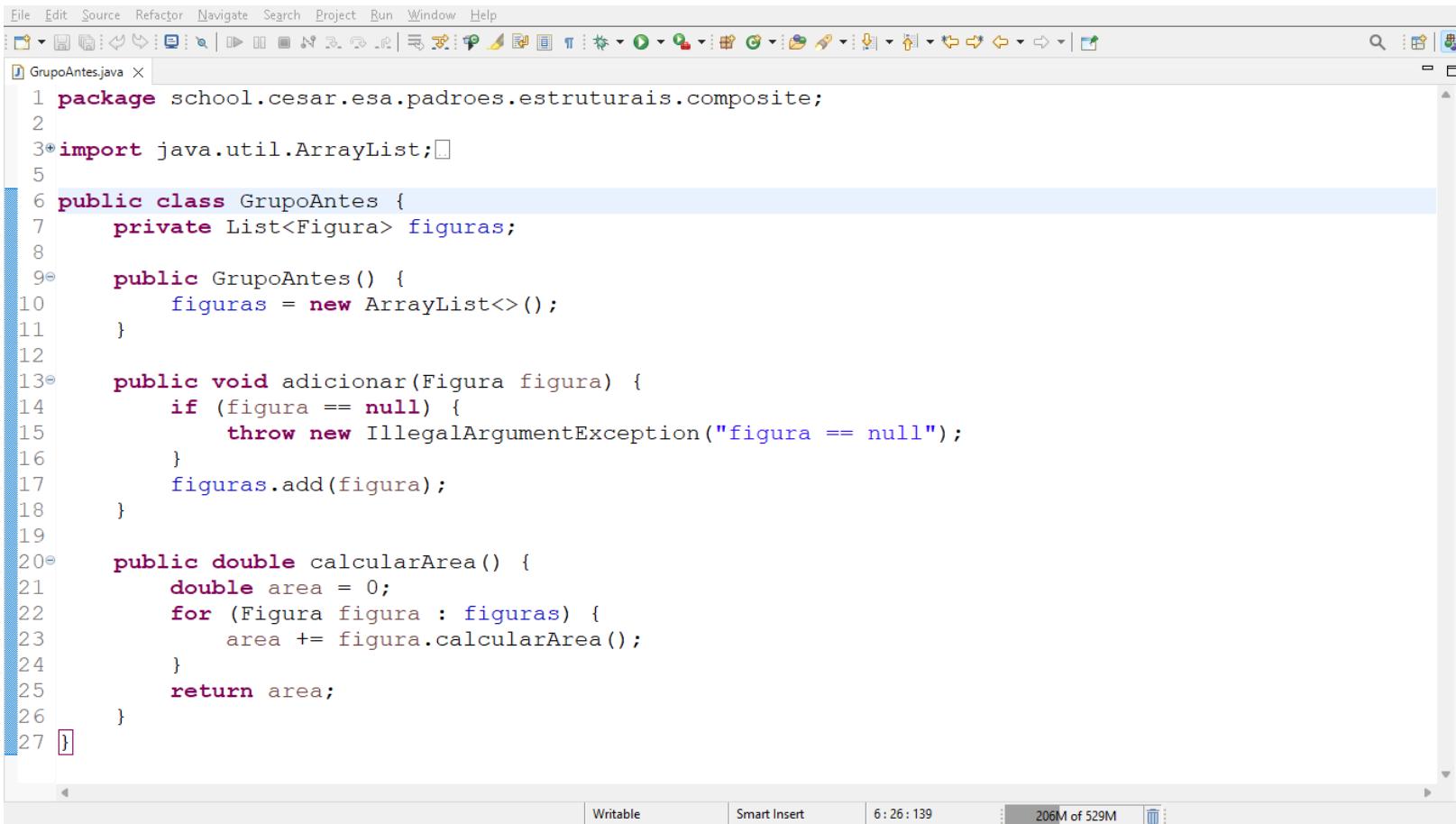
The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes icons for file operations like Open, Save, Print, and various search and navigation functions.
- Code Editor:** The file `Circulo.java` is open. The code implements the `Figura` interface and contains methods for calculating area and perimeter, along with validation logic for the radius.

```
1 package school.cesar.esa.padroes.estruturais.composite;
2
3 public class Circulo implements Figura {
4     private double raio;
5
6     public Circulo(double raio) {
7         if (raio < 0) {
8             throw new IllegalArgumentException("O raio deve ser maior ou igual a zero");
9         }
10        this.raio = raio;
11    }
12
13    @Override
14    public double calcularArea() {
15        return Math.PI * Math.pow(raio, 2);
16    }
17 }
```

- Status Bar:** Shows Writable, Smart Insert, 3:41:99, 178M of 529M, and a trash bin icon.

Composite

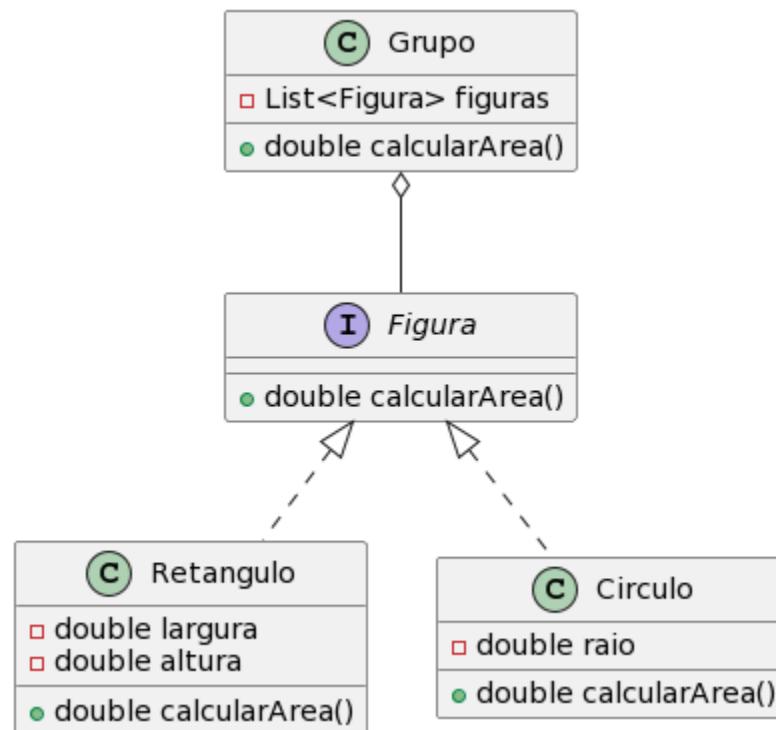


The screenshot shows a Java code editor with the file `GrupoAntes.java` open. The code implements the Composite pattern. It defines a class `GrupoAntes` that contains a list of `Figura` objects. The class has methods to add figures and calculate the total area.

```
File Edit Source Refactor Navigate Search Project Run Window Help
J GrupoAntes.java X
1 package school.cesar.esa.padroes.estruturais.composite;
2
3 import java.util.ArrayList;
4
5 public class GrupoAntes {
6     private List<Figura> figuras;
7
8     public GrupoAntes() {
9         figuras = new ArrayList<>();
10    }
11
12    public void adicionar(Figura figura) {
13        if (figura == null) {
14            throw new IllegalArgumentException("figura == null");
15        }
16        figuras.add(figura);
17    }
18
19    public double calcularArea() {
20        double area = 0;
21        for (Figura figura : figuras) {
22            area += figura.calcularArea();
23        }
24    }
25    return area;
26}
27]
```

The status bar at the bottom of the IDE shows the following information: Writable, Smart Insert, 6:26:139, 206M of 529M.

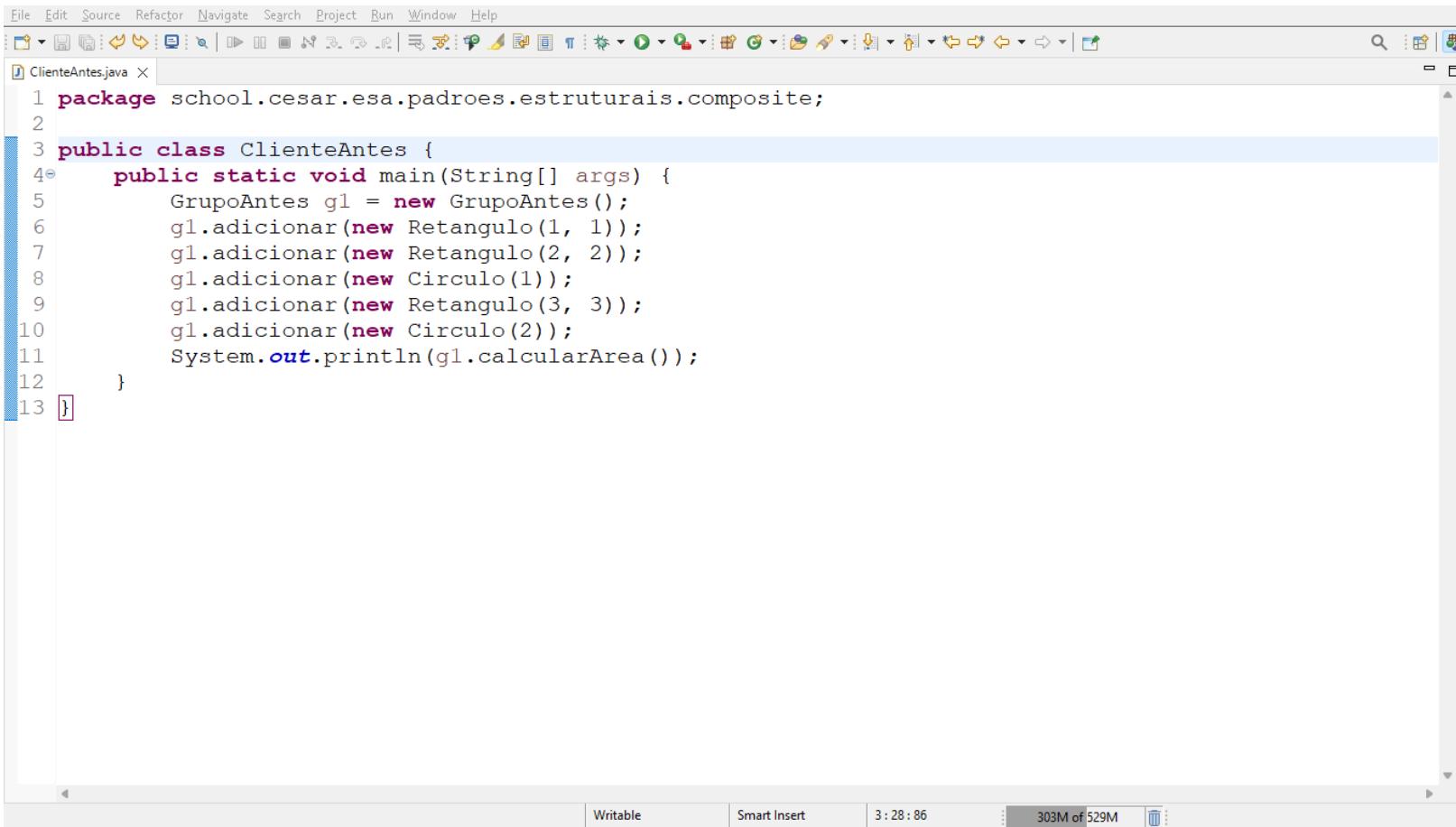
Composite



Composite

- g1
 - r1
 - r2
 - c1
 - r3
 - c2

Composite



The screenshot shows a Java code editor window titled "ClienteAntes.java". The code implements the Composite pattern:

```
File Edit Source Refactor Navigate Search Project Run Window Help
ClienteAntes.java X
1 package school.cesar.esa.padroes.estruturais.composite;
2
3 public class ClienteAntes {
4     public static void main(String[] args) {
5         GrupoAntes gl = new GrupoAntes();
6         gl.adicionar(new Retangulo(1, 1));
7         gl.adicionar(new Retangulo(2, 2));
8         gl.adicionar(new Circulo(1));
9         gl.adicionar(new Retangulo(3, 3));
10        gl.adicionar(new Circulo(2));
11        System.out.println(gl.calcularArea());
12    }
13 }
```

The code defines a `ClienteAntes` class with a `main` method. It creates a `GrupoAntes` object and adds several `Retangulo` and `Circulo` objects to it. Finally, it prints the total area calculated by the `calcularArea` method of the `GrupoAntes` object.

Composite

- g1
 - r1
 - r2
 - c1
 - r3
 - c2
 - *g2*
 - c3
 - c4
 - r5

Composite

The screenshot shows an IDE interface with the following components:

- MenuBar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar.
- Project Explorer:** Shows a single file named "ClienteAntes.java".
- Code Editor:** Displays the following Java code:

```
1 package school.cesar.esa.padroes.estruturais.composite;
2
3 public class ClienteAntes {
4     public static void main(String[] args) {
5         GrupoAntes g2 = new GrupoAntes();
6         g2.adicionar(new Circulo(3));
7         g2.adicionar(new Circulo(4));
8         g2.adicionar(new Retangulo(5, 5));
9
10        GrupoAntes g1 = new GrupoAntes();
11        g1.adicionar(new Retangulo(1, 1));
12        g1.adicionar(new Retangulo(2, 2));
13        g1.adicionar(new Circulo(1));
14        g1.adicionar(new Retangulo(3, 3));
15        g1.adicionar(new Circulo(2));
16        g1.adicionar(g2);
17        System.out.println(g1.calcularArea());
18    }
19}
```

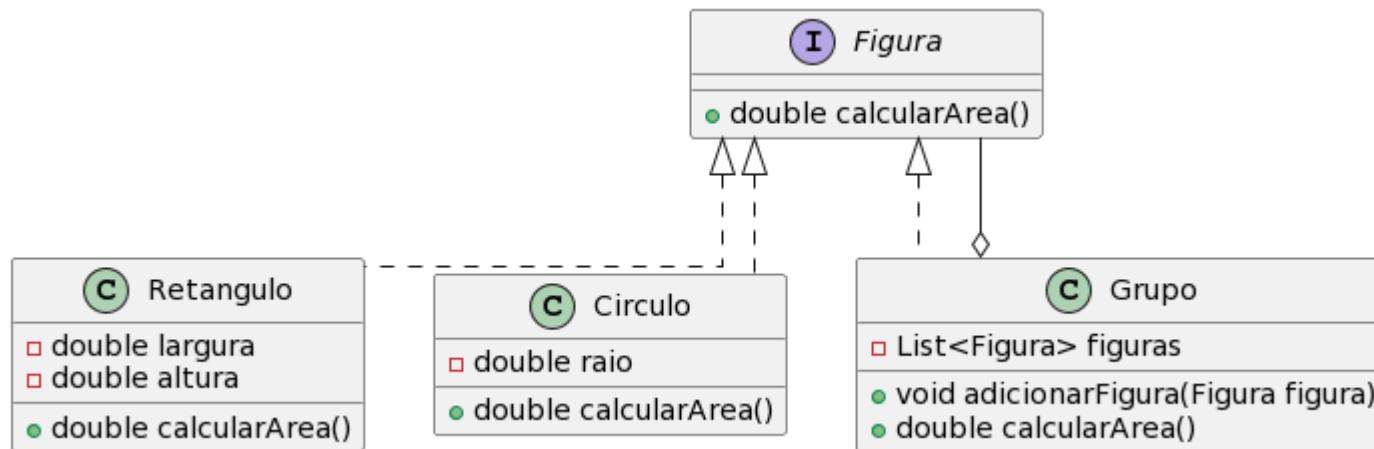
- Problems View:** Shows 1 error and 3 warnings. The error is: "The method adicionar(Figura) in the type GrupoAntes is not applicable for the arguments (GrupoAntes)".
- Properties for Java Problem:** A modal dialog for the selected error. It contains the following details:

Marker	
Severity:	Error
On element:	ClienteAntes.java
Path:	composite/src/main/java/school/cesar/esa/padroes/estruturais/composite
Location:	line 16
Creation time:	17 de março de 2024 12:24:23

Description: The method adicionar(Figura) in the type GrupoAntes is not applicable for the arguments (GrupoAntes)

Buttons at the bottom: ? (Help), Apply and Close (highlighted in blue), and Cancel.

Composite



Composite

The screenshot shows a Java code editor with the following code:

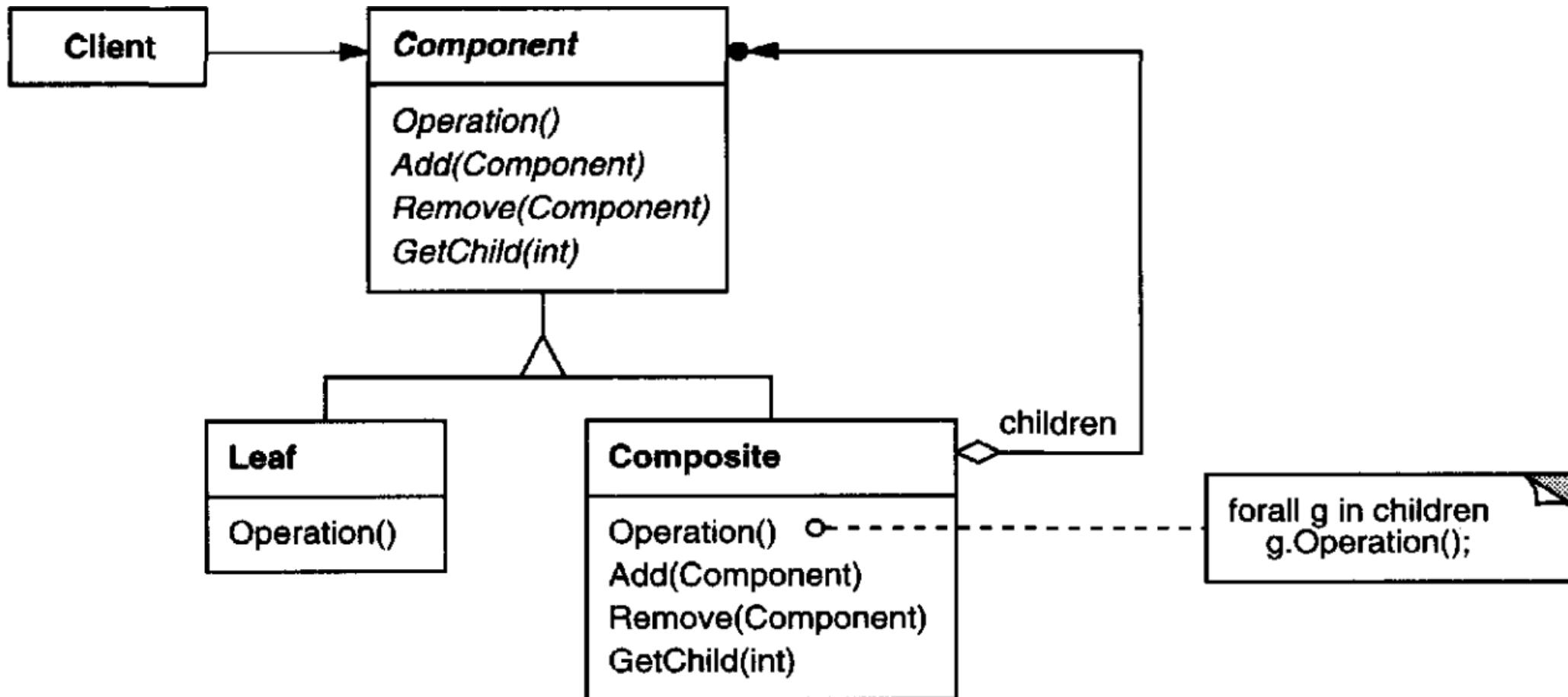
```
File Edit Source Refactor Navigate Search Project Run Window Help
ClientDepois.java X
1 package school.cesar.esa.padroes.estruturais.composite;
2
3 public class ClienteDepois {
4     public static void main(String[] args) {
5         GrupoDepois g2 = new GrupoDepois();
6         g2.adicionar(new Circulo(3));
7         g2.adicionar(new Circulo(4));
8         g2.adicionar(new Retangulo(5, 5));
9
10        GrupoDepois g1 = new GrupoDepois();
11        g1.adicionar(new Retangulo(1, 1));
12        g1.adicionar(new Retangulo(2, 2));
13        g1.adicionar(new Circulo(1));
14        g1.adicionar(new Retangulo(3, 3));
15        g1.adicionar(new Circulo(2));
16        g1.adicionar(g2);
17        System.out.println(g1.calcularArea());
18    }
19 }
```

The code demonstrates the Composite pattern. It defines a `ClienteDepois` class with a `main` method. Inside `main`, it creates two `GrupoDepois` objects, `g1` and `g2`. `g1` contains three `Retangulo` objects and one `Circulo` object. `g2` contains three `Circulo` objects and one `Retangulo` object. Both `g1` and `g2` implement the `calcularArea` method.

