EAD

EDUCAÇÃO A DISTÂNCIA

UNIDADE II

Aplicações de Linguagem de Programação Orientadas a Objetos

Prof. Me. Lauro Tomiatti

Atributos e métodos

- Quando trabalhamos com orientação a objetos, a concepção de uma estrutura de programação se torna mais documentada por sua própria definição.

- Quais suas características e o que é possível fazer se tornam peças de uma fração do projeto.

- Trabalhamos com as características e começaremos a trabalhar com as ações.

Eventos

- Para o funcionamento completo das GUIs, são necessários eventos; tarefas que são realizadas quando um usuário faz interação com algum componente da GUI ou quando alguma mudança é detectada.

- Os componentes da GUI sabem trabalhar convertendo suas alterações em eventos.

- Qualquer alteração do projeto que não advém da programação estrutural é um evento, a maior parte dos eventos é causada pelo usuário.

Eventos

- Quando a janela é fechada.
- Quando é pressionada uma tecla.
- Quando ocorre um clique ou movimentação do *mouse*.
- Quando ocorre alguma ação.

Eventos

- Origem do evento – componente GUI do qual o usuário interagiu.
- Objeto do evento – os dados do evento invocado por meio de um objeto.
- Ouvinte do evento – objeto notificado pela origem do evento quando ocorrido.

