Qual das opções a seguir pode preencher os espaços em branco para compilar esse código?

```
__________ a = __________.getConnection(
url, userName, password);
__________ b = a.prepareStatement(sql);
__________ c = b.executeQuery();
if (c.next()) System.out.println(c.getString(1));
```

A. Connection, Driver, PreparedStatement, ResultSet
B. Connection, DriverManager, PreparedStatement, ResultSet
C. Connection, DataSource, PreparedStatement, ResultSet
D. Driver, Connection, PreparedStatement, ResultSet
E. DriverManager, Connection, PreparedStatement, ResultSet
F. DataSource, Connection, PreparedStatement, ResultSet

ANÁLISE DA QUESTÃO

Verificando cada alternativa se está correta ou não:

Alternativa A:
Connection, Driver, PreparedStatement, ResultSet

```
Connection a = Driver.getConnection(url, userName, password);
```

- Driver **não tem um método getConnection estático**. Geralmente, Driver é registrado no DriverManager, que é então usado para obter uma conexão.

```
PreparedStatement b = a.prepareStatement(sql);
ResultSet c = b.executeQuery();
```

Resultado: Incorreta

Alternativa B:
Connection, DriverManager, PreparedStatement, ResultSet

```
Connection a = DriverManager.getConnection(url, userName, password);
```

- DriverManager é a classe correta para obter uma conexão de banco de dados.

```
PreparedStatement b = a.prepareStatement(sql);
ResultSet c = b.executeQuery();
```

Resultado: Correta

Alternativa C:

Connection, DataSource, PreparedStatement, ResultSet

```
Connection a = DataSource.getConnection(url, userName, password);
```

- DataSource é uma interface que geralmente é injetada ou buscada em um contêiner, e não tem um método estático getConnection.

```
PreparedStatement b = a.prepareStatement(sql);
ResultSet c = b.executeQuery();
```

Resultado: Incorreta

Alternativa D:

Driver, Connection, PreparedStatement, ResultSet

```
Driver a Connection.getConnection(url, userName, password);
```

- Connection não tem um método estático getConnection, e Driver não é uma interface correta aqui.

```
PreparedStatement b = a.prepareStatement(sql);
ResultSet c = b.executeQuery();
```

Resultado: Incorreta

Alternativa E:

DriverManager, Connection, PreparedStatement, ResultSet

```
DriverManager a = Connection.getConnection(url, userName, password);
```

- Connection.getConnection não é um método estático válido e DriverManager deve ser usado para obter uma conexão.

```
PreparedStatement b = a.prepareStatement(sql);
ResultSet c = b.executeQuery();
```

Resultado: Incorreta

Alternativa F:

DataSource, Connection, PreparedStatement, ResultSet

```
DataSource a = Connection.getConnection(url, userName, password);
```

- Connection.getConnection não é um método estático válido e DataSource é uma interface que fornece uma conexão, não obtém uma.

```
PreparedStatement b = a.prepareStatement(sql);
ResultSet c = b.executeQuery();

Resultado: Incorreta
```

Resumo:

A única alternativa correta é a **Alternativa B: Connection, DriverManager, PreparedStatement, ResultSet.**

Vamos mergulhar nesse mundo das interfaces Java usadas para trabalhar com bancos de dados. Vou explicar cada uma delas e destacar seus principais métodos.

1. Connection

Interface: java.sql.Connection

Descrição: Representa uma conexão com um banco de dados. Você usa um objeto *Connection* para se comunicar com o banco de dados.

Principais Métodos:

- *createStatement():* Cria um objeto Statement para enviar instruções SQL ao banco de dados.
- *prepareStatement(String sql):* Cria um objeto PreparedStatement para instruções SQL pré-compiladas com parâmetros.
- *commit():* Confirma (commit) todas as mudanças feitas desde o último commit/rollback.
- rollback(): Desfaz (rollback) todas as mudanças feitas desde o último commit/rollback.
- *close():* Fecha a conexão e libera todos os recursos associados.

2. Driver

Interface: java.sql.Driver

Descrição: Um driver é responsável por estabelecer uma conexão com o banco de dados. A interface Driver deve ser implementada por qualquer driver JDBC.

Principais Métodos:

- *connect(String url, Properties info):* Tenta fazer uma conexão com o banco de dados dado um URL e propriedades de conexão.
- *acceptsURL(String url):* Verifica se o driver pode se conectar ao banco de dados especificado pelo URL.
- *getPropertyInfo(String url, Properties info):* Obtém informações sobre as propriedades de conexão suportadas pelo driver.

3. PreparedStatement

Interface: java.sql.PreparedStatement

Descrição: Representa uma instrução SQL pré-compilada. Permite a execução eficiente de instruções SQL repetidas vezes com parâmetros diferentes.

Principais Métodos:

- *setInt(int parameterIndex, int x):* Define um valor inteiro para o parâmetro especificado.

- *setString(int parameterIndex, String x):* Define uma string para o parâmetro especificado.
- *executeQuery():* Executa a instrução SQL de consulta e retorna um objeto ResultSet.
- *executeUpdate():* Executa uma instrução SQL de atualização (INSERT, UPDATE ou DELETE) e retorna o número de linhas afetadas.
- *clearParameters():* Limpa os parâmetros atuais da instrução.

4. ResultSet

Interface: java.sql.ResultSet

Descrição: Representa um conjunto de resultados de uma consulta ao banco de dados. Você usa um objeto ResultSet para iterar pelos resultados.

Principais Métodos:

- *next():* Move o cursor para a próxima linha do conjunto de resultados.
- *getInt(String columnLabel):* Obtém o valor inteiro da coluna especificada pelo rótulo.
- *getString(String columnLabel):* Obtém o valor string da coluna especificada pelo rótulo.
- *close():* Fecha o ResultSet e libera os recursos associados.

5. DataSource

Interface: javax.sql.DataSource

Descrição: Um DataSource é uma alternativa mais flexível para obter conexões com bancos de dados. Pode ser configurado para fornecer pooling de conexões e gerenciamento eficiente.

Principais Métodos:

- *getConnection():* Obtém uma conexão com o banco de dados.
- *getConnection(String username, String password):* Obtém uma conexão com o banco de dados usando credenciais específicas.
- *setLogWriter(PrintWriter out):* Define um escritor de log para registrar mensagens de log.

6. DriverManager

Classe: java.sql.DriverManager

Descrição: Gerencia uma lista de drivers de banco de dados e estabelece a conexão com um banco de dados.

Principais Métodos:

- *static getConnection(String url):* Estabelece uma conexão com o banco de dados especificado pelo URL.
- *static getConnection(String url, Properties info):* Estabelece uma conexão usando um URL e propriedades de conexão.

- *static getConnection(String url, String user, String password):* Estabelece uma conexão usando um URL, nome de usuário e senha.
- *static registerDriver(Driver driver):* Registra um novo driver com o DriverManager.

Todos os métodos principais da classe *DriverManager* são estáticos. Isso faz sentido porque DriverManager é usado para gerenciar drivers e conexões a nível global, e não em instâncias específicas.

Essas interfaces e classes são fundamentais para trabalhar com JDBC (Java Database Connectivity), permitindo a comunicação entre a aplicação Java e um banco de dados.