Composite

“Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”

Composite



Composite

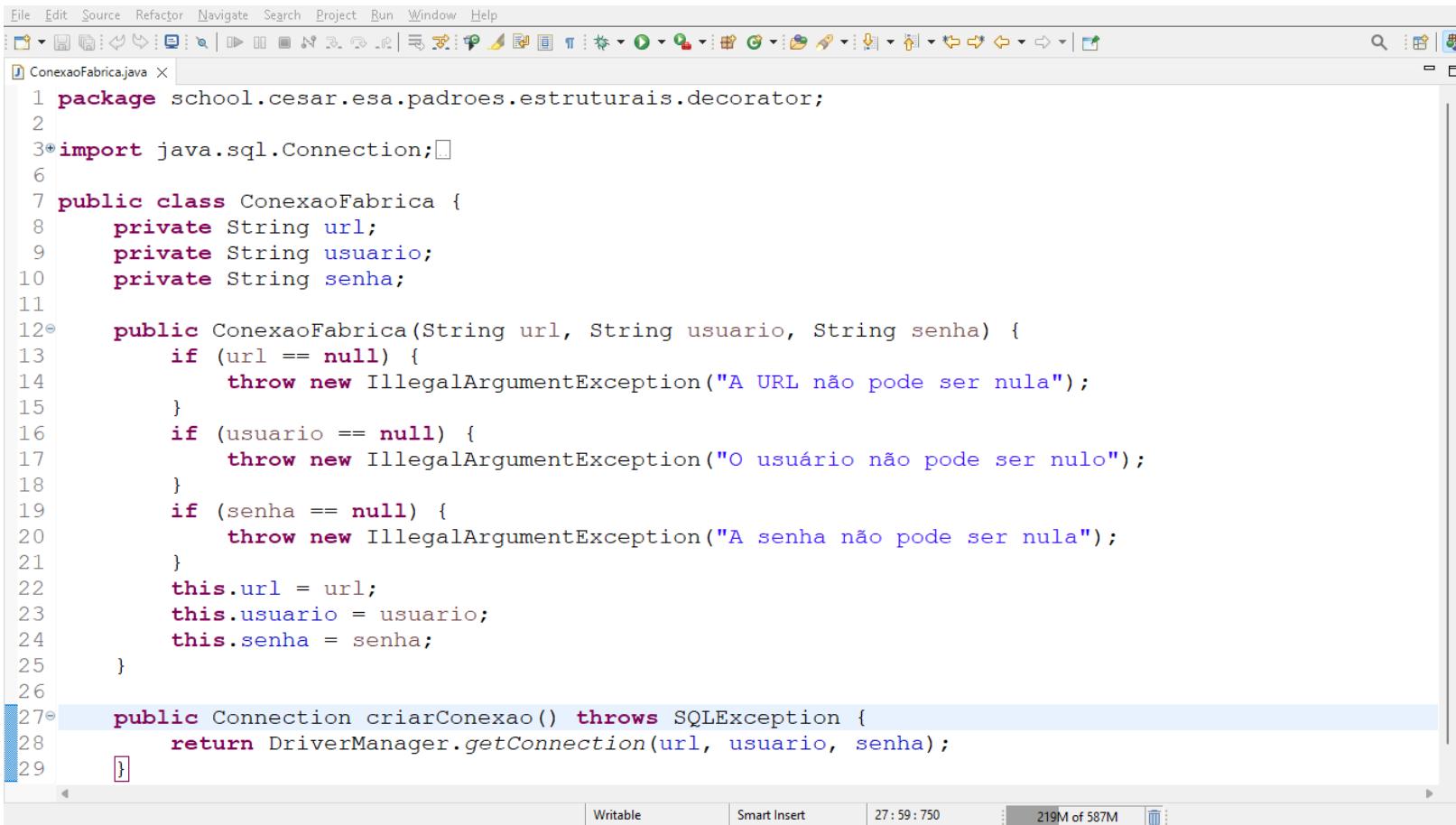
- SOLID
 - Responsabilidade única (**S**ingle responsibility)
 - Aberto-fechado (**O**pen-closed)
 - Substituição de Liskov (**L**iskob substitution)
 - Segregação de interfaces (**I**nterface segregation)
 - Inversão de dependências (**D**evelopment dependency inversion)
- Prefira composição à herança
- Demeter

Composite

- Integridade conceitual
- (Alta) Coesão
- (Baixo) Acoplamento
- Ocultamento de informações

Decorator

Decorator

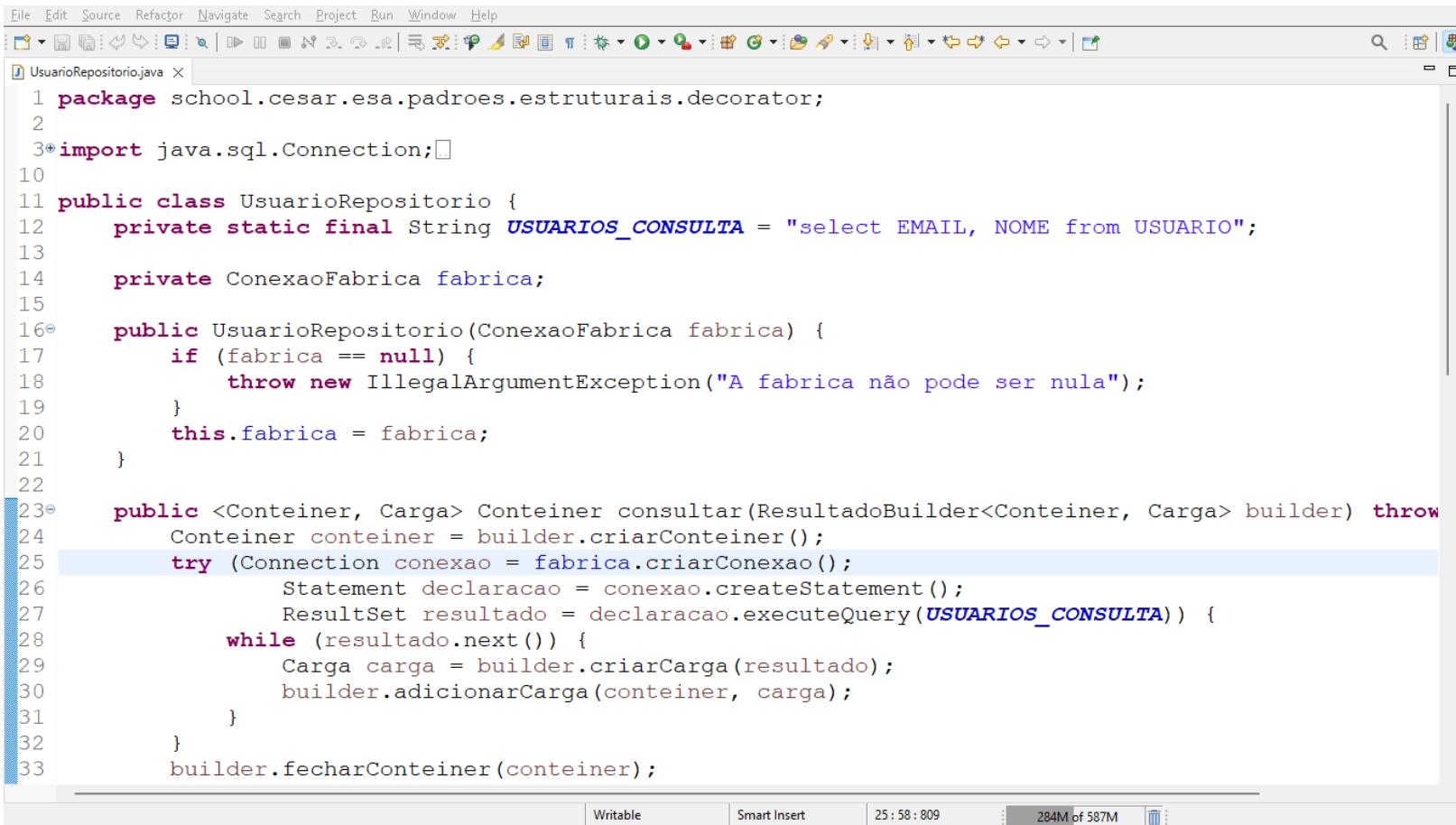


The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes various icons for file operations, search, and navigation.
- Code Editor:** The file is named "ConexaoFabrica.java". The code implements the Decorator pattern for database connections.
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.decorator;
2
3 import java.sql.Connection;
4
5 public class ConexaoFabrica {
6     private String url;
7     private String usuario;
8     private String senha;
9
10    public ConexaoFabrica(String url, String usuario, String senha) {
11        if (url == null) {
12            throw new IllegalArgumentException("A URL não pode ser nula");
13        }
14        if (usuario == null) {
15            throw new IllegalArgumentException("O usuário não pode ser nulo");
16        }
17        if (senha == null) {
18            throw new IllegalArgumentException("A senha não pode ser nula");
19        }
20        this.url = url;
21        this.usuario = usuario;
22        this.senha = senha;
23    }
24
25    public Connection criarConexao() throws SQLException {
26        return DriverManager.getConnection(url, usuario, senha);
27    }
28}
```
- Status Bar:** Writable Smart Insert 27:59:750 219M of 587M

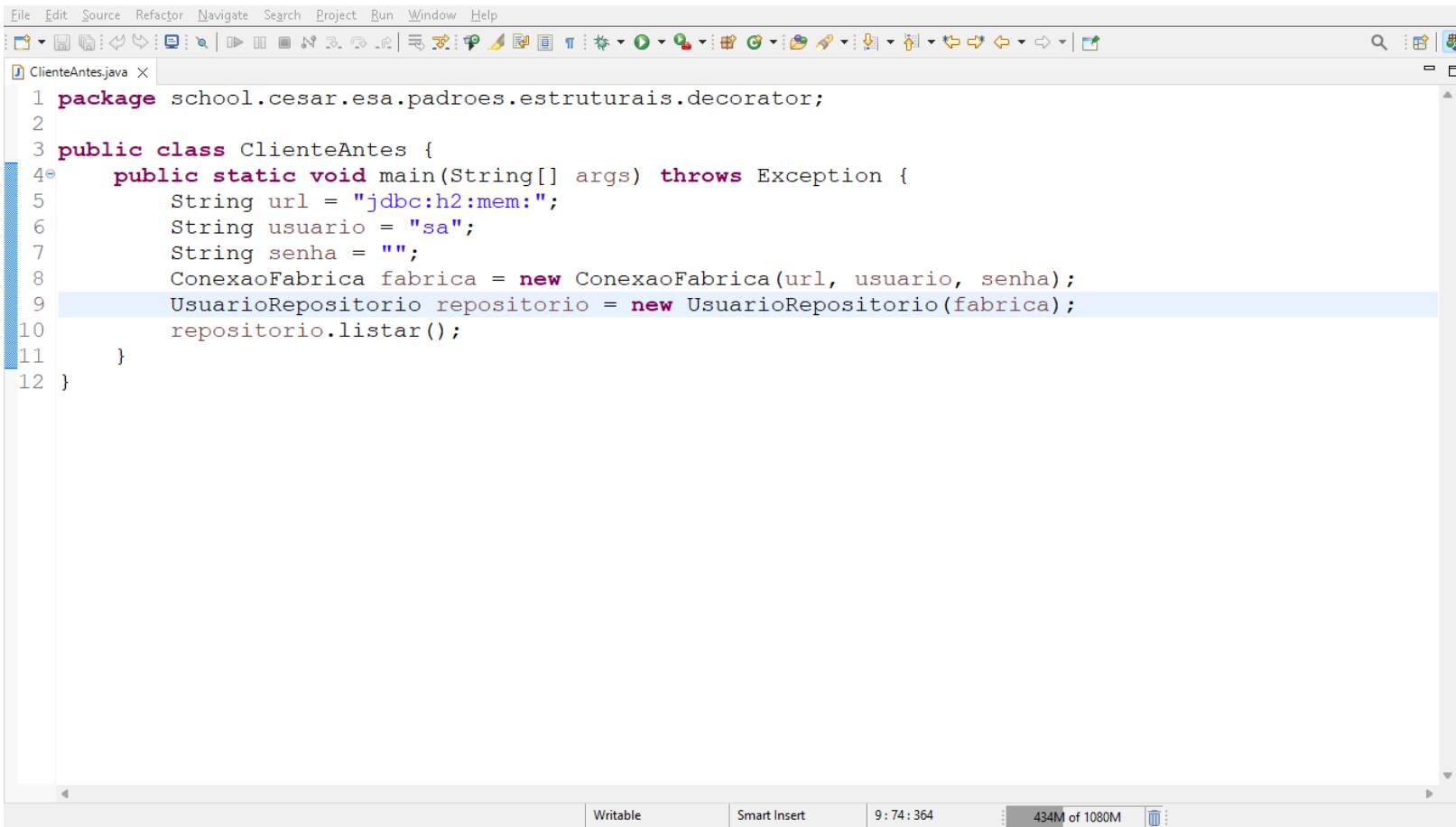
Decorator



The screenshot shows a Java code editor with the file `UsuarioRepositorio.java` open. The code implements the Decorator pattern for managing user data from a database. It includes a constructor that takes a `ConexaoFabrica` object and a method `consultar` that performs a query to retrieve user information.

```
File Edit Source Refactor Navigate Search Project Run Window Help
J UsuarioRepositorio.java X
1 package school.cesar.esa.padroes.estruturais.decorator;
2
3 import java.sql.Connection;
10
11 public class UsuarioRepositorio {
12     private static final String USUARIOS_CONSULTA = "select EMAIL, NOME from USUARIO";
13
14     private ConexaoFabrica fabrica;
15
16     public UsuarioRepositorio(ConexaoFabrica fabrica) {
17         if (fabrica == null) {
18             throw new IllegalArgumentException("A fabrica não pode ser nula");
19         }
20         this.fabrica = fabrica;
21     }
22
23     public <Conteiner, Carga> Conteiner consultar(ResultadoBuilder<Conteiner, Carga> builder) throws
24     Conteiner conteiner = builder.criarConteiner();
25     try (Connection conexao = fabrica.criarConexao();
26         Statement declaracao = conexao.createStatement();
27         ResultSet resultado = declaracao.executeQuery(USUARIOS_CONSULTA)) {
28         while (resultado.next()) {
29             Carga carga = builder.criarCarga(resultado);
30             builder.adicionarCarga(conteiner, carga);
31         }
32     }
33     builder.fecharConteiner(conteiner);
}
Writable | Smart Insert | 25 : 58 : 809 | 284M of 587M |
```

Decorator



The screenshot shows a Java code editor window titled "ClienteAntes.java". The code implements the Decorator pattern:

```
File Edit Source Refactor Navigate Search Project Run Window Help
ClienteAntes.java X
1 package school.cesar.esa.padroes.estruturais.decorator;
2
3 public class ClienteAntes {
4     public static void main(String[] args) throws Exception {
5         String url = "jdbc:h2:mem:";
6         String usuario = "sa";
7         String senha = "";
8         ConexaoFabrica fabrica = new ConexaoFabrica(url, usuario, senha);
9         UsuarioRepositorio repositorio = new UsuarioRepositorio(fabrica);
10        repositorio.listar();
11    }
12 }
```

The code defines a package named "school.cesar.esa.padroes.estruturais.decorator". It contains a class "ClienteAntes" with a main method. The main method initializes a database connection (url: "jdbc:h2:mem:"), sets the user to "sa", and creates a "ConexaoFabrica" object. It then creates a "UsuarioRepositorio" object using this fabrica and calls its "listar()" method.

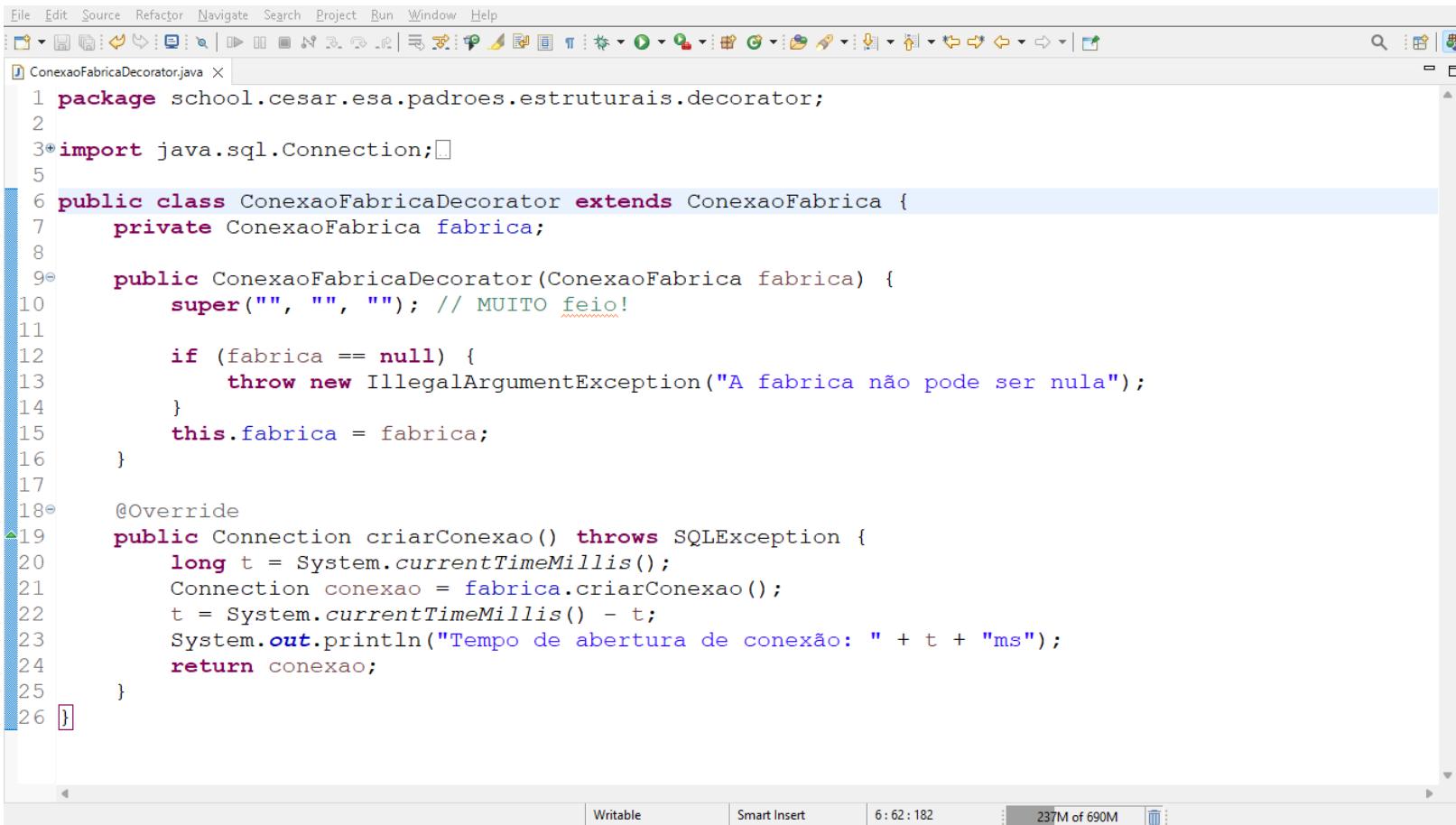
Decorator

- Alguns SGBDs podem ser configurados para
 - Aceitar um número máximo de conexões abertas simultaneamente
 - Após esse número ser alcançado, um processo que tente abrir uma nova conexão ficará esperando até que alguma das conexões seja fechada
- Como podemos medir e registrar o tempo de abertura de uma conexão sem alterar a classe ConexaoFabrica?

Decorator



Decorator

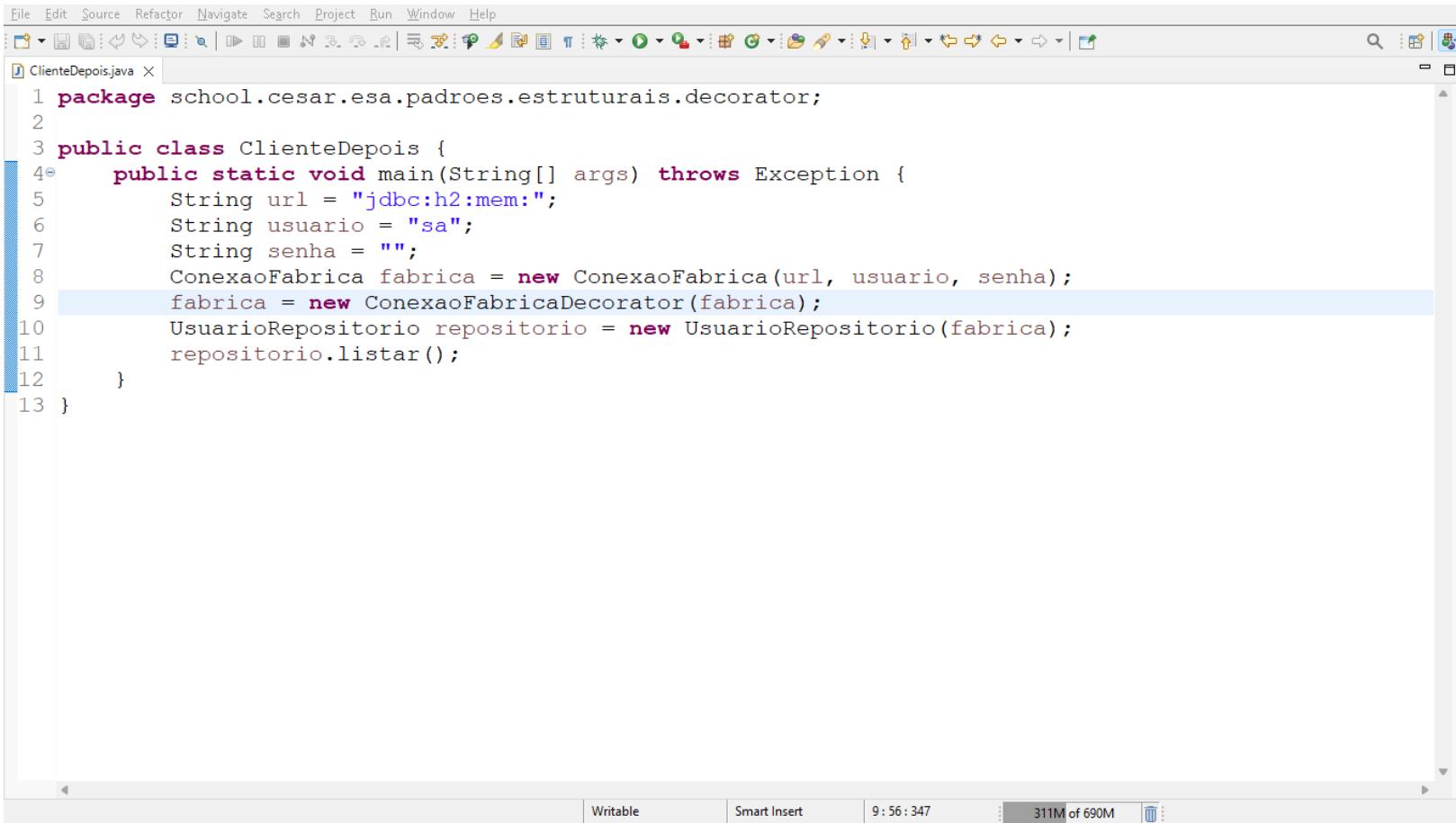


The screenshot shows a Java code editor window with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Code Editor:** The file `ConexaoFabricaDecorator.java` is open. The code implements the Decorator pattern for database connections.
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.decorator;
2
3 import java.sql.Connection;
4
5 public class ConexaoFabricaDecorator extends ConexaoFabrica {
6     private ConexaoFabrica fabrica;
7
8     public ConexaoFabricaDecorator(ConexaoFabrica fabrica) {
9         super("", "", ""); // MUITO feio!
10    }
11
12    if (fabrica == null) {
13        throw new IllegalArgumentException("A fabrica não pode ser nula");
14    }
15    this.fabrica = fabrica;
16 }
17
18 @Override
19 public Connection criarConexao() throws SQLException {
20     long t = System.currentTimeMillis();
21     Connection conexao = fabrica.criarConexao();
22     t = System.currentTimeMillis() - t;
23     System.out.println("Tempo de abertura de conexão: " + t + "ms");
24     return conexao;
25 }
26 }
```
- Status Bar:** Shows the file is Writable, Smart Insert is enabled, the current time is 6:62:182, and the disk usage is 237M of 690M.

Decorator



The screenshot shows a Java code editor window with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
ClientDepois.java X
1 package school.cesar.esa.padroes.estruturais.decorator;
2
3 public class ClienteDepois {
4     public static void main(String[] args) throws Exception {
5         String url = "jdbc:h2:mem:";
6         String usuario = "sa";
7         String senha = "";
8         ConexaoFabrica fabrica = new ConexaoFabrica(url, usuario, senha);
9         fabrica = new ConexaoFabricaDecorator(fabrica);
10        UsuarioRepositorio repositorio = new UsuarioRepositorio(fabrica);
11        repositorio.listar();
12    }
13 }
```

The code implements the Decorator pattern. It defines a `ClienteDepois` class with a `main` method. Inside the `main` method, it creates a `ConexaoFabrica` object and then wraps it with a `ConexaoFabricaDecorator`. Finally, it creates a `UsuarioRepositorio` object using the decorated `ConexaoFabrica` and calls its `listar` method.

Decorator

GETTING BETTER

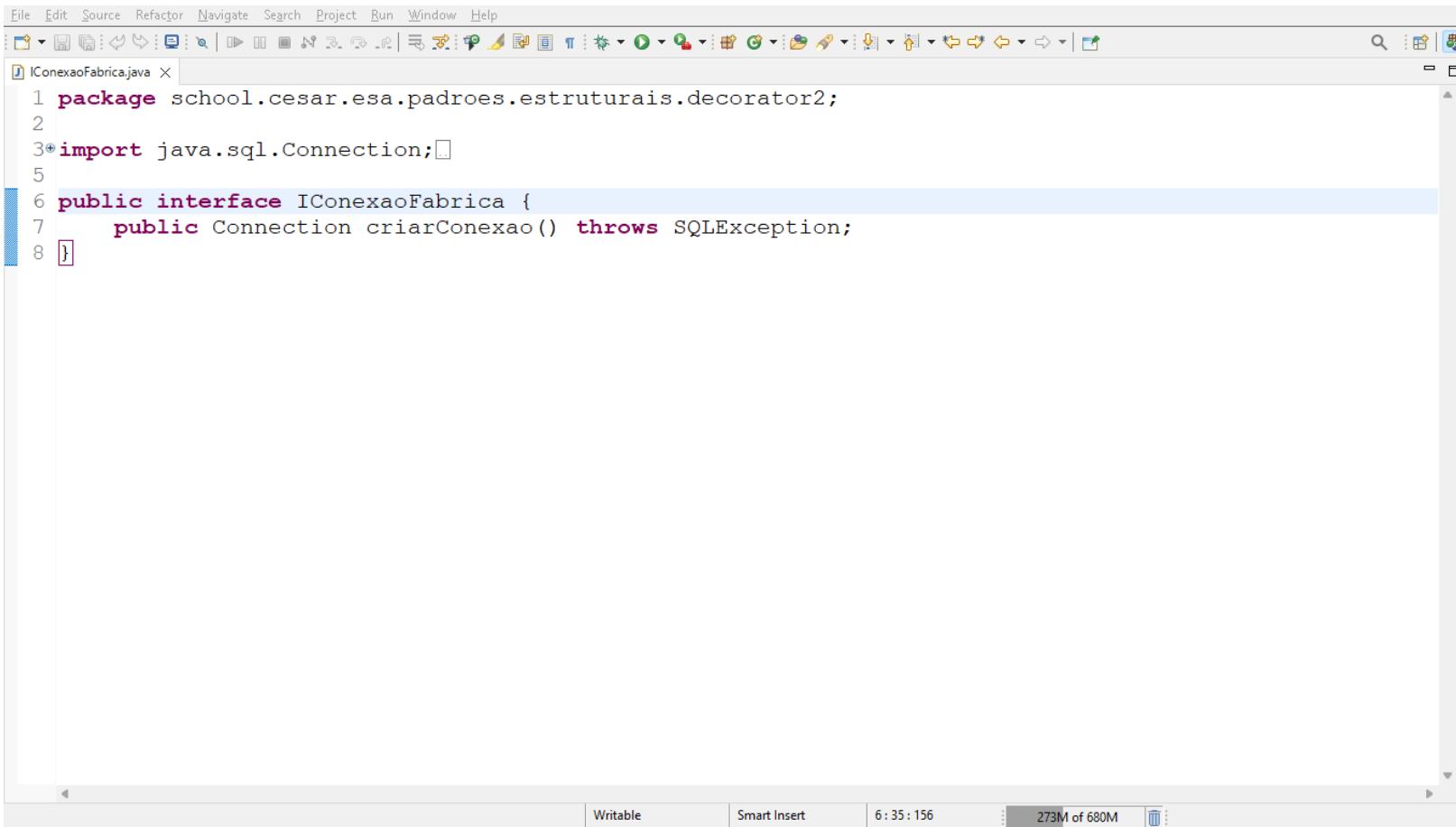
Music and Words by CHRIS DEAN and PETER WILDEMAN



NORTHERN SONGS LIMITED

31

Decorator



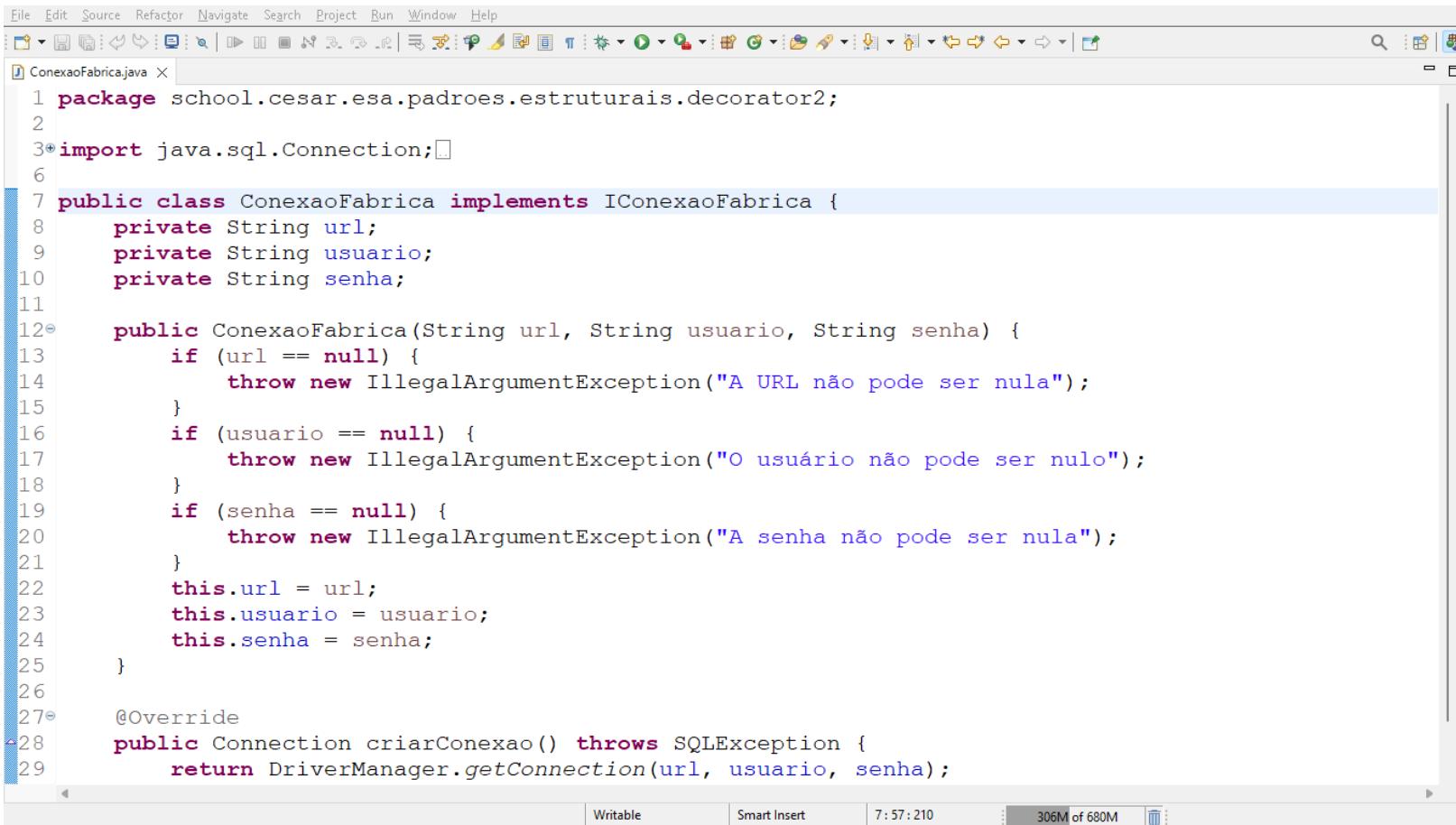
The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes icons for file operations like Open, Save, Print, and various search and navigation functions.
- Code Editor:** The file `IConexaoFabrica.java` is open. The code is as follows:

```
1 package school.cesar.esa.padroes.estruturais.decorator2;
2
3 import java.sql.Connection;
4
5 public interface IConexaoFabrica {
6     public Connection criarConexao() throws SQLException;
7 }
8 }
```

- Status Bar:** Shows "Writable", "Smart Insert", the current time "6:35:156", and disk usage "273M of 680M".

Decorator

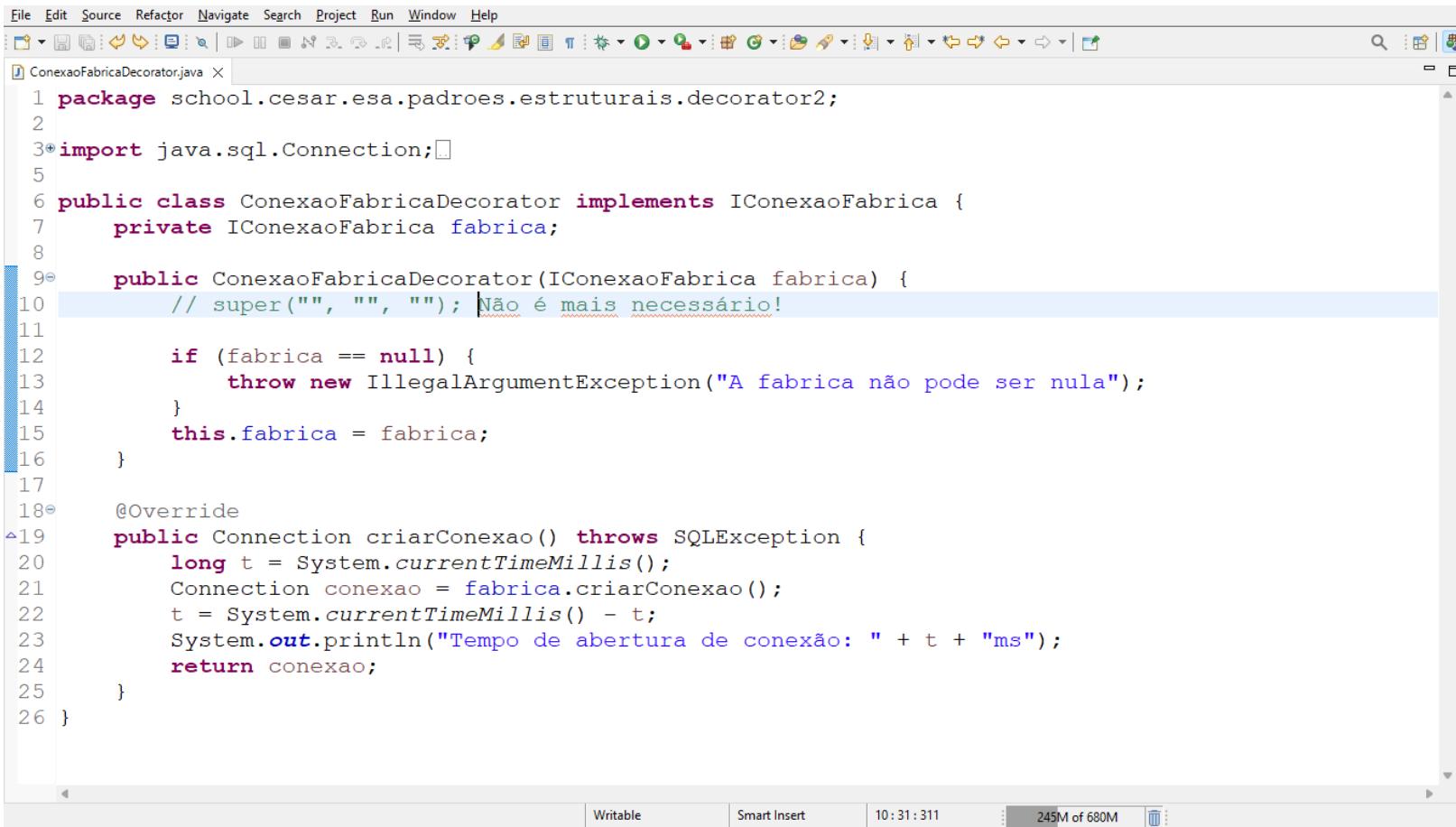


The screenshot shows a Java code editor window with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
ConexaoFabrica.java X
1 package school.cesar.esa.padroes.estruturais.decorator2;
2
3 import java.sql.Connection;
4
5
6 public class ConexaoFabrica implements IConexaoFabrica {
7     private String url;
8     private String usuario;
9     private String senha;
10
11    public ConexaoFabrica(String url, String usuario, String senha) {
12        if (url == null) {
13            throw new IllegalArgumentException("A URL não pode ser nula");
14        }
15        if (usuario == null) {
16            throw new IllegalArgumentException("O usuário não pode ser nulo");
17        }
18        if (senha == null) {
19            throw new IllegalArgumentException("A senha não pode ser nula");
20        }
21        this.url = url;
22        this.usuario = usuario;
23        this.senha = senha;
24    }
25
26    @Override
27    public Connection criarConexao() throws SQLException {
28        return DriverManager.getConnection(url, usuario, senha);
29    }
}
Writable | Smart Insert | 7:57:210 | 306M of 680M
```

The code defines a class `ConexaoFabrica` that implements an interface `IConexaoFabrica`. It has private fields for `url`, `usuario`, and `senha`. The constructor validates these parameters and initializes them. The `criarConexao` method returns a database connection using `DriverManager.getConnection`.