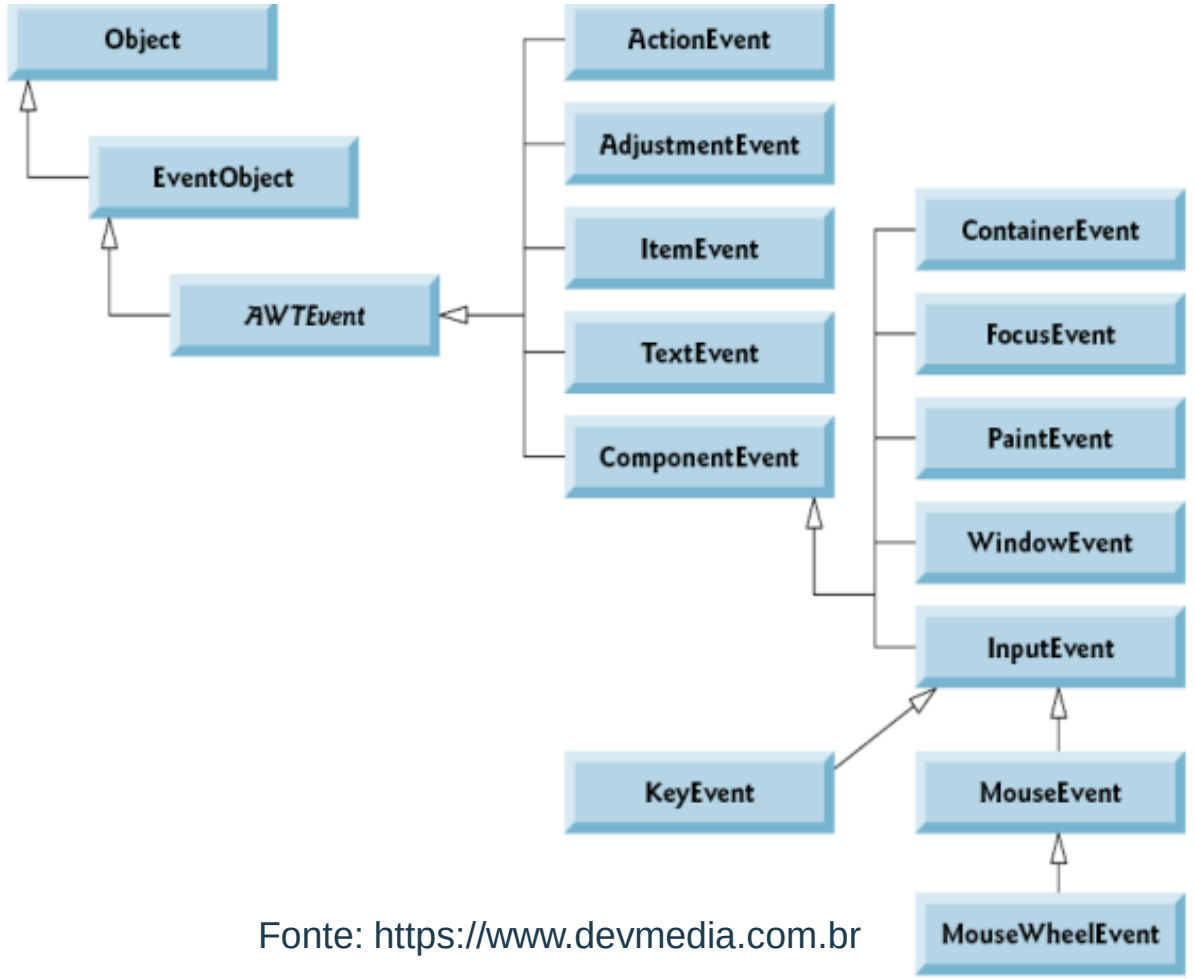
Eventos

- Quando um evento é causado, ele deve ser capturado.
- Cada evento é criado seguindo as regras da OO.
  - *WindowEvent*.
  - *KeyEvent*.
  - *MouseEvent*.
  - *ActionEvent*.

Fonte: Adaptado de: https://ceroshjacob.medium.com

Eventos