Decorator

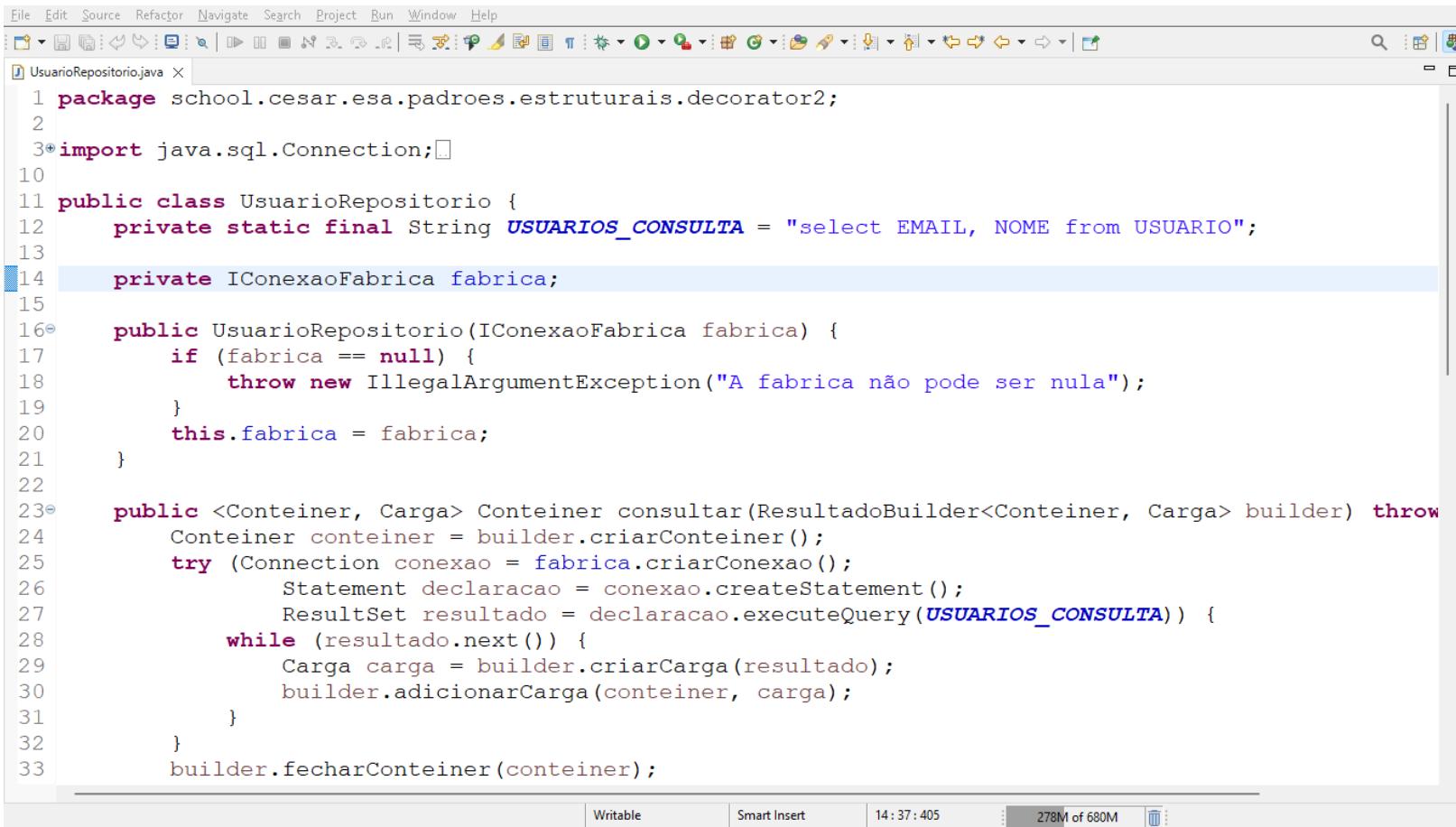


The screenshot shows a Java code editor window with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Code Editor:** The file `ConexaoFabricaDecorator.java` is open. The code implements the `IConexaoFabrica` interface and uses the Decorator pattern to wrap a connection.
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.decorator2;
2
3 import java.sql.Connection;
4
5 public class ConexaoFabricaDecorator implements IConexaoFabrica {
6     private IConexaoFabrica fabrica;
7
8     public ConexaoFabricaDecorator(IConexaoFabrica fabrica) {
9         // super("", "", ""); Não é mais necessário!
10    }
11
12    if (fabrica == null) {
13        throw new IllegalArgumentException("A fabrica não pode ser nula");
14    }
15    this.fabrica = fabrica;
16 }
17
18 @Override
19 public Connection criarConexao() throws SQLException {
20     long t = System.currentTimeMillis();
21     Connection conexao = fabrica.criarConexao();
22     t = System.currentTimeMillis() - t;
23     System.out.println("Tempo de abertura de conexão: " + t + "ms");
24     return conexao;
25 }
26 }
```
- Status Bar:** Shows the file is Writable, Smart Insert is enabled, the current time is 10:31:311, and the memory usage is 245M of 680M.

Decorator

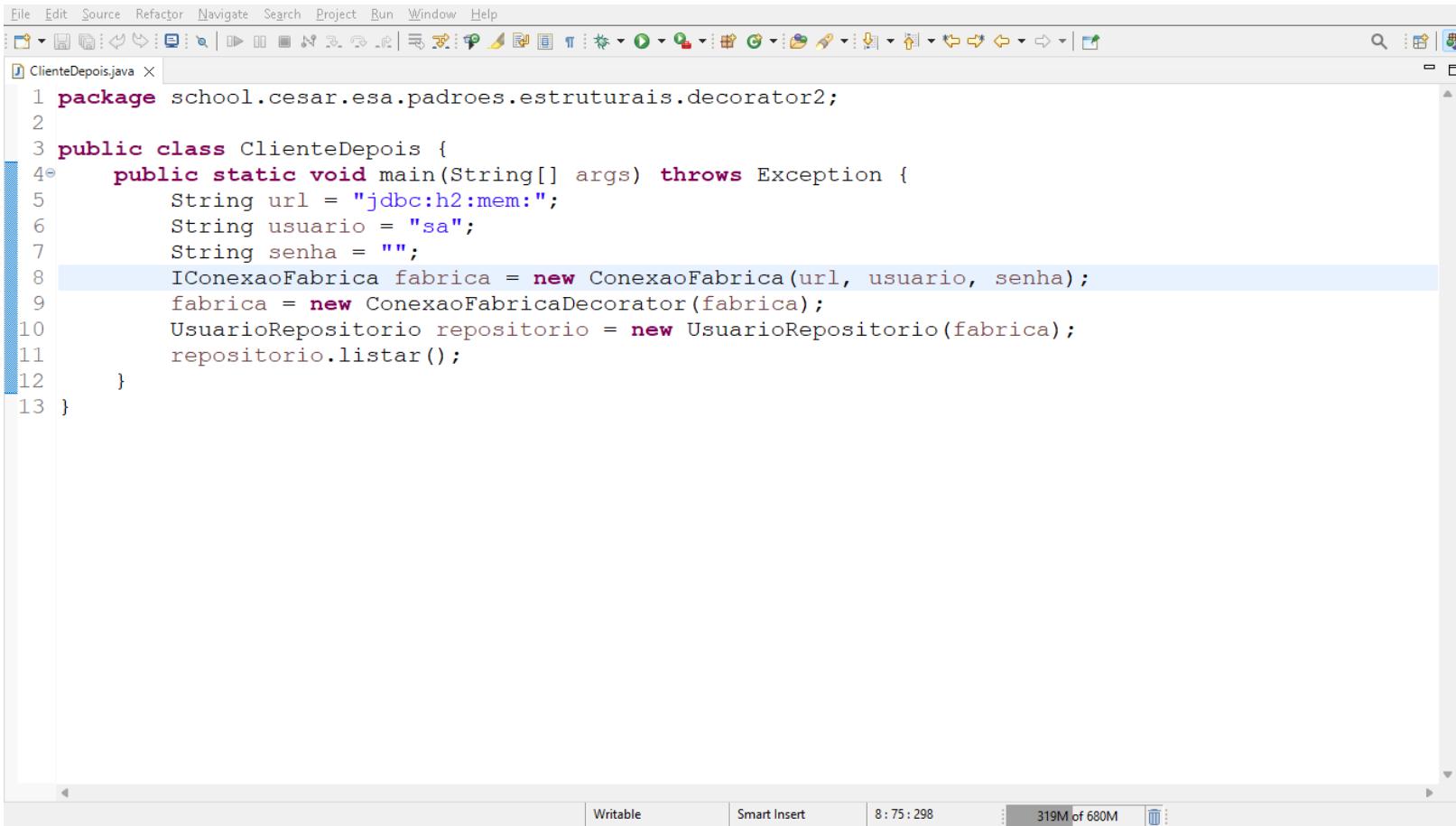


The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes icons for file operations like Open, Save, Print, and various search and navigation functions.
- Code Area:** The file `UsuarioRepositorio.java` is open. The code implements the Decorator pattern for managing user data from a database.
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.decorator2;
2
3 import java.sql.Connection;
4
5 public class UsuarioRepositorio {
6     private static final String USUARIOS_CONSULTA = "select EMAIL, NOME from USUARIO";
7
8     private IConexaoFabrica fabrica;
9
10    public UsuarioRepositorio(IConexaoFabrica fabrica) {
11        if (fabrica == null) {
12            throw new IllegalArgumentException("A fabrica não pode ser nula");
13        }
14        this.fabrica = fabrica;
15    }
16
17    public <Conteiner, Carga> Conteiner consultar(ResultadoBuilder<Conteiner, Carga> builder) throws
18        Conteiner conteiner = builder.criarConteiner();
19        try (Connection conexao = fabrica.criarConexao();
20            Statement declaracao = conexao.createStatement();
21            ResultSet resultado = declaracao.executeQuery(USUARIOS_CONSULTA)) {
22            while (resultado.next()) {
23                Carga carga = builder.criarCarga(resultado);
24                builder.adicionarCarga(conteiner, carga);
25            }
26        }
27        builder.fecharConteiner(conteiner);
28    }
29}
```
- Status Bar:** Shows Writable, Smart Insert, 14:37:405, 278M of 680M, and a trash bin icon.

Decorator



The screenshot shows a Java code editor window with the following code:

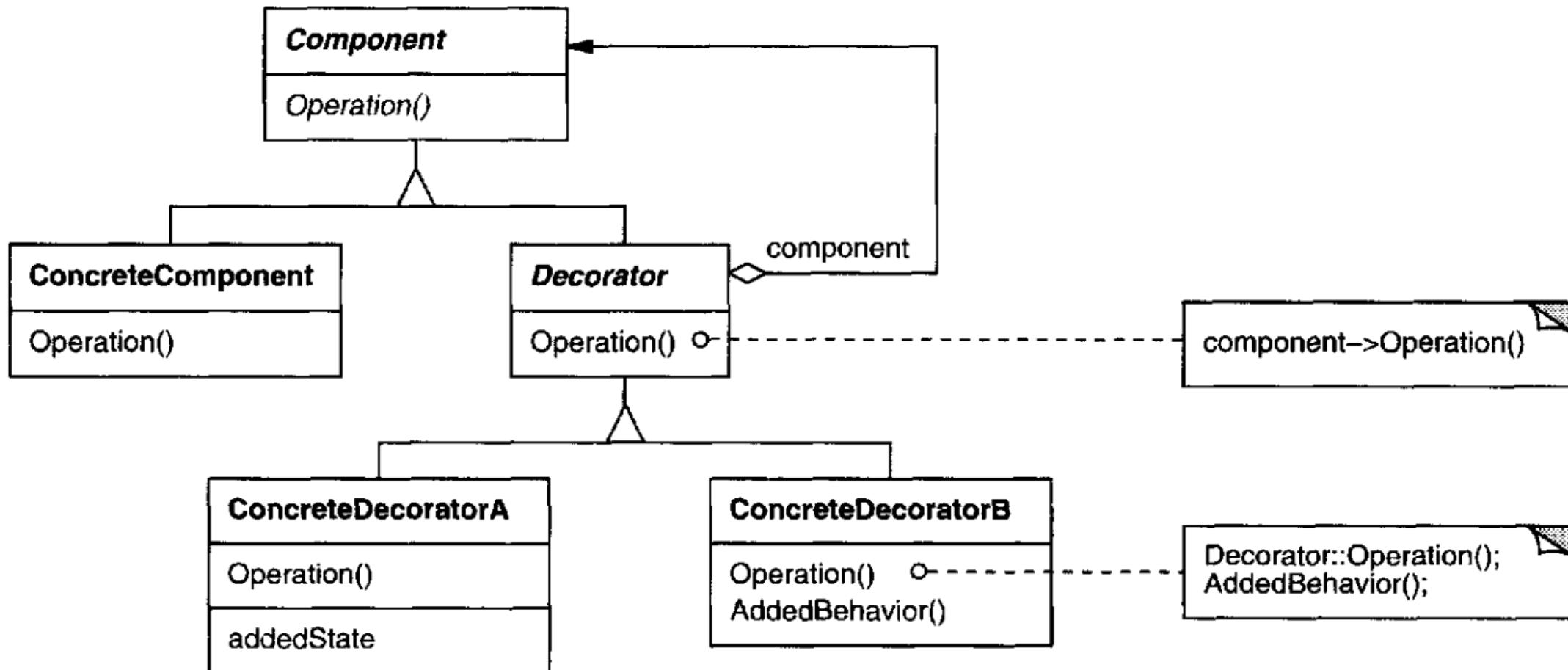
```
File Edit Source Refactor Navigate Search Project Run Window Help
ClientDepois.java X
1 package school.cesar.esa.padroes.estruturais.decorator2;
2
3 public class ClienteDepois {
4     public static void main(String[] args) throws Exception {
5         String url = "jdbc:h2:mem:";
6         String usuario = "sa";
7         String senha = "";
8         IConexaoFabrica fabrica = new ConexaoFabrica(url, usuario, senha);
9         fabrica = new ConexaoFabricaDecorator(fabrica);
10        UsuarioRepositorio repositorio = new UsuarioRepositorio(fabrica);
11        repositorio.listar();
12    }
13 }
```

The code implements the Decorator pattern. It defines a `ClienteDepois` class with a `main` method. Inside the `main` method, it creates a `IConexaoFabrica` object using the `new ConexaoFabrica(url, usuario, senha);` constructor. This object is then wrapped by a `ConexaoFabricaDecorator` object using the `fabrica = new ConexaoFabricaDecorator(fabrica);` assignment. Finally, a `UsuarioRepositorio` object is created using the decorated `fabrica` object, and its `listar()` method is called.

Decorator

“Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.”

Decorator



Decorator

- SOLID
 - Responsabilidade única (**S**ingle responsibility)
 - Aberto-fechado (**O**pen-closed)
 - Substituição de Liskov (**L**iskob substitution)
 - Segregação de interfaces (**I**nterface segregation)
 - Inversão de dependências (**D**evelopment dependency inversion)
- Prefira composição à herança
- Demeter

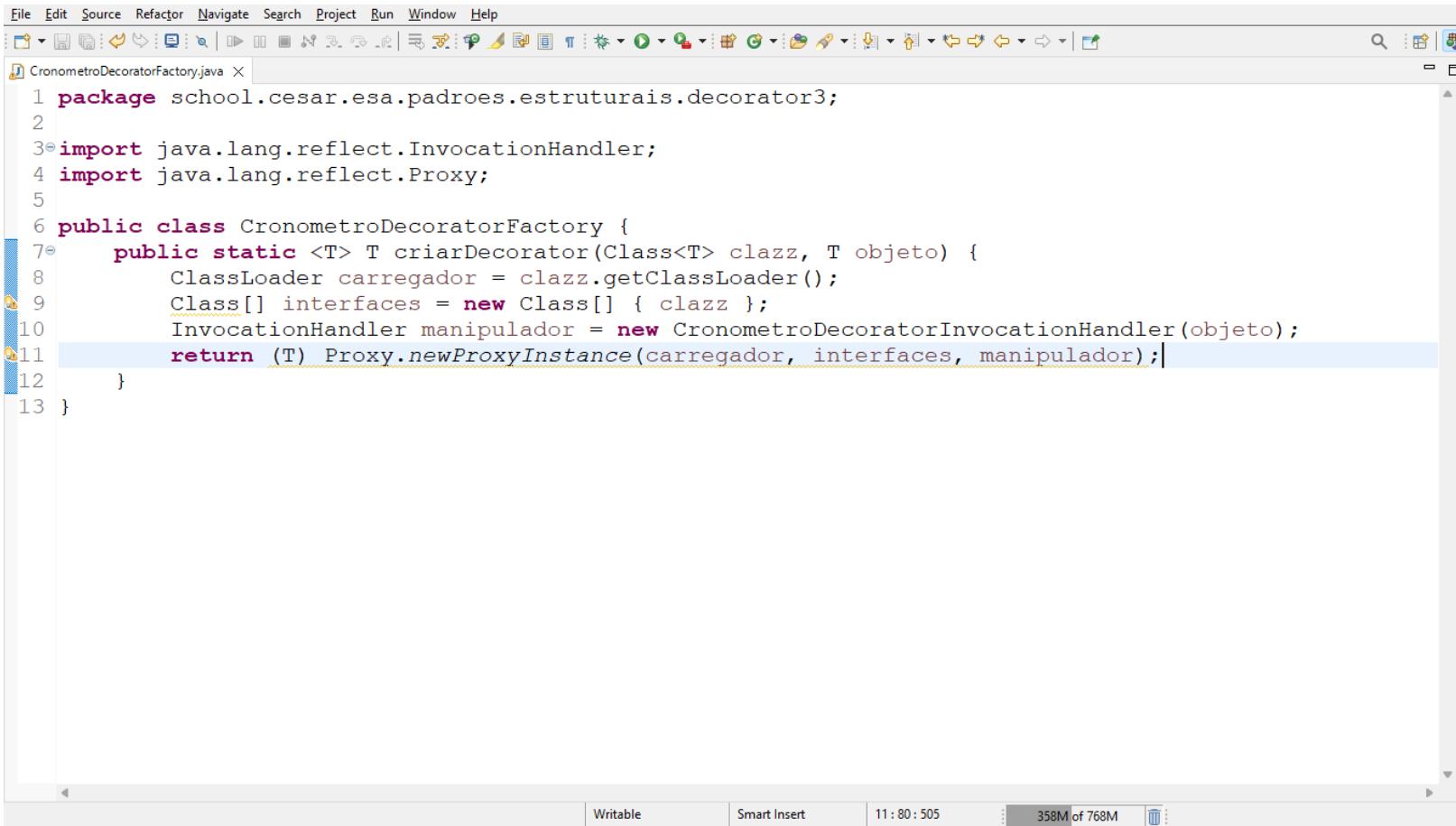
Decorator

- Integridade conceitual
- (Alta) Coesão
- (Baixo) Acoplamento
- Ocultamento de informações

Decorator



Decorator

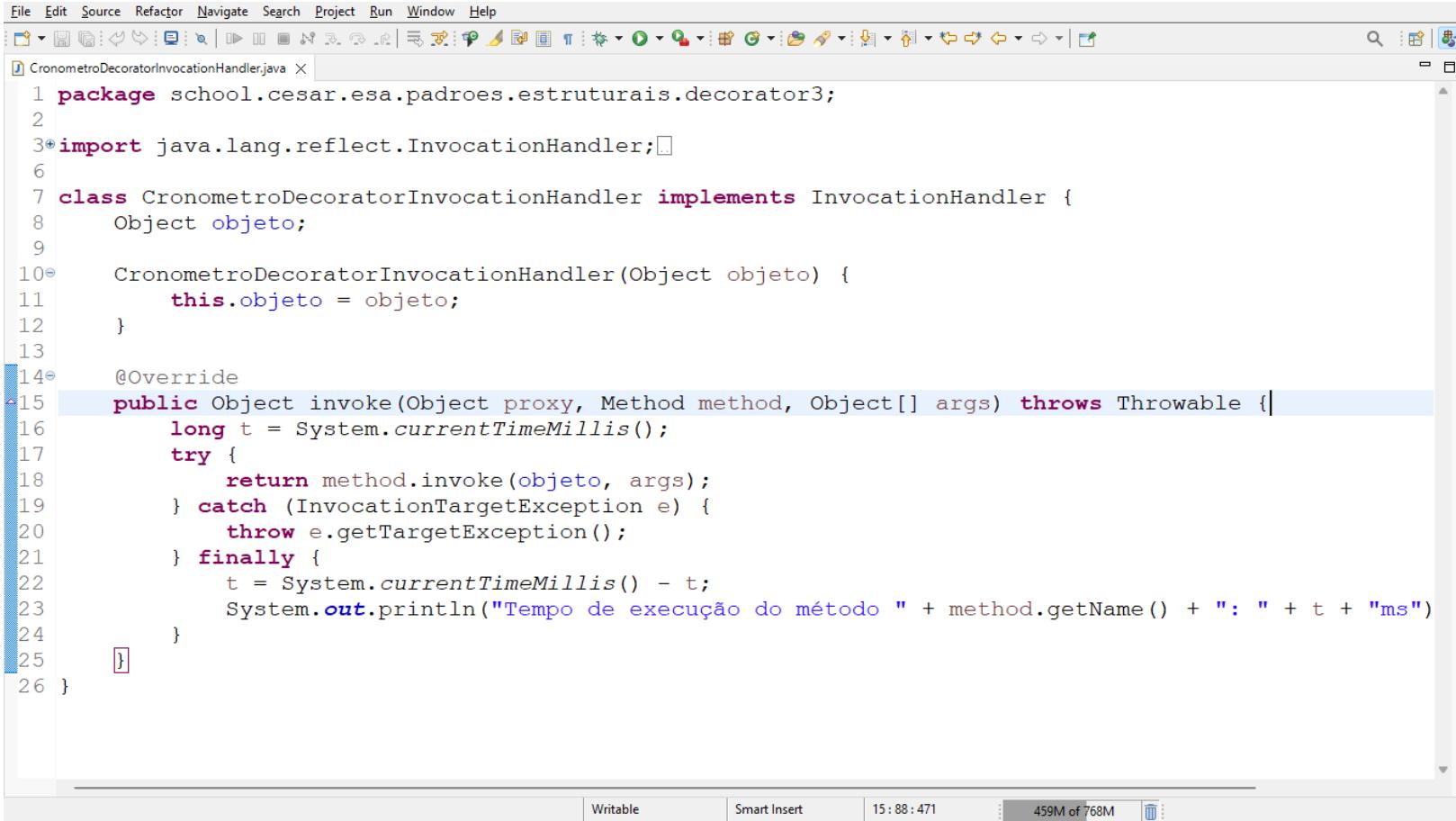


The screenshot shows a Java code editor window with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
CronometroDecoratorFactory.java X
1 package school.cesar.esa.padroes.estruturais.decorator3;
2
3 import java.lang.reflect.InvocationHandler;
4 import java.lang.reflect.Proxy;
5
6 public class CronometroDecoratorFactory {
7     public static <T> T criarDecorator(Class<T> clazz, T objeto) {
8         ClassLoader carregador = clazz.getClassLoader();
9         Class[] interfaces = new Class[] { clazz };
10        InvocationHandler manipulador = new CronometroDecoratorInvocationHandler(objeto);
11        return (T) Proxy.newProxyInstance(carregador, interfaces, manipulador);|
12    }
13 }
```

The code implements the Decorator pattern. It defines a factory method `criarDecorator` that takes a class and an object, creates a class loader, defines the interfaces, creates an invocation handler, and then uses `Proxy.newProxyInstance` to return a proxy object.

Decorator

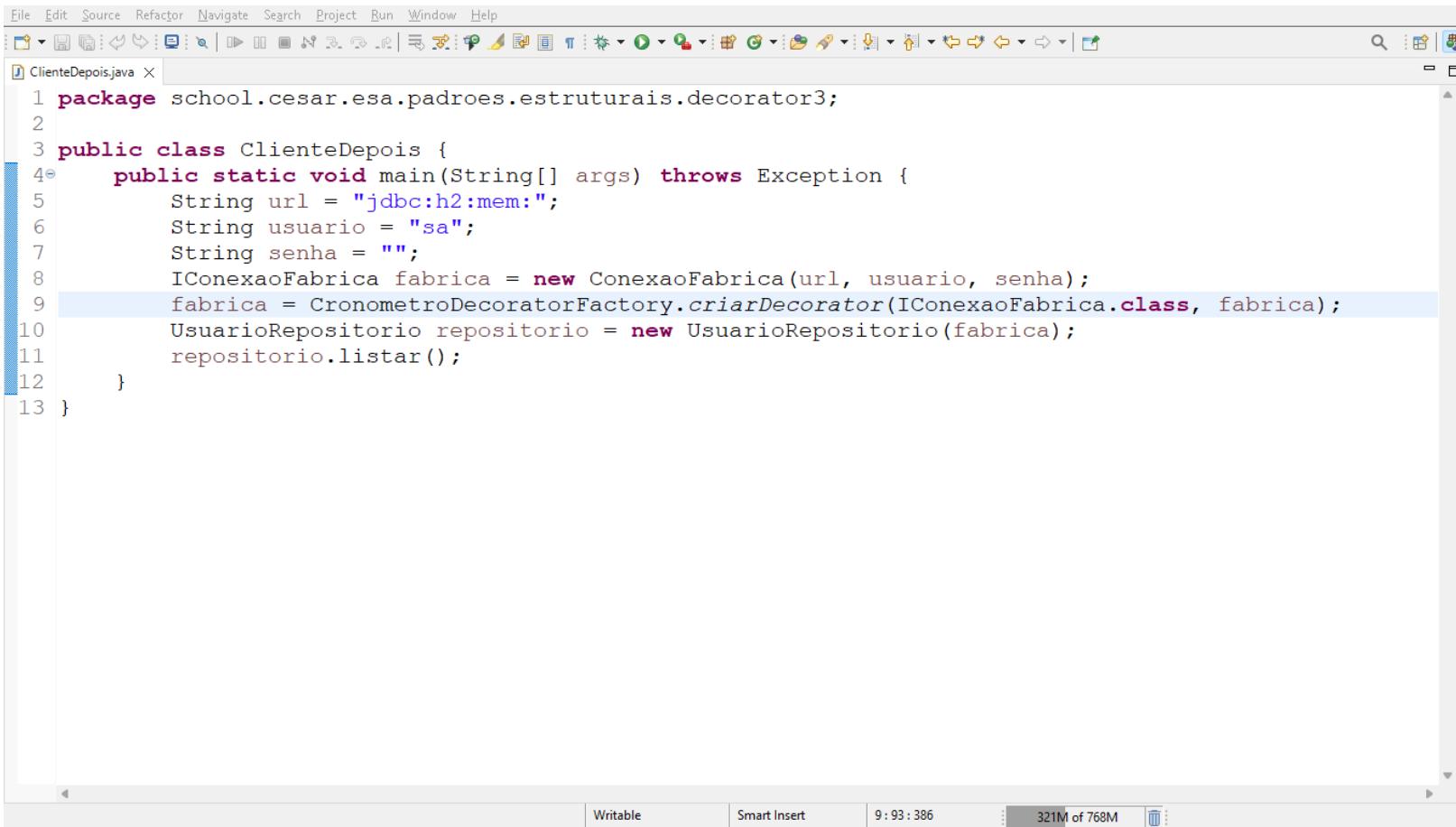


The screenshot shows a Java code editor with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
CronometroDecoratorInvocationHandler.java ×
1 package school.cesar.esa.padroes.estruturais.decorator3;
2
3 import java.lang.reflect.InvocationHandler;[]
4
5 class CronometroDecoratorInvocationHandler implements InvocationHandler {
6     Object objeto;
7
8     CronometroDecoratorInvocationHandler(Object objeto) {
9         this.objeto = objeto;
10    }
11
12    @Override
13    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
14        long t = System.currentTimeMillis();
15        try {
16            return method.invoke(objeto, args);
17        } catch (InvocationTargetException e) {
18            throw e.getTargetException();
19        } finally {
20            t = System.currentTimeMillis() - t;
21            System.out.println("Tempo de execução do método " + method.getName() + ": " + t + "ms")
22        }
23    }
24}
25
26 }
```

The code implements the `InvocationHandler` interface. It maintains a reference to the target object (`objeto`) and overrides the `invoke` method. Inside the `invoke` method, it measures the execution time by getting the current time before and after the target method is called. The result is then printed to the console.

Decorator



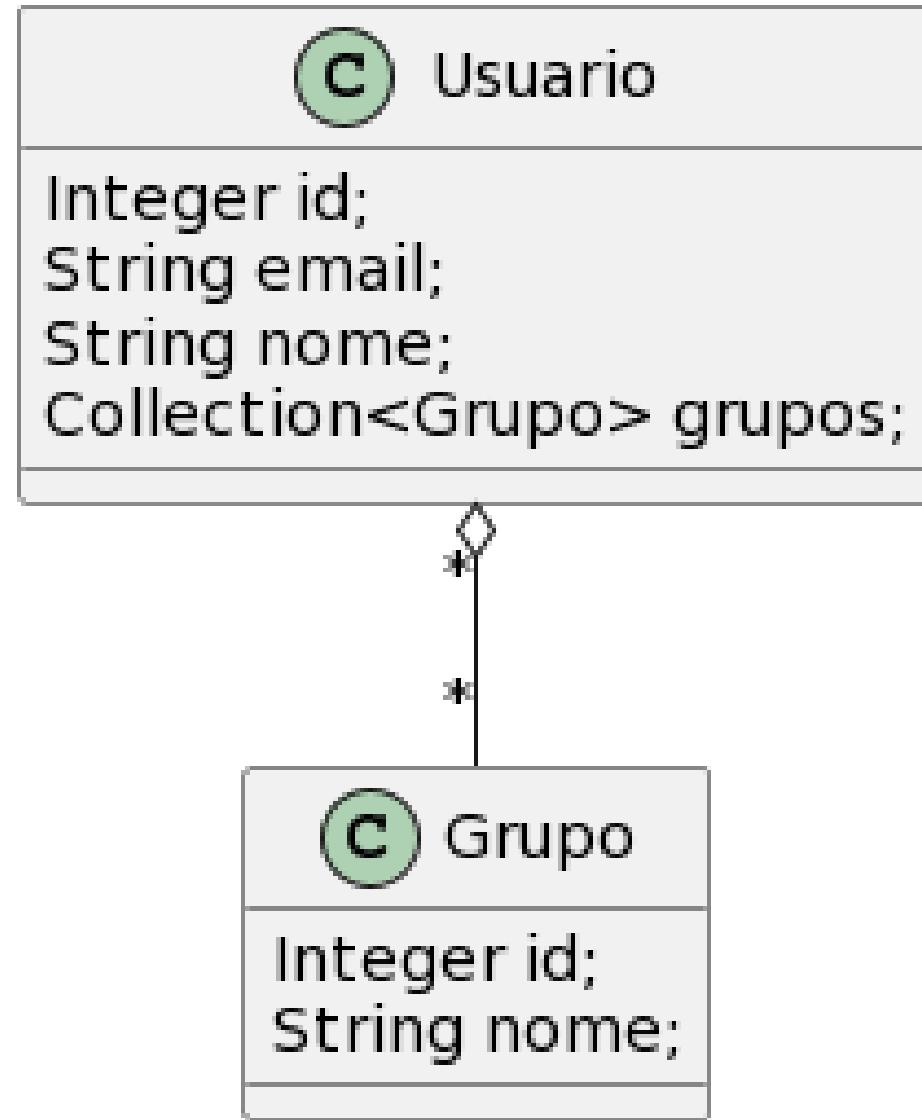
The screenshot shows a Java code editor window with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
ClientDepois.java X
1 package school.cesar.esa.padroes.estruturais.decorator3;
2
3 public class ClienteDepois {
4     public static void main(String[] args) throws Exception {
5         String url = "jdbc:h2:mem:";
6         String usuario = "sa";
7         String senha = "";
8         IConexaoFabrica fabrica = new ConexaoFabrica(url, usuario, senha);
9         fabrica = CronometroDecoratorFactory.criarDecorator(IConexaoFabrica.class, fabrica);
10        UsuarioRepositorio repositorio = new UsuarioRepositorio(fabrica);
11        repositorio.listar();
12    }
13 }
```

The line `fabrica = CronometroDecoratorFactory.criarDecorator(IConexaoFabrica.class, fabrica);` is highlighted with a light blue background.

Proxy

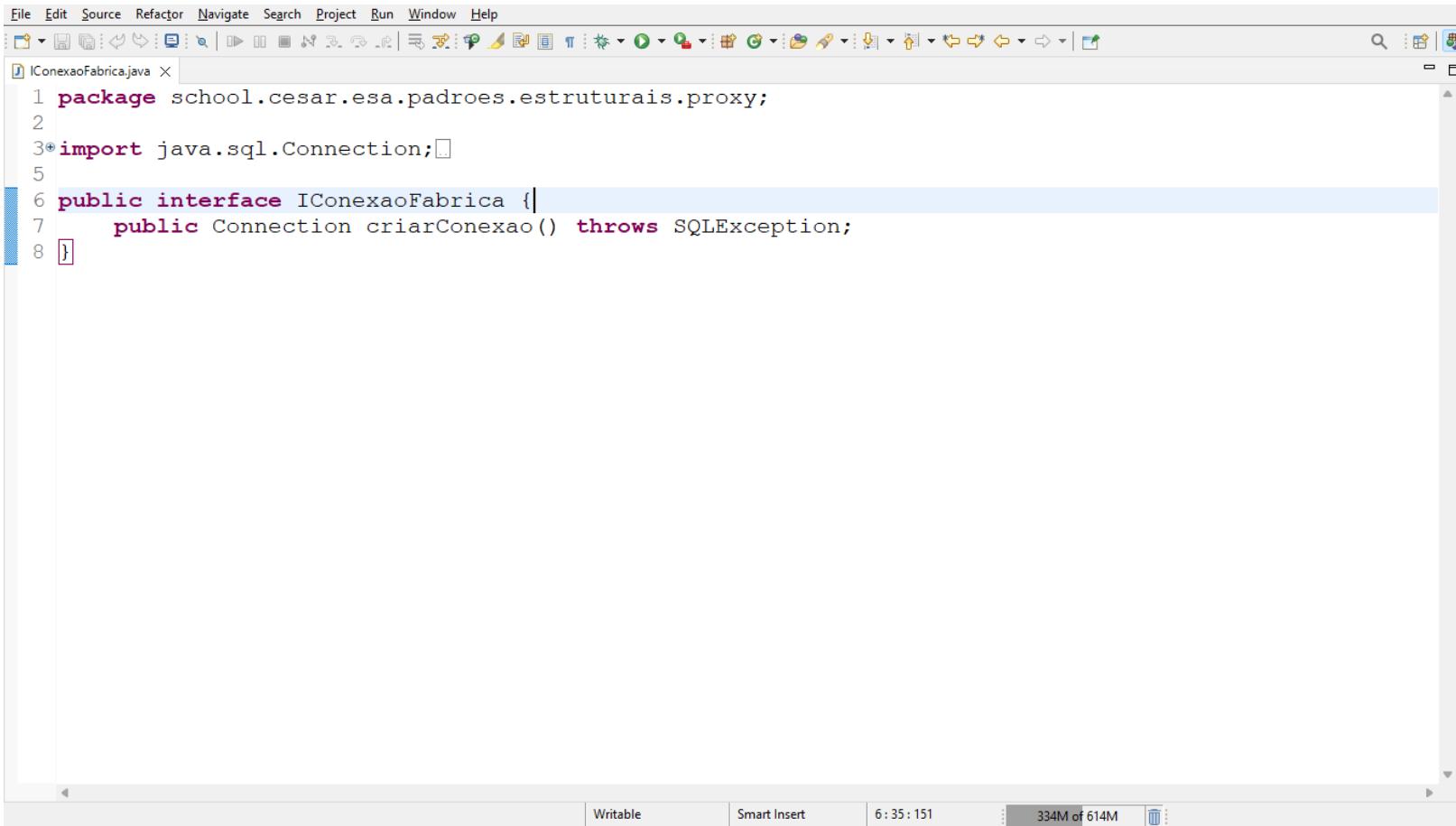
Proxy



Proxy



Proxy



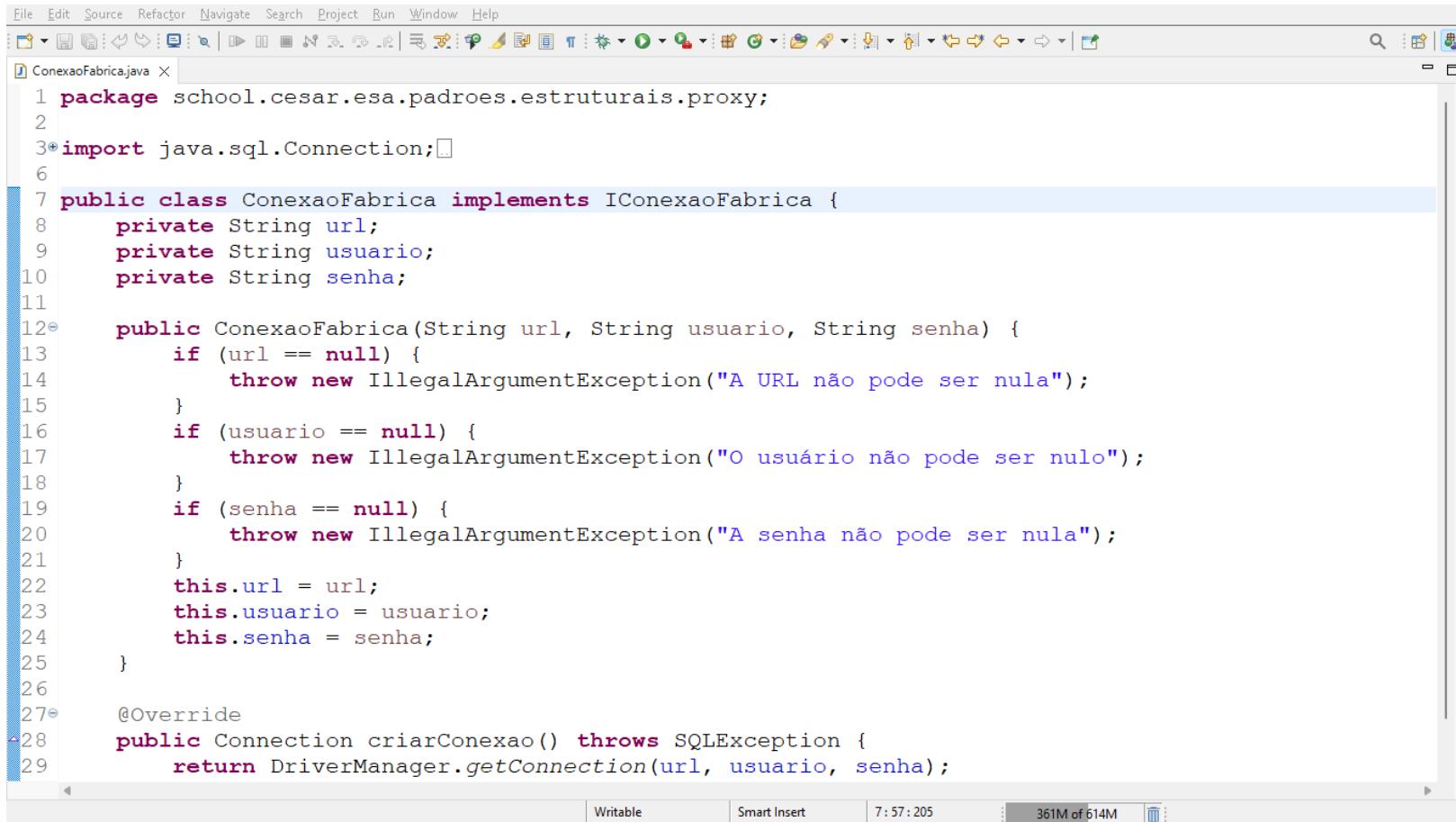
The screenshot shows a Java code editor window with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Code Editor:** The file `IConexaoFabrica.java` is open. The code defines a public interface with one method: `Connection criarConexao()` which throws `SQLException`.