- A árvore de eventos é extensa, e conseguem detectar desde o foco até a troca ou adição de uma letra no componente.
- Essa estrutura está localizada na biblioteca AWT.



Fonte: https://www.devmedia.com.br

Eventos

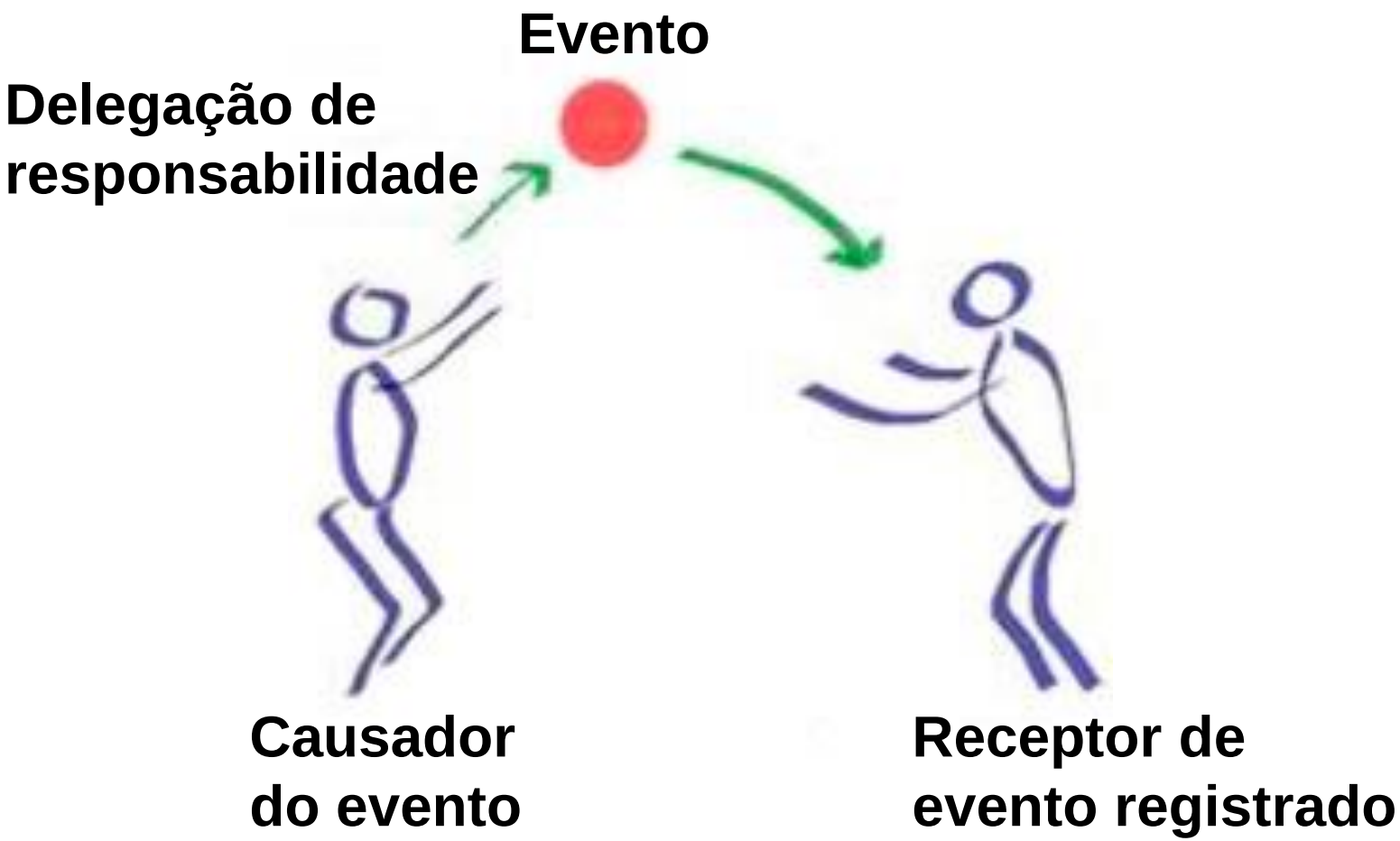
A utilização dos eventos pode ser implementada de duas formas:

- A classe que possui o evento implemente as interfaces relacionadas a quaisquer números de eventos que considere implementar.

- O componente origem do evento implemente a interface adicionando um ouvinte no componente controlado. Nesse caso, devemos implementar todos os métodos das interfaces de *Listener*.

Listeners

- Para detectar esse acontecimento, é necessário um *Listener.*
- Os *Listeners* são receptores desses eventos por meio de seus métodos.
- Os *Listeners* são interfaces.



Fonte: Adaptado de: https://ceroshjacob.medium.com

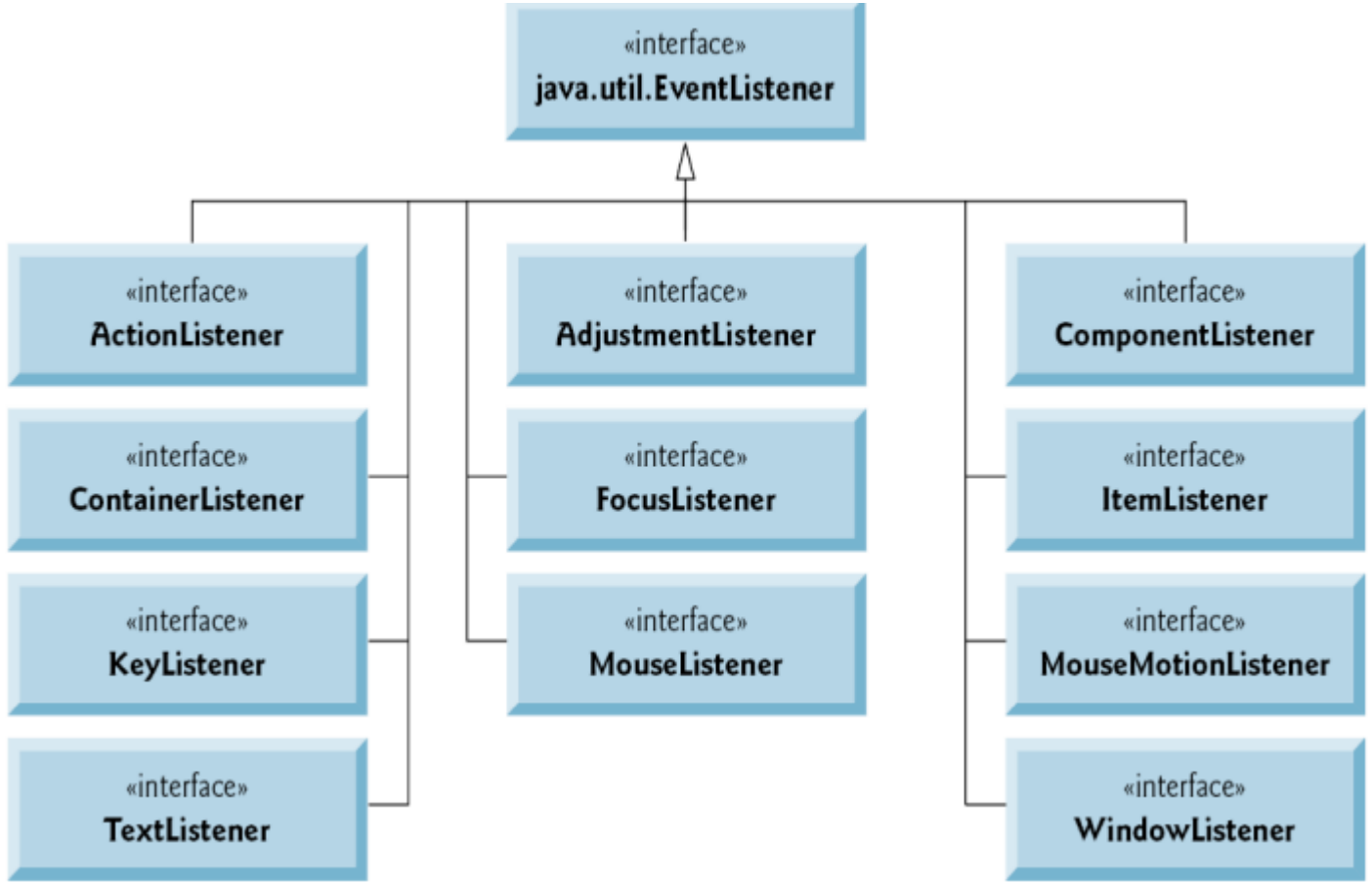
Listener

- Um exemplo de *Listener* é a interface *WindowListener* (ou ouvinte de janelas).

- Ela possui métodos abstratos capazes de detectar abertura de janela (windowOpened), fechamento de janela (windowClosing, windowClosed), quando a janela é dada ou retirada o foco (windowActivated, windowDeactivated) etc.

Árvore de *Listeners*

- A árvore de eventos é extensa, e conseguem detectar desde o foco até a troca ou adição de uma letra no componente.
- Essa estrutura está localizada na biblioteca AWT.



Fonte: https://www.devmedia.com.br

Interatividade

Dentro do conceito de interface, qual dos itens a seguir **não** se aplica?

a) As interfaces devem conter os métodos explicados das classes que ela herda.
b) As interfaces não devem implementar métodos.
c) As interfaces não podem ser instanciadas.
d) As interfaces dispõem do que chamamos de assinatura dos métodos.
e) Para implementar uma interface, utilizamos a palavra reservada *implements*.

Resposta

Dentro do conceito de interface, qual dos itens a seguir **não** se aplica?

a) As interfaces devem conter os métodos explicados das classes que ela herda.
b) As interfaces não devem implementar métodos.
c) As interfaces não podem ser instanciadas.
d) As interfaces dispõem do que chamamos de assinatura dos métodos.
e) Para implementar uma interface, utilizamos a palavra reservada *implements*.

Bom uso de interfaces e implementações

- “Qualquer pessoa pode escrever código que um computador pode entender, bons programadores escrevem código que humanos podem entender.” (Martin Fowler, 1999)

- Independente do uso, mantenha a consistência no código.

Implementação

- Mas as interfaces necessitam de implementação.

- Existem algumas maneiras de se trabalhar implementando essas interfaces.

- O importante é criarmos um bloco de instrução que siga as regras desejadas para o evento. Também é importante notar que, se a alteração for dos componentes, é importante relembrar do método repaint().

Criando uma classe

- Nesse caso, temos uma implementação própria feita em uma classe nova para cuidar das especificações do componente.
- Mais compreensível.

```
public class OuvinteDaJanelaDeCadastro implements WindowListener {

    @Override
    public void windowOpened(WindowEvent e) {
        System.out.println("A Janela abriu!");

    }

    @Override
    public void windowClosing(WindowEvent e) {
```

```
public class Janela extends JFrame {

    public Janela() {
        addWindowListener(new OuvinteDaJanelaDeCadastro());
    }
```

Fonte: Autoria própria.

Utilizando o próprio componente

- Nesse cenário, podemos fazer a própria janela implementar seus métodos, da mesma maneira, ela precisa adicionar à si própria como Listener.
- Cria menos classes para o projeto permanecer limpo.

```
public class Janela extends JFrame implements WindowListener {

    public Janela() {
        addWindowListener(this);
    }

    @Override
    public void windowOpened(WindowEvent e) {
        System.out.println("A Janela abriu!");
    }

    @Override
    public void windowClosing(WindowEvent e) {
```

Fonte: Autoria própria.

Instanciar uma classe anônima

- Podemos instanciar uma das classes que já implementam esses Listeners e modificar seus métodos, os *adapters*.
- Cria menos classes para o projeto permanecer limpo.

```
public class Janela extends JFrame {

    public Janela() {

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.out.println("A Janela abriu!");
            }
        });

    }
```

Fonte: Autoria própria.

Adapters

- Os adaptadores são classes providas pela biblioteca AWT e consistem na implementação vazia de interfaces *Listener.*

- Ela não necessita que todos os métodos sejam implementados, facilitando programas que devem trabalhar apenas com algumas interações.

O objeto evento

- O “e” passado como parâmetro para os métodos podem ser validados e possuem as características do evento para que saibamos o que deve ser feito ou utilizarmos métodos mais genéricos.

- Um exemplo disso é o ActionListener, que possui o método actionPerformed, que recebe um ActionEvent como parâmetro. É o único método desta interface. Sempre que um componente for ligado, esse método é acionado.

Interatividade

Os conceitos que permitem essa mutabilidade estão relacionados a um conceito de orientação a objetos específico, qual é esse conceito?

a) Herança.
b) Encapsulamento.
c) Polimorfismo.
d) Instância.
e) Paradigma de linguagem.

Resposta

Os conceitos que permitem essa mutabilidade estão relacionados a um conceito de orientação a objetos específico, qual é esse conceito?

a) Herança.
b) Encapsulamento.
c) Polimorfismo.
d) Instância.
e) Paradigma de linguagem.

Carta Aberta aos Hobbistas

Para mim, a coisa mais crítica no mercado de hobby neste momento é a falta de bons cursos de *software*, livros e o próprio *software*. Sem bons *softwares* e um usuário que entenda de programação, um computador de hobby é desperdiçado. Será que *softwares* de qualidade serão escritos para o mercado de hobbistas?

- O mercado de tecnologia passou por diversos acontecimentos marcantes que surpreenderam até os mais céticos.

Bibliotecas de banco de dados

- Geralmente, cada fabricante de BD fornece uma biblioteca para facilitar a conexão com o banco de dados.

- Essa biblioteca se chama connector, é um driver.

- Para que o Java se conecte com o banco de dados, é necessário o driver para seu acesso. Eles, geralmente, são fornecidos gratuitamente como um arquivo .JAR.

JDBC

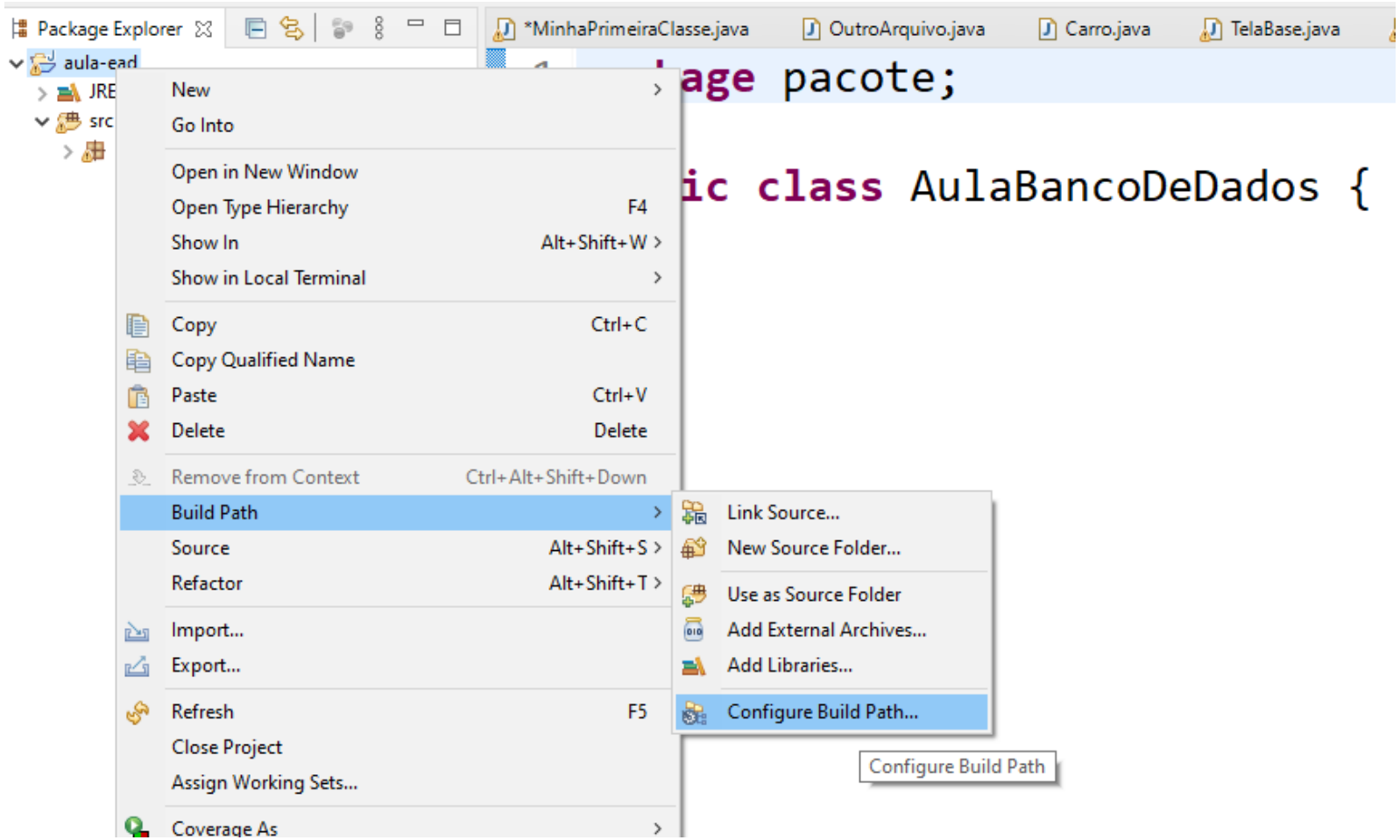
- Usaremos o JDBC (Java DataBase Connectivity) como nosso driver.
- É a biblioteca de persistência de dados relacionais do Java.
- Para que o Java se conecte com o banco de dados, é necessário o driver para seu acesso. Eles, geralmente, são fornecidos gratuitamente como um arquivo .JAR.