```
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.sql.Connection;
4
5 public interface IConexaoFabrica {
6     public Connection criarConexao() throws SQLException;
7 }
8 }
```

- Status Bar:** Shows "Writable", "Smart Insert", the current time "6:35:151", and memory usage "334M of 614M".

Proxy

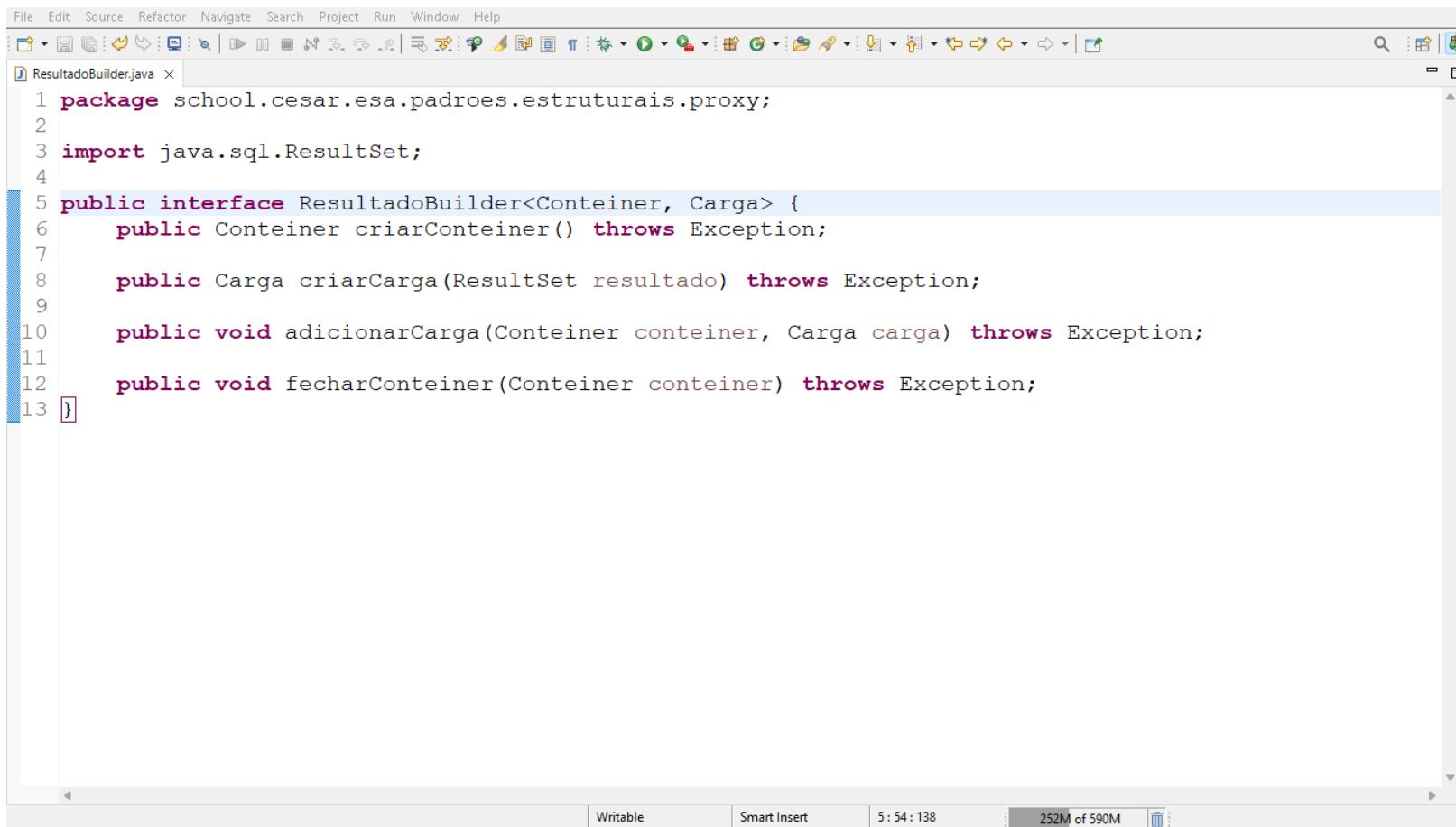


The screenshot shows a Java code editor window with the following code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
ConexaoFabrica.java X
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.sql.Connection;
4
5
6 public class ConexaoFabrica implements IConexaoFabrica {
7     private String url;
8     private String usuario;
9     private String senha;
10
11
12     public ConexaoFabrica(String url, String usuario, String senha) {
13         if (url == null) {
14             throw new IllegalArgumentException("A URL não pode ser nula");
15         }
16         if (usuario == null) {
17             throw new IllegalArgumentException("O usuário não pode ser nulo");
18         }
19         if (senha == null) {
20             throw new IllegalArgumentException("A senha não pode ser nula");
21         }
22         this.url = url;
23         this.usuario = usuario;
24         this.senha = senha;
25     }
26
27     @Override
28     public Connection criarConexao() throws SQLException {
29         return DriverManager.getConnection(url, usuario, senha);
30     }
}
Writable | Smart Insert | 7:57:205 | 361M of 614M
```

The code defines a class `ConexaoFabrica` that implements an interface `IConexaoFabrica`. It has private fields for `url`, `usuario`, and `senha`. The constructor takes these parameters and performs null checks. The `criarConexao` method returns a database connection using `DriverManager.getConnection`.

Proxy



The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Title Bar:** ResultadoBuilder.java X
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.sql.ResultSet;
4
5 public interface ResultadoBuilder<Conteiner, Carga> {
6     public Conteiner criarConteiner() throws Exception;
7
8     public Carga criarCarga(ResultSet resultado) throws Exception;
9
10    public void adicionarCarga(Conteiner conteiner, Carga carga) throws Exception;
11
12    public void fecharConteiner(Conteiner conteiner) throws Exception;
13 }
```
- Status Bar:** Writable Smart Insert 5:54:138 252M of 590M

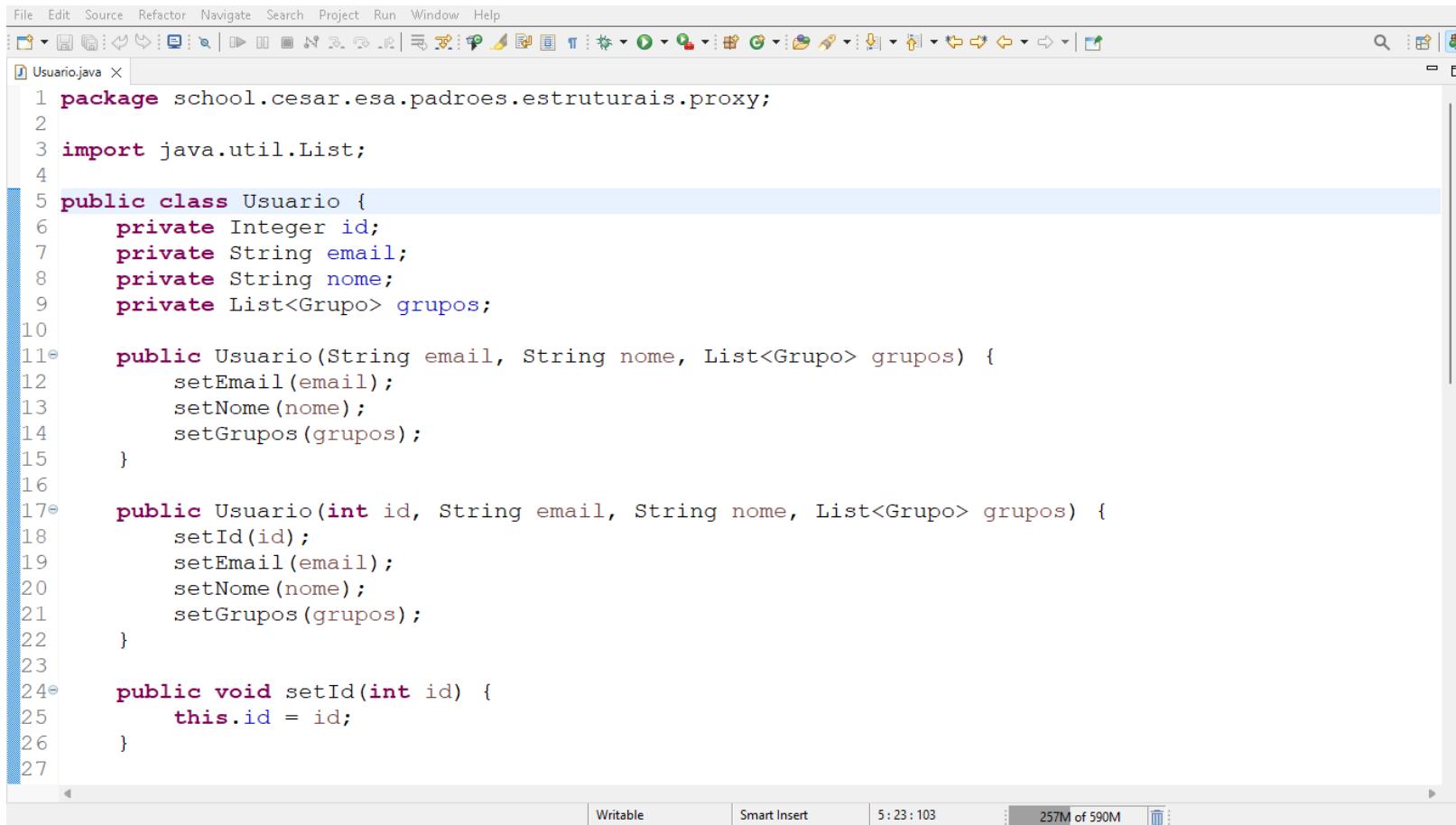
Proxy

The screenshot shows a Java code editor with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Print, and various search and navigation functions.
- Code Area:** The file `Repository.java` is open. The code implements the `Repositorio` interface, which takes an `IConexaoFabrica` as a parameter in its constructor. It then uses this fabrica to create a `Conteiner` and a `Carga` object, and performs a database query using `Statement` and `ResultSet`.
- Status Bar:** Shows the current file is `Repository.java`, the file is Writable, the current line is 17, the total file size is 217M of 590M, and the current time is 17: 68 : 454.

```
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.sql.Connection;
4
5
6 public class Repositorio {
7     private IConexaoFabrica fabrica;
8
9
10    public Repositorio(IConexaoFabrica fabrica) {
11        if (fabrica == null) {
12            throw new IllegalArgumentException("A fabrica não pode ser nula");
13        }
14        this.fabrica = fabrica;
15    }
16
17    public <Conteiner, Carga> Conteiner consultar(String consulta,
18          ResultadoBuilder<Conteiner, Carga> builder) throws Exception {
19        if (consulta == null) {
20            throw new IllegalArgumentException("A consulta não pode ser nula");
21        }
22        if (builder == null) {
23            throw new IllegalArgumentException("O builder não pode ser nulo");
24        }
25        Conteiner conteiner = builder.criarConteiner();
26        try (Connection conexao = fabrica.criarConexao();
27             Statement declaracao = conexao.createStatement();
28             ResultSet resultado = declaracao.executeQuery(consulta)) {
29            while (resultado.next()) {
```

Proxy



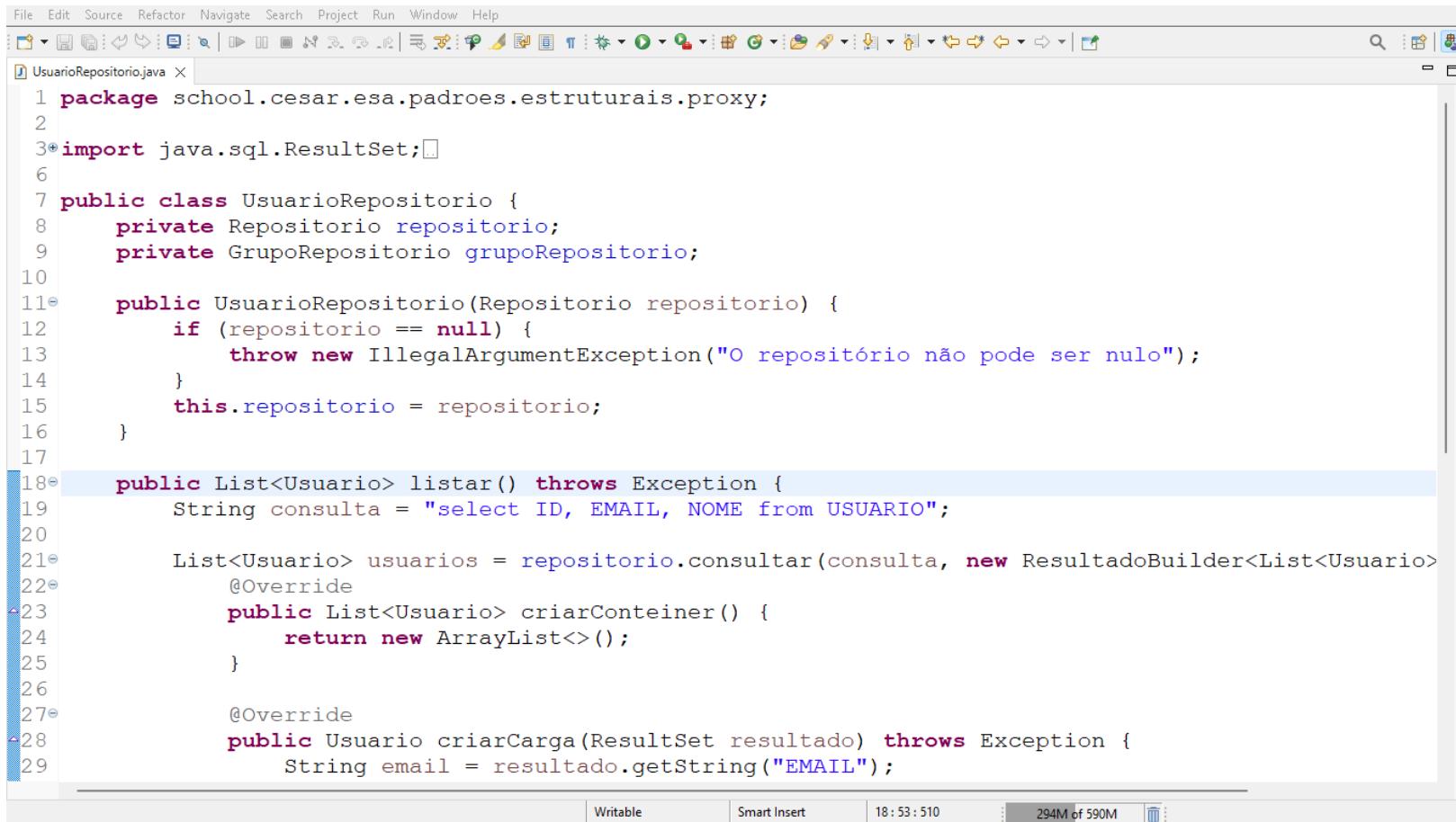
The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes various icons for file operations like Open, Save, Find, Copy, Paste, and others.
- Code Editor:** The file `Usuario.java` is open. The code defines a class `Usuario` with private fields `id`, `email`, `nome`, and a list of `Grupos`. It has two constructors and a setter for `id`.

```
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.util.List;
4
5 public class Usuario {
6     private Integer id;
7     private String email;
8     private String nome;
9     private List<Grupo> grupos;
10
11    public Usuario(String email, String nome, List<Grupo> grupos) {
12        setEmail(email);
13        setNome(nome);
14        setGrupos(grupos);
15    }
16
17    public Usuario(int id, String email, String nome, List<Grupo> grupos) {
18        setId(id);
19        setEmail(email);
20        setNome(nome);
21        setGrupos(grupos);
22    }
23
24    public void setId(int id) {
25        this.id = id;
26    }
27}
```

- Status Bar:** Writable, Smart Insert, 5:23:103, 257M of 590M

Proxy

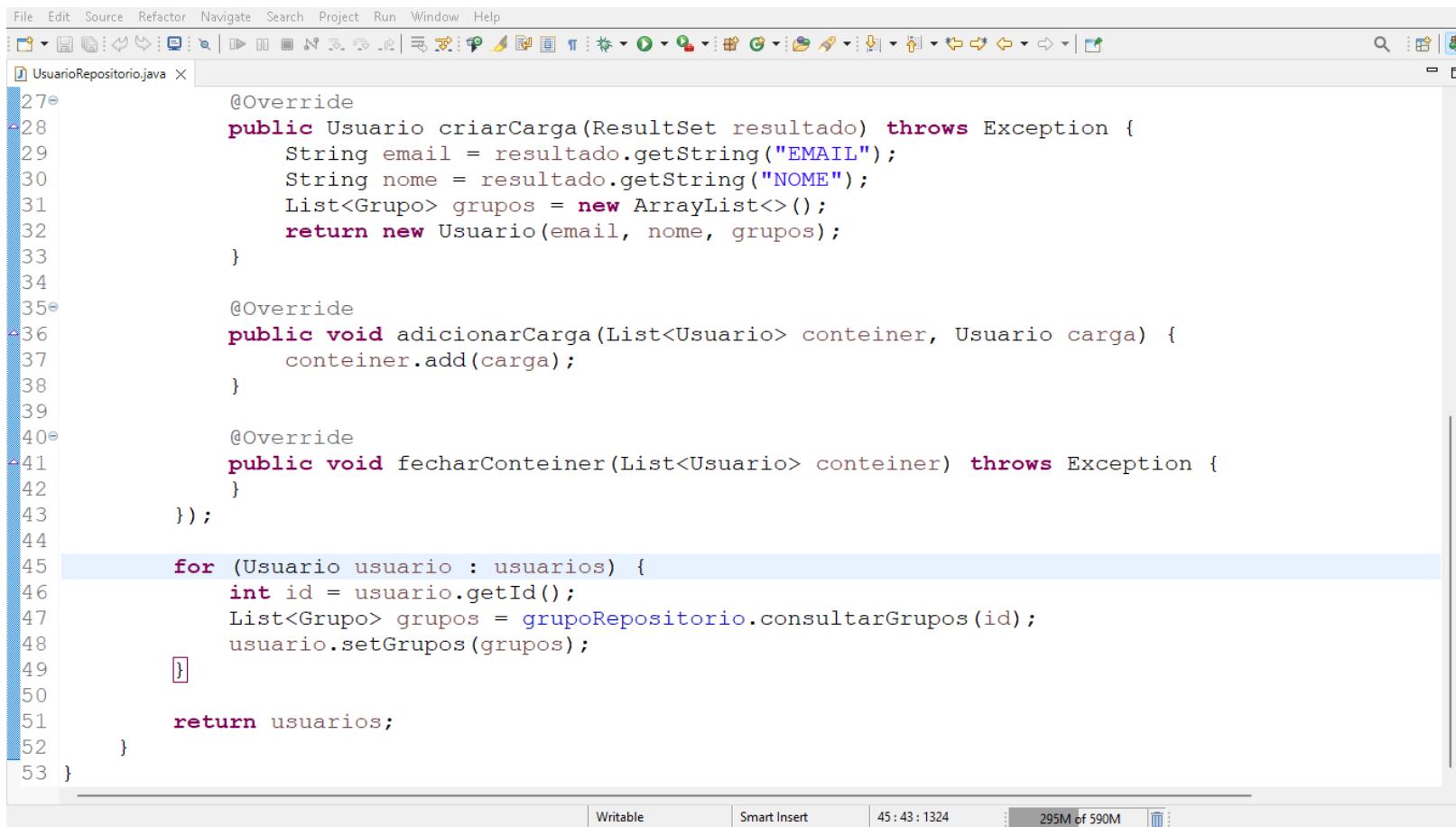


The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes various icons for file operations, search, and navigation.
- Code Editor:** The file `UsuarioRepositorio.java` is open. The code implements a repository pattern for managing users. It includes a constructor that takes a `Repositorio` instance and throws an `IllegalArgumentException` if it's null. It also includes methods for listing users and creating a new user from a `ResultSet`.
- Status Bar:** Shows Writable, Smart Insert, 18:53:510, 294M of 590M, and a trash icon.

```
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.sql.ResultSet;
4
5 public class UsuarioRepositorio {
6     private Repositorio repositorio;
7     private GrupoRepositorio grupoRepositorio;
8
9     public UsuarioRepositorio(Repositorio repositorio) {
10         if (repositorio == null) {
11             throw new IllegalArgumentException("O repositório não pode ser nulo");
12         }
13         this.repositorio = repositorio;
14     }
15
16     public List<Usuario> listar() throws Exception {
17         String consulta = "select ID, EMAIL, NOME from USUARIO";
18
19         List<Usuario> usuarios = repositorio.consultar(consulta, new ResultadoBuilder<List<Usuario>>
20             @Override
21             public List<Usuario> criarConteiner() {
22                 return new ArrayList<>();
23             }
24
25             @Override
26             public Usuario criarCarga(ResultSet resultado) throws Exception {
27                 String email = resultado.getString("EMAIL");
28             }
29         }
30     }
31 }
```

Proxy

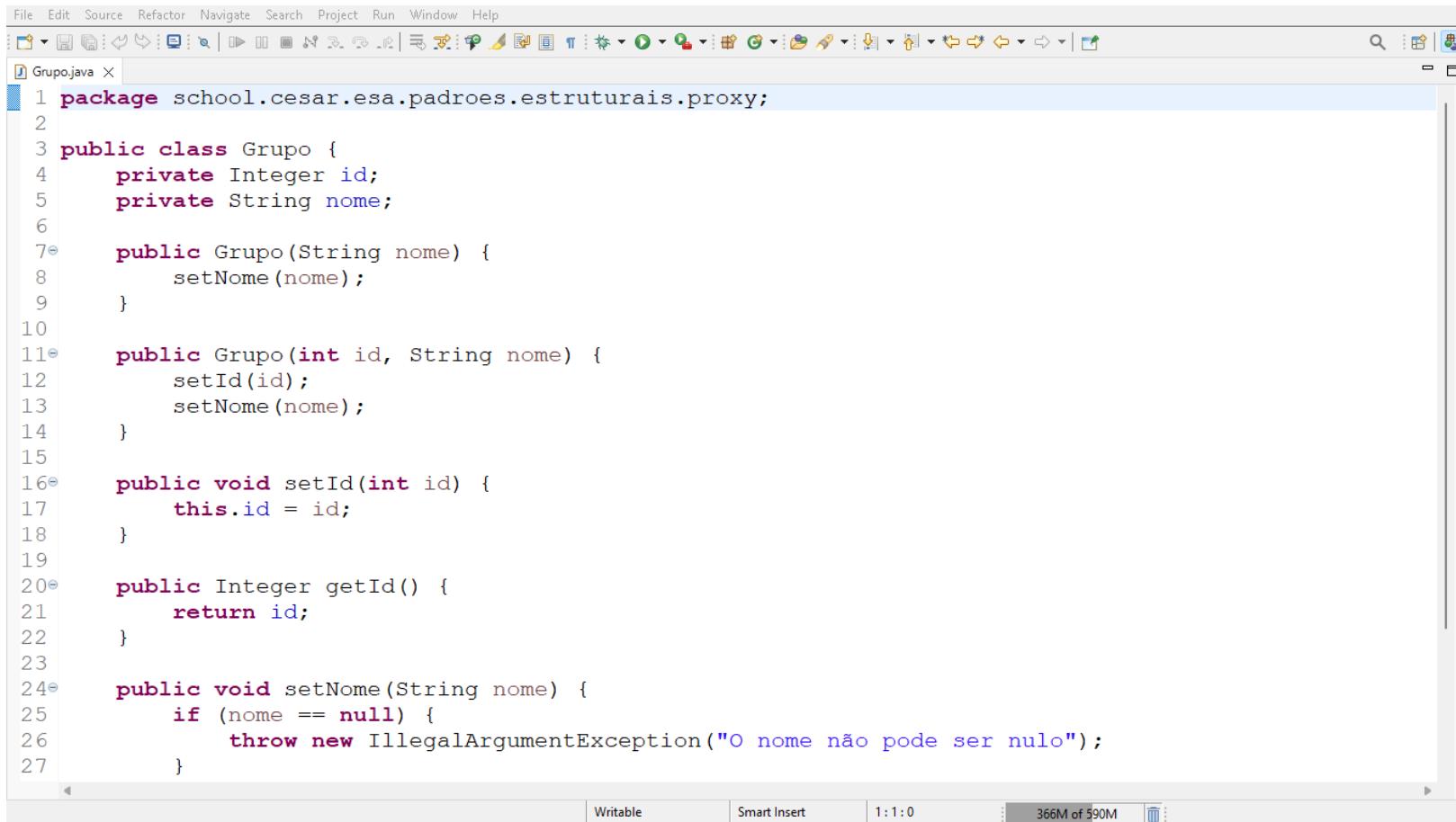


The screenshot shows a Java code editor window with the file `UsuarioRepositorio.java` open. The code implements a repository interface for managing users. It includes methods for creating a user from a database result set, adding users to a container, closing the container, and returning all users. The code uses Java annotations like `@Override`, `throws Exception`, and `Iterable`. It also imports `ResultSet`, `List`, and `ArrayList` from the Java standard library.

```
File Edit Source Refactor Navigate Search Project Run Window Help
UserRepository.java X
27  @Override
28  public Usuario criarCarga(ResultSet resultado) throws Exception {
29      String email = resultado.getString("EMAIL");
30      String nome = resultado.getString("NOME");
31      List<Grupo> grupos = new ArrayList<>();
32      return new Usuario(email, nome, grupos);
33  }
34
35  @Override
36  public void adicionarCarga(List<Usuario> conteiner, Usuario carga) {
37      conteiner.add(carga);
38  }
39
40  @Override
41  public void fecharConteiner(List<Usuario> conteiner) throws Exception {
42  }
43  });
44
45  for (Usuario usuario : usuarios) {
46      int id = usuario.getId();
47      List<Grupo> grupos = grupoRepositorio.consultarGrupos(id);
48      usuario.setGrupos(grupos);
49  }
50
51  return usuarios;
52 }
53 }
```

Writable | Smart Insert | 45:43:1324 | 295M of 590M

Proxy



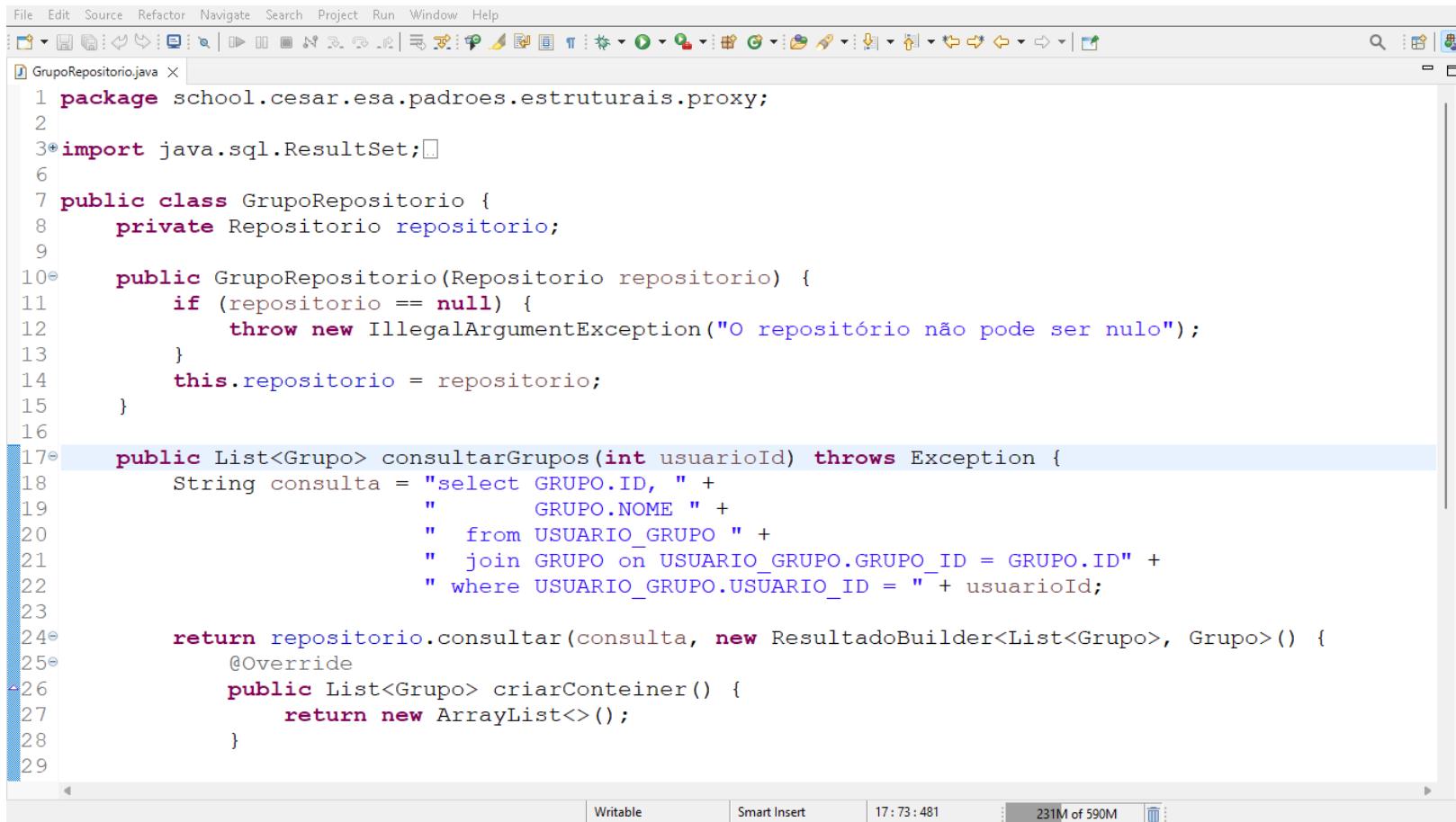
The screenshot shows a Java code editor window with the following details:

- File Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Includes various icons for file operations like Open, Save, Find, Copy, Paste, and others.
- Code Editor:** The file "Grupo.java" is open. The code defines a class "Grupo" with methods for setting and getting ID and name, and a constructor for setting both.