Classes principais

- DriverManager – gerencia o Driver (carregado pelo ClassLoader que será visto posteriormente).
- Connection – responsável pela conexão com o banco de dados.
- Statement (ou PreparedStatement) – nosso portal de acesso às queries (as nossas consultas e interação com o BD).
- ResultSet – objeto que guarda os dados de uma query do banco que ainda esteja aberta.

Adicionando o driver

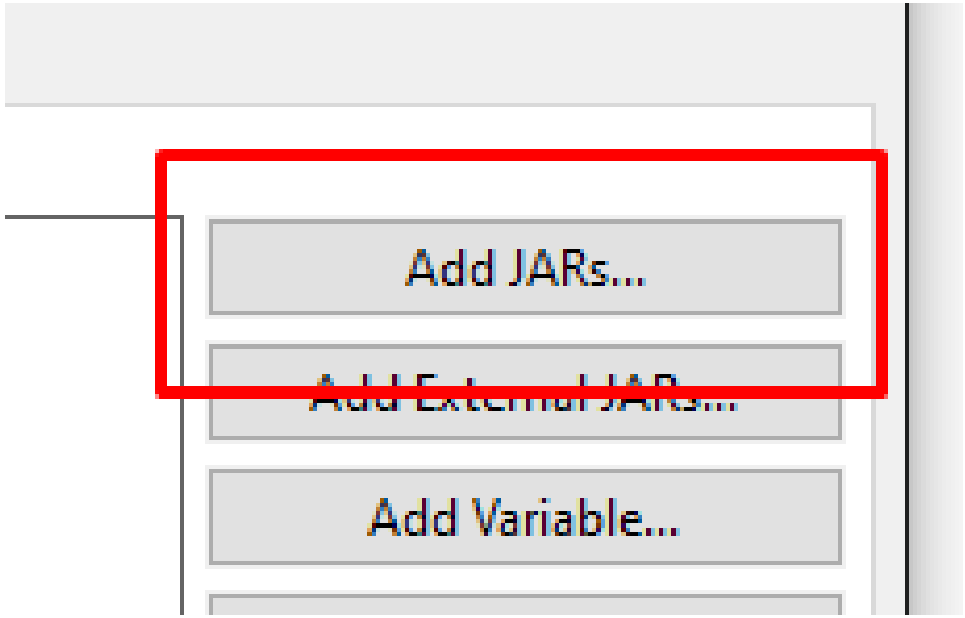
- Para adicionar, podemos ir por meio do Build Path se não possuirmos nenhum gerenciador de dependência.



Fonte: Autoria própria.

Adicionando o driver

- Após isso, clique em Add JARs e selecione o driver de sua preferência baixado.
- Ele precisa estar no formato .JAR.



Fonte: Autoria própria.

Gerenciador de dependências

- Facilitam para os desenvolvedores a administração de suas bibliotecas.
- Permite a atualização de uma biblioteca com apenas a troca de uma linha.
- A adição também é mais simples, e, uma vez na máquina, não é mais necessário baixá-la, apenas atualizar as linhas de código.

Iniciando o banco de dados no projeto

- Para começar, precisamos informar o banco de dados utilizado por meio do Class.forName, que é um método para informarmos o driver que vamos usar.

- A classe DriverManager possui métodos para procurar entre os drivers carregados o que esteja compatível.

```
Class.forName("com.mysql.jdbc.Driver");
```

Fonte: Autoria própria.