```
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 public class Grupo {
4     private Integer id;
5     private String nome;
6
7     public Grupo(String nome) {
8         setNome(nome);
9     }
10
11    public Grupo(int id, String nome) {
12        setId(id);
13        setNome(nome);
14    }
15
16    public void setId(int id) {
17        this.id = id;
18    }
19
20    public Integer getId() {
21        return id;
22    }
23
24    public void setNome(String nome) {
25        if (nome == null) {
26            throw new IllegalArgumentException("O nome não pode ser nulo");
27        }
28    }
29}
```

- Status Bar:** Writable Smart Insert 1:1:0 366M of 590M

Proxy



The screenshot shows a Java code editor window with the file `GrupoRepositorio.java` open. The code implements the Proxy pattern for managing groups.

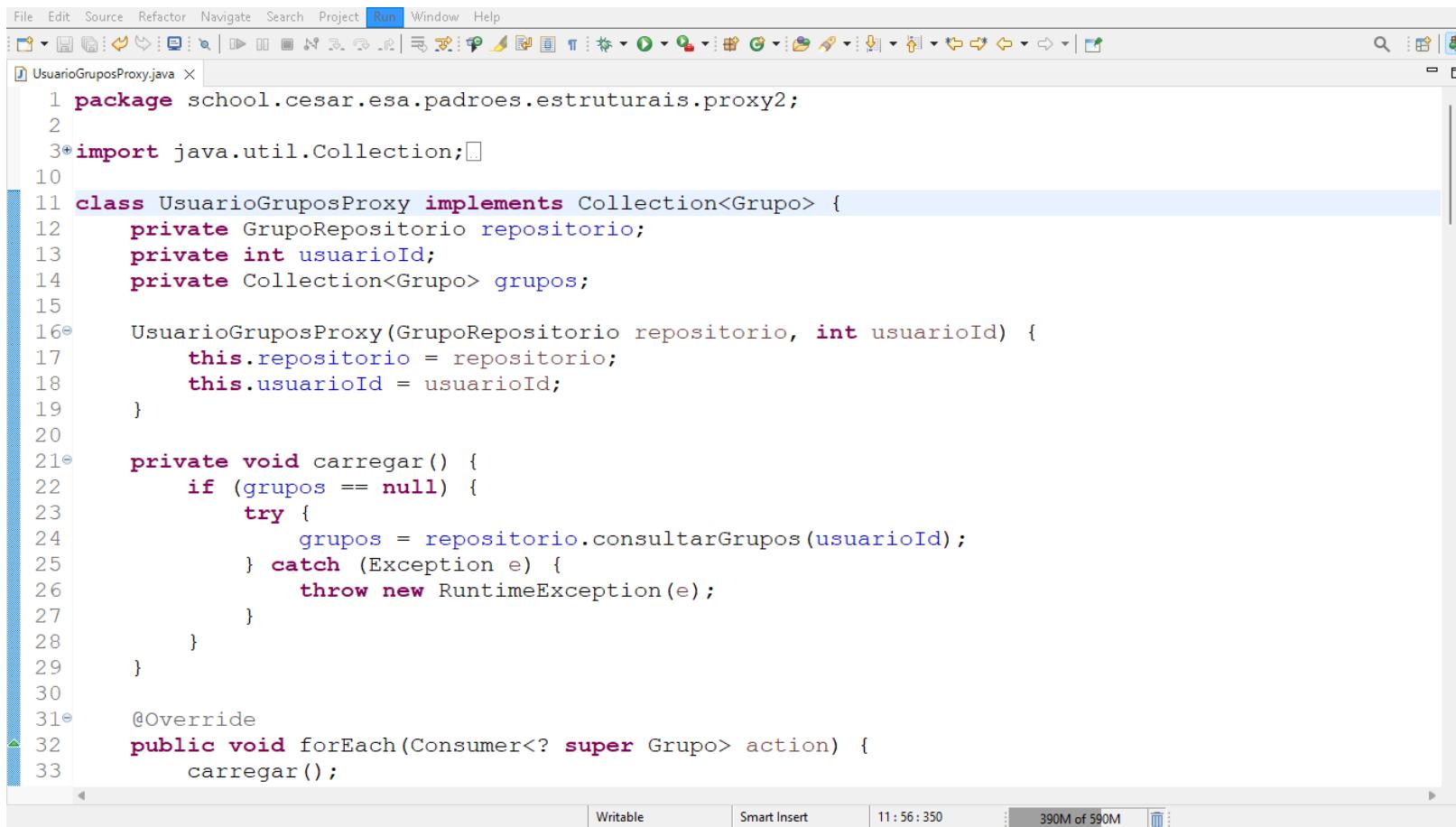
```
File Edit Source Refactor Navigate Search Project Run Window Help
J GrupoRepositorio.java X
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.sql.ResultSet;
4
5 public class GrupoRepositorio {
6     private Repositorio repositorio;
7
8     public GrupoRepositorio(Repositorio repositorio) {
9         if (repositorio == null) {
10            throw new IllegalArgumentException("O repositório não pode ser nulo");
11        }
12        this.repositorio = repositorio;
13    }
14
15    public List<Grupo> consultarGrupos(int usuarioId) throws Exception {
16        String consulta = "select GRUPO.ID, " +
17                     "      GRUPO.NOME " +
18                     "  from USUARIO_GRUPO " +
19                     " join GRUPO on USUARIO_GRUPO.GRUPO_ID = GRUPO.ID" +
20                     " where USUARIO_GRUPO.USUARIO_ID = " + usuarioId;
21
22        return repositorio.consultar(consulta, new ResultadoBuilder<List<Grupo>, Grupo>() {
23            @Override
24            public List<Grupo> criarContainer() {
25                return new ArrayList<>();
26            }
27        });
28    }
29}
```

The code defines a `GrupoRepositorio` class that takes a `Repositorio` object as a parameter in its constructor. It provides a `consultarGrupos` method that constructs a SQL query to retrieve groups for a given user ID and returns a list of `Grupo` objects using the `Repositorio`'s `consultar` method. The `consultar` method is implemented using a `ResultadoBuilder`.

Proxy



Proxy



The screenshot shows a Java code editor window with the file `UsuarioGruposProxy.java` open. The code implements the `Collection<Grupo>` interface, using a proxy pattern to interact with a repository. The code includes methods for initializing the proxy, loading groups for a user, and performing operations on the collection.

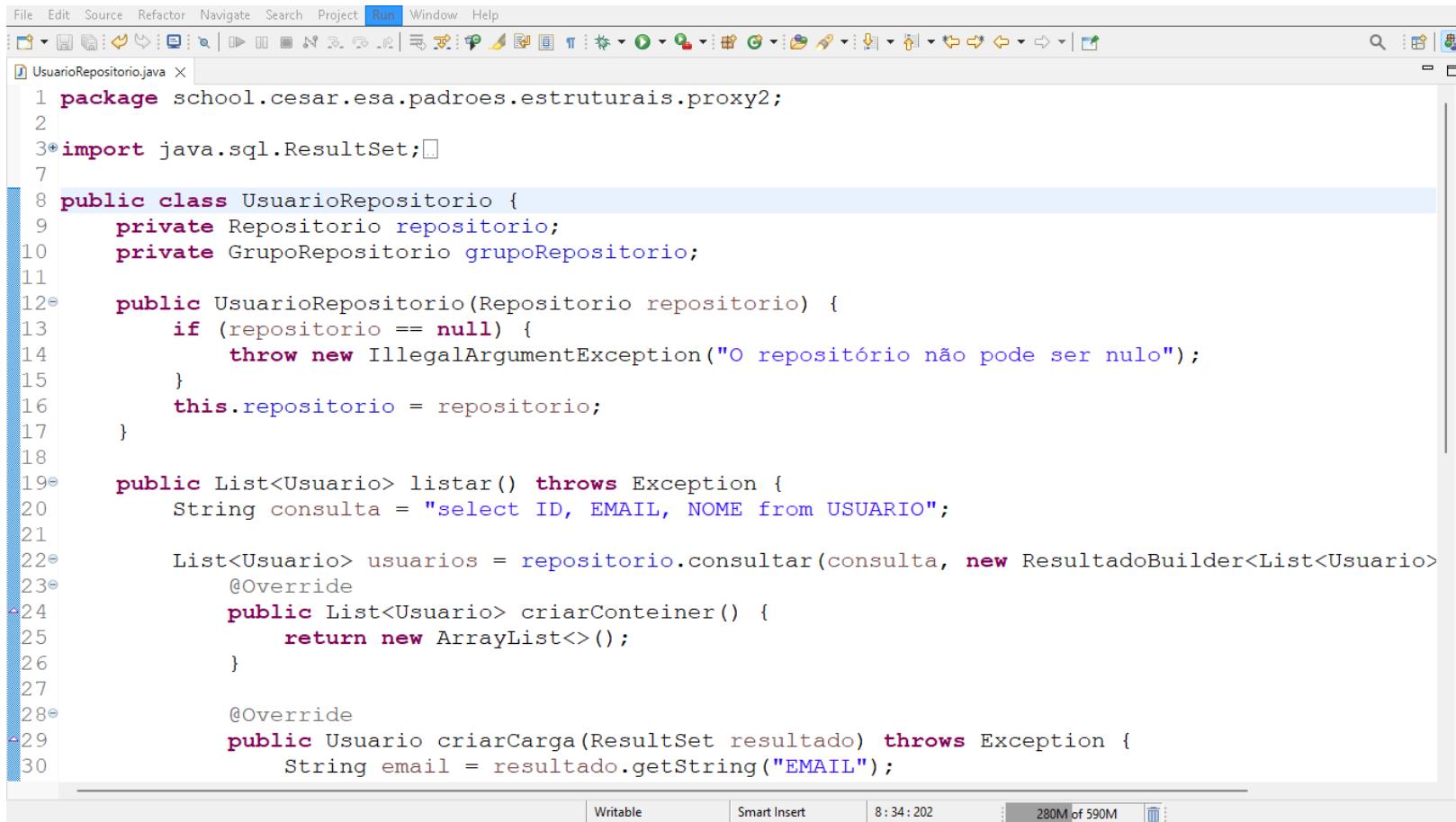
```
File Edit Source Refactor Navigate Search Project Run Window Help
User GruposProxy.java
1 package school.cesar.esa.padroes.estruturais.proxy;
2
3 import java.util.Collection;
4
5 class UsuarioGruposProxy implements Collection<Grupo> {
6     private GrupoRepositorio repositorio;
7     private int usuarioId;
8     private Collection<Grupo> grupos;
9
10    UsuarioGruposProxy(GrupoRepositorio repositorio, int usuarioId) {
11        this.repositorio = repositorio;
12        this.usuarioId = usuarioId;
13    }
14
15    private void carregar() {
16        if (grupos == null) {
17            try {
18                grupos = repositorio.consultarGrupos(usuarioId);
19            } catch (Exception e) {
20                throw new RuntimeException(e);
21            }
22        }
23    }
24
25    @Override
26    public void forEach(Consumer<? super Grupo> action) {
27        carregar();
28    }
29
30    @Override
31    public boolean add(Grupo grupo) {
32        carregar();
33        return false;
34    }
35
36    @Override
37    public boolean remove(Grupo grupo) {
38        carregar();
39        return false;
40    }
41
42    @Override
43    public boolean contains(Grupo grupo) {
44        carregar();
45        return false;
46    }
47
48    @Override
49    public int size() {
50        carregar();
51        return 0;
52    }
53
54    @Override
55    public boolean isEmpty() {
56        carregar();
57        return true;
58    }
59
60    @Override
61    public void clear() {
62        carregar();
63    }
64
65    @Override
66    public Grupo get(int index) {
67        carregar();
68        return null;
69    }
70
71    @Override
72    public Grupo[] toArray() {
73        carregar();
74        return new Grupo[0];
75    }
76
77    @Override
78    public int hashCode() {
79        carregar();
80        return 0;
81    }
82
83    @Override
84    public boolean equals(Object obj) {
85        carregar();
86        return false;
87    }
88
89    @Override
90    public Iterator<Grupo> iterator() {
91        carregar();
92        return null;
93    }
94
95    @Override
96    public void forEach(Consumer<? super Grupo> action) {
97        carregar();
98    }
99
100   @Override
101   public void removeIf(Predicate<? super Grupo> action) {
102       carregar();
103   }
104 }
```

Proxy

The screenshot shows a Java code editor with the file `UsuarioGruposProxy.java` open. The code implements a proxy pattern for managing user groups. It includes methods for iteration, size, emptiness, and containment, all delegating to a `grupos` collection.

```
File Edit Source Refactor Navigate Search Project Run Window Help
UserGruposProxy.java X
30
31    @Override
32    public void forEach(Consumer<? super Grupo> action) {
33        carregar();
34        grupos.forEach(action);
35    }
36
37    @Override
38    public int size() {
39        carregar();
40        return grupos.size();
41    }
42
43    @Override
44    public boolean isEmpty() {
45        carregar();
46        return grupos.isEmpty();
47    }
48
49    @Override
50    public boolean contains(Object o) {
51        carregar();
52        return grupos.contains(o);
53    }
54
55    @Override
56    public Iterator<Grupo> iterator() {
```

Proxy

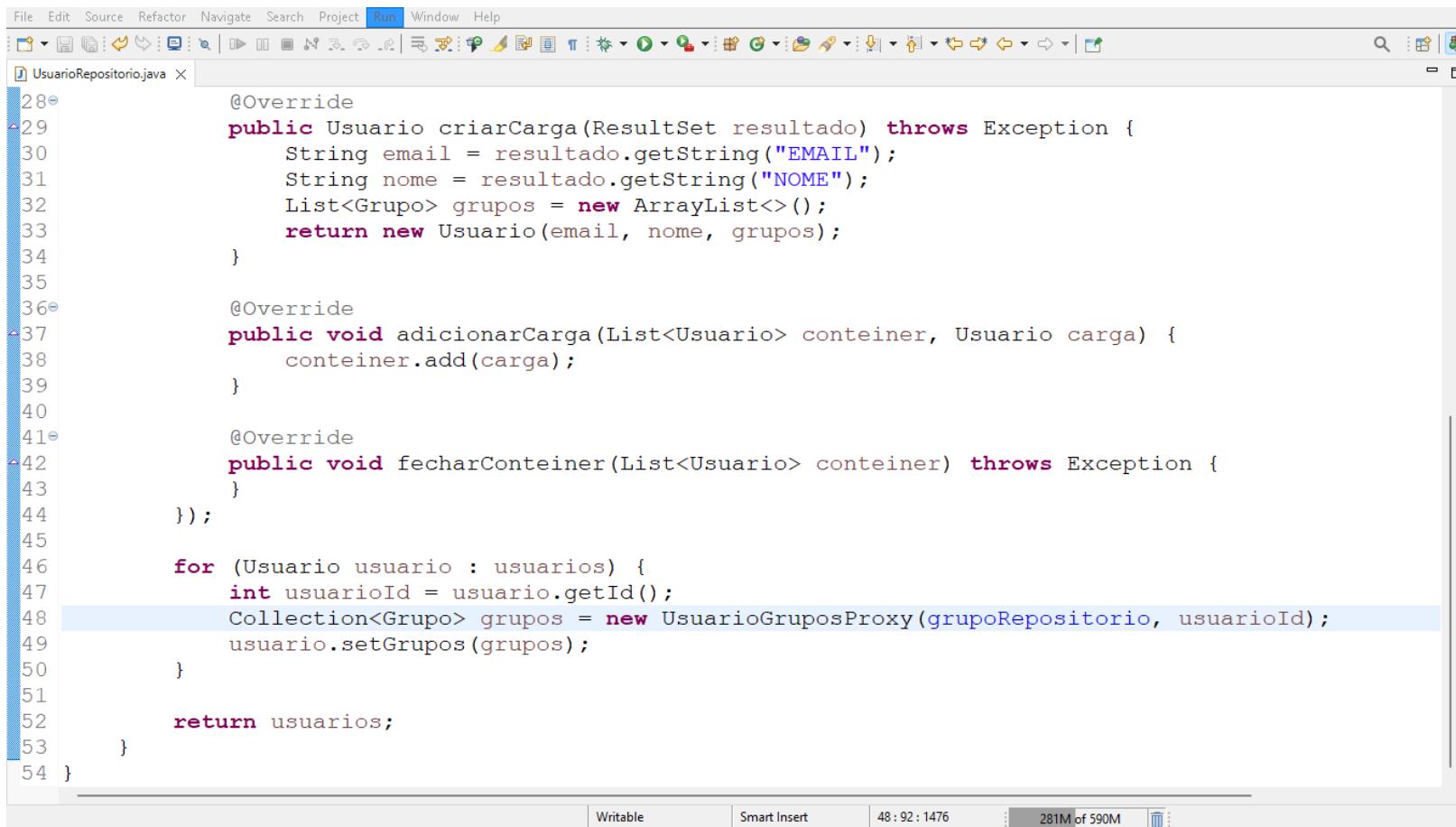


The screenshot shows a Java code editor window with the following details:

- File Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Code Editor:** The file `UsuarioRepositorio.java` is open. The code implements a proxy pattern for user repository operations.
- Code Content:**

```
1 package school.cesar.esa.padroes.estruturais.proxy2;
2
3 import java.sql.ResultSet;
4
5 public class UsuarioRepositorio {
6     private Repositorio repositorio;
7     private GrupoRepositorio grupoRepositorio;
8
9     public UsuarioRepositorio(Repositorio repositorio) {
10         if (repositorio == null) {
11             throw new IllegalArgumentException("O repositório não pode ser nulo");
12         }
13         this.repositorio = repositorio;
14     }
15
16     public List<Usuario> listar() throws Exception {
17         String consulta = "select ID, EMAIL, NOME from USUARIO";
18
19         List<Usuario> usuarios = repositorio.consultar(consulta, new ResultadoBuilder<List<Usuario>>();
20         @Override
21         public List<Usuario> criarContainer() {
22             return new ArrayList<>();
23         }
24
25         @Override
26         public Usuario criarCarga	ResultSet resultado) throws Exception {
27             String email = resultado.getString("EMAIL");
28         }
29     }
30 }
```
- Status Bar:** Writable, Smart Insert, 8:34:202, 280M of 590M.

Proxy



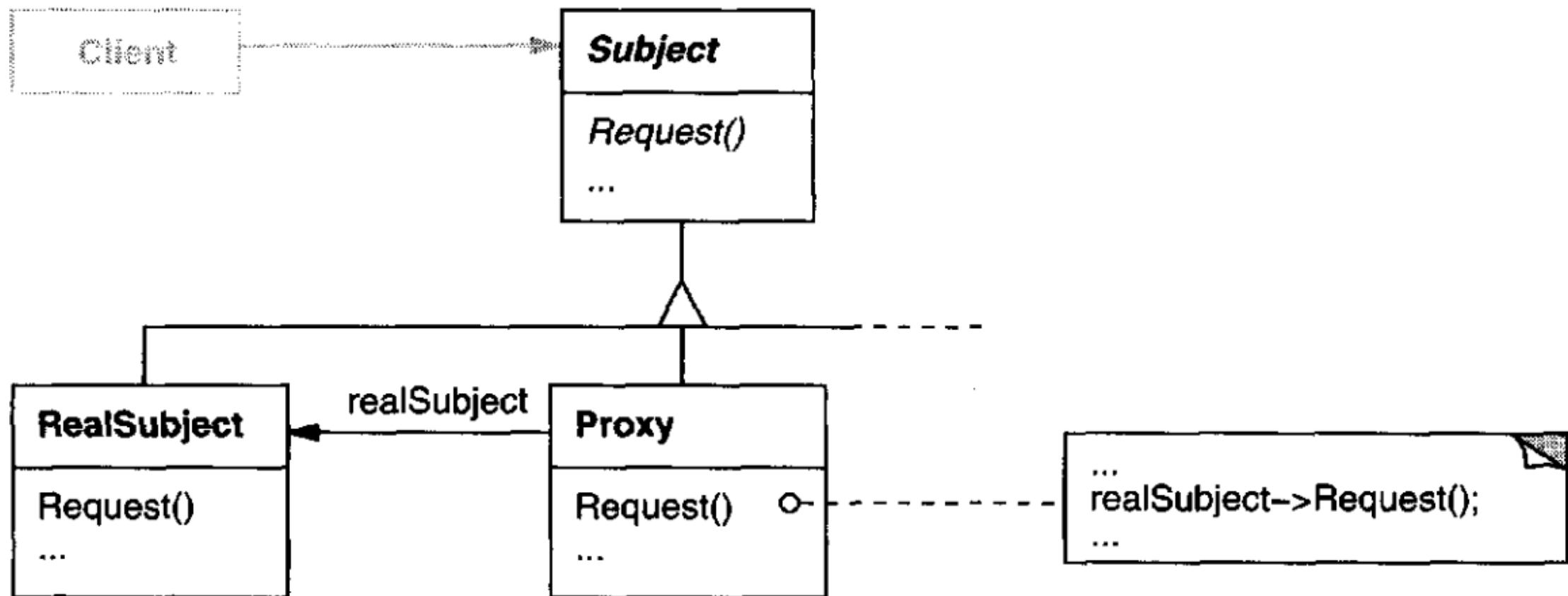
The screenshot shows a Java code editor with the file `UsuarioRepositorio.java` open. The code implements a repository interface for managing users. It includes methods to create a user from a database result set, add a user to a container, and close the container. The key part of the code is a loop that iterates through users and sets their groups using a proxy class. The proxy class is defined in the same file.

```
File Edit Source Refactor Navigate Search Project Run Window Help
UserRepository.java X
28     @Override
29     public Usuario criarCarga(ResultSet resultado) throws Exception {
30         String email = resultado.getString("EMAIL");
31         String nome = resultado.getString("NOME");
32         List<Grupo> grupos = new ArrayList<>();
33         return new Usuario(email, nome, grupos);
34     }
35
36     @Override
37     public void adicionarCarga(List<Usuario> conteiner, Usuario carga) {
38         conteiner.add(carga);
39     }
40
41     @Override
42     public void fecharConteiner(List<Usuario> conteiner) throws Exception {
43     }
44 );
45
46     for (Usuario usuario : usuarios) {
47         int usuarioId = usuario.getId();
48         Collection<Grupo> grupos = new UsuarioGruposProxy(grupoRepositorio, usuarioId);
49         usuario.setGrupos(grupos);
50     }
51
52     return usuarios;
53 }
54 }
```

Proxy

“Provide a surrogate or placeholder for another object to control access to it.”

Proxy



Proxy

- SOLID
 - Responsabilidade única (**S**ingle responsibility)
 - Aberto-fechado (**O**pen-closed)
 - Substituição de Liskov (**L**iskob substitution)
 - Segregação de interfaces (**I**nterface segregation)
 - Inversão de dependências (**D**evelopment dependency inversion)
- Prefira composição à herança
- Demeter

Proxy

- Integridade conceitual
- (Alta) Coesão
- (Baixo) Acoplamento
- Ocultamento de informações