Iniciando o banco de dados no projeto

- DriverManager possui isso por meio do método getConnection, que necessita de uma URL.
- Ambas precisam de uma declaração de try catch.
- getConnection devolve um objeto Connection.

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/nomeDoBancoDeDados"
        + "?useTimezone=true&serverTimezone=UTC");
```

Fonte: Autoria própria.

Interatividade

Por que persistir os dados entre as execuções do programa?

a) Para garantir que o uso da memória seja menor.
b) Para melhorar o desempenho do programa, pois trabalha com menos objetos.
c) Para guardar dados de texto que variáveis não conseguem armazenar.
d) Para que outros usuários tenham acesso aos dados.
e) Para que os dados não se percam com o fechamento da janela.

Resposta

Por que persistir os dados entre as execuções do programa?

a) Para garantir que o uso da memória seja menor.
b) Para melhorar o desempenho do programa, pois trabalha com menos objetos.
c) Para guardar dados de texto que variáveis não conseguem armazenar.
d) Para que outros usuários tenham acesso aos dados.
e) Para que os dados não se percam com o fechamento da janela.

Aprendendo fundações

- “Antes que *software* possa ser reutilizável, primeiro ele deve ser usável.” (Ralph Johnson, 1994)
  A fundamentação teórica e suas complexidades são recompensados com uma facilidade na resolução dos problemas.

Trabalhando com banco de dados

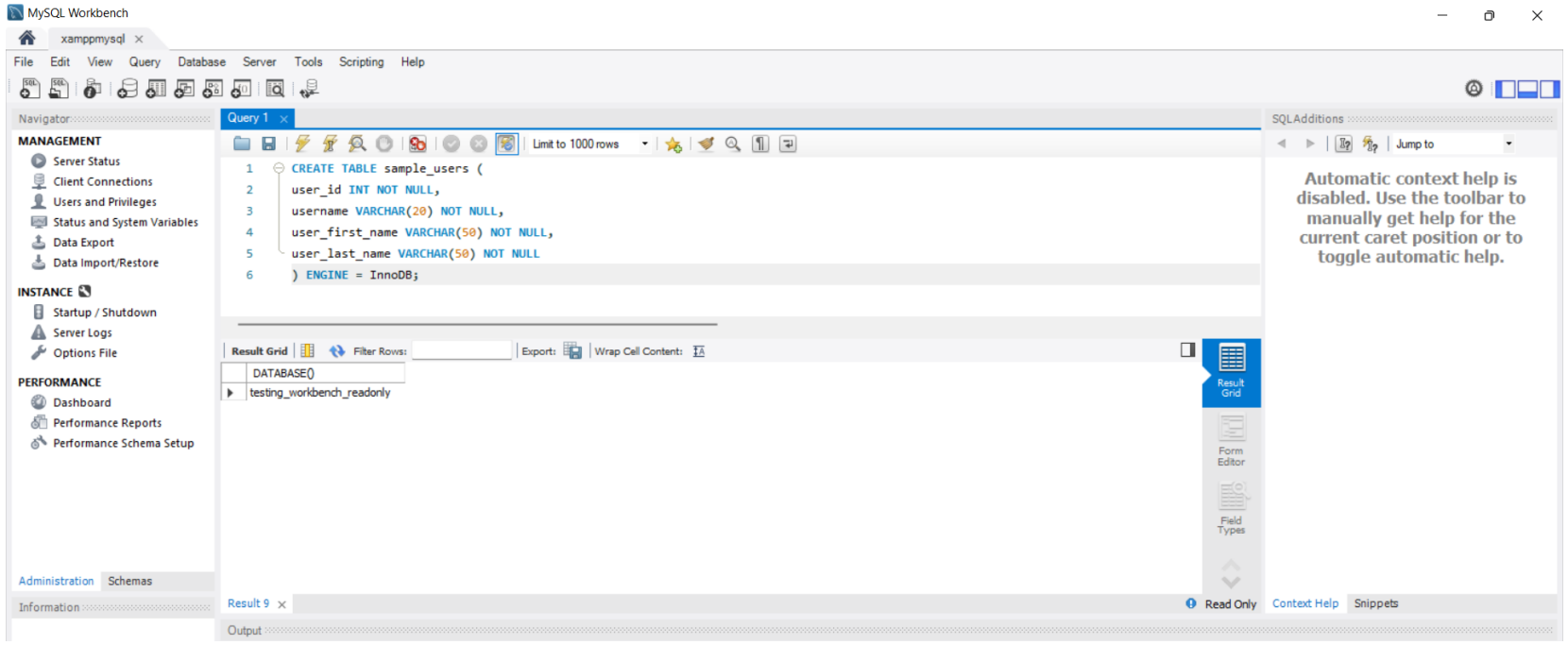
- O DriverManager devolve uma conexão em seu método getConnection.
- Essa conexão será o pilar principal de nossa utilização do banco de dados.

```
Connection conexao =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/nomeDoBancoDeDados"
        + "?useTimezone=true&serverTimezone=UTC");
```

Fonte: Autoria própria.

Conexão

- O banco de dados aberto pode ser acessado por uma conexão, assim que a conexão for estabelecida, temos a nosso dispor um processo semelhante ao de ambientes de trabalhos de banco de dados.



Fonte: www.delftstack.com

E se der errado?

- Podemos capturar a origem do erro pelo tipo de exceção que é enviado como parâmetro, devemos lembrar que isso utiliza o conceito de polimorfismo em relação às exceptions.

```
    |
}catch(ClassNotFoundException ex){
    System.out.println("JDBC não encontrado");
}catch(SQLException ex){
    System.out.println("Problemas de conexão");
}catch(Exception ex){
    System.out.println("Alguma outra exceção");
}
```

Fonte: Autoria própria.

Criação de um objeto para representar a entidade

- Em muitos casos, o cenário ideal é a criação de um objeto que seja a representação da tabela. Com isso, podemos trabalhar facilmente no manuseio, além de possuir uma representação em código no nosso programa que podemos consultar para saber quais campos existem.
- Níveis mais avançados de programação focada em persistência com acompanhamento de ferramentas específicas utilizam os objetos para executar suas responsabilidades.

```
public class Carro {

    private String marca;
    private String modelo;
    private String cor;
    private int ano;
    private double valor;

    // Setters e getters
```

Fonte: Autoria própria.

Criação de query

- Com a conexão aberta, podemos iniciar o processo de criação de uma query.
- Colocaremos ela em um Statement.

```
String query = "SELECT * FROM carros";

PreparedStatement statement = conexao.prepareStatement(query);
```

Fonte: Autoria própria.

Statement ou PreparedStatement

- SQLInjection.
- Necessidade de atribuições por métodos.

```
String marca = "null; DROP DATABASE carros";

String query = "SELECT * FROM carros WHERE marca = " + marca;
```

Fonte: Autoria própria.

ResultSet

- Com isso, podemos executar nossa query e receber do método um ResultSet.
- ResultSet trabalha com ponteiros.

```
String query = "SELECT * FROM carros";

PreparedStatement statement = conexao.prepareStatement(query);

ResultSet colecao = statement.executeQuery();
```

Fonte: Autoria própria.

ResultSet

- Percorremos utilizando o método next() sempre que quisermos trocar de objeto.
- O método next() foi designado pensando na interação com o laço de repetição while.

```
while(colecao.next()) {
    Carro novoCarro = new Carro();
    novoCarro.setMarca(colecao.getString("marca"));
    novoCarro.setModelo(colecao.getString("modelo"));
    novoCarro.setAno(colecao.getInt("ano"));
    novoCarro.setCor(colecao.getString("cor"));
    novoCarro.setValor(colecao.getDouble("valor"));
    System.out.println(novoCarro.getMarca() + " " + novoCarro.getModelo());
    System.out.println("Ano: " + novoCarro.getAno() + " Cor: " + novoCarro.getCor());
    System.out.println("Valor: " + novoCarro.getValor());
}
```

Fonte: Autoria própria.

Interatividade

Qual conceito de orientação a objetos estamos aplicando com o uso do preenchimento do objeto por meio de Getters e Setters?

a) Herança.
b) Encapsulamento.
c) Polimorfismo.
d) Instância.
e) Atributos.

Resposta

Qual conceito de orientação a objetos estamos aplicando com o uso do preenchimento do objeto por meio de Getters e Setters?

a) Herança.
b) Encapsulamento.
c) Polimorfismo.
d) Instância.
e) Atributos.

Referências

- FOWLER, Martin. *Refactoring:* Improving the Design of Existing Code. Boston: Addison-Wesley*,*1999.
- JOHNSON, Ralph *et al*. *Design Patterns*: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley*,* 1994.
- GATES, Bill. *Open Letter to Hobbyists*. *Homebrew Computer Club Newsletter Volume 2*, 1976.

ATÉ A PRÓXIMA!