**Teoria:**

1. Defina:
   1. Sobrecarga de métodos (OVERLOADING)
   2. Sobrescrita de métodos (OVERRIDE)
   3. Métodos abstratos
2. Explique a função de cada palavra reservada em Java:
   1. **interface**
   2. **static**
   3. **implements**
   4. **extends**
   5. **throw**
   6. **throws**
3. Analise o código abaixo, marque a **alternativa correta** e **justifique sua resposta**:

**class** A{

**public** **final** **void** print() {

System.***out***.println("Hello, A!");

}

}

**class** B **extends** A{

@Override

**public** **void** print() {

System.***out***.println("Hello, B!");

}

}

**class** Test{

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

B b = **new** B();

b.print();

}

}

* + O código executa com sucesso e imprime “Hello, A!”
  + O código executa com sucesso e imprime “Hello, B!”
  + O código não compila
  + O código compila, mas ao executar lança exceção

|  |  |
| --- | --- |
| **Resposta:** |  |
| **Justificativa:** |  |

1. Analise o código abaixo, marque a **alternativa correta** e **justifique sua resposta**:

**final** **class** A{

**public** **final** **void** print() {

System.***out***.println("Hello, A!");

}

}

**class** B **extends** A{

}

**class** Test{

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

B b = **new** B();

b.print();

}

}

* + O código executa com sucesso e imprime “Hello, A!”
  + O código não compila
  + O código compila, mas ao executar lança exceção

|  |  |
| --- | --- |
| **Resposta:** |  |
| **Justificativa:** |  |

1. Analise o código abaixo, marque a **alternativa correta** e **justifique sua resposta**:

**final** **class** A{

**public** **void** print() **throws** Exception {

System.***out***.println("Hello, A!");

}

}

**class** Test{

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

A a = **new** A();

a.print();

}

}

* + O código executa com sucesso e imprime “Hello, A!”
  + O código não compila
  + O código compila, mas ao executar lança exceção

|  |  |
| --- | --- |
| **Resposta:** |  |
| **Justificativa:** |  |

1. Analise o código abaixo, marque a **alternativa correta** e **justifique sua resposta**:

**class** A{

**public** **static** **void** print() {

System.***out***.println("Hello, A!");

}

}

**class** Test{

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

A a = **new** A();

a.*print*();

}

}

* + O código executa com sucesso e imprime “Hello, A!”
  + O código não compila
  + O código compila, mas ao executar lança exceção

|  |  |
| --- | --- |
| **Resposta:** |  |
| **Justificativa:** |  |

1. Analise o código abaixo, marque a **alternativa correta** e **justifique sua resposta**:

**interface** IA{

**void** print();

}

**class** A **implements** IA{

@Override

**public** **void** print() {

System.***out***.println("Hello, A!");

}

}

**class** Test{

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

A a = **new** A();

a.print();

}

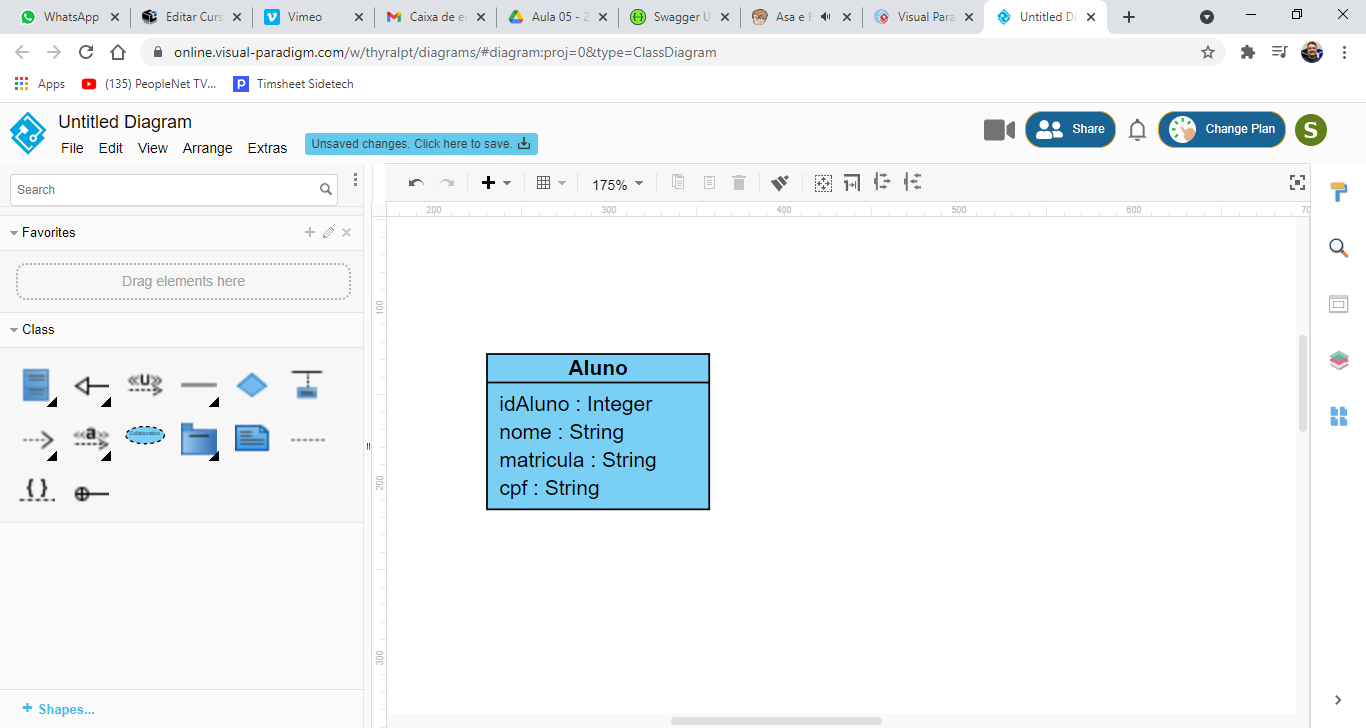
}

* + O código executa com sucesso e imprime “Hello, A!”
  + O código não compila
  + O código compila, mas ao executar lança exceção

|  |  |
| --- | --- |
| **Resposta:** |  |
| **Justificativa:** |  |

**Prática:**

1. Crie um projeto do Java:  
   Neste projeto, desenvolva uma entidade “Aluno”, conforme abaixo:
   1. Utilize o padrão JavaBean
      1. Atributos privados
      2. Sobrecarga de construtores
      3. Métodos de encapsulamento set e get



1. Crie um banco de dados no PostGreSQL e desenvolva neste banco de dados uma tabela “Aluno” com os mesmos campos da classe de entidade.
2. Utilize o Pattern Factory Method para gerar a conexão com o banco de dados.
3. Crie um repositório “AlunoRepository” contendo métodos para inserir, alterar, excluir e consultar todos e consultar por id na base de dados (utilize JDBC).
4. Crie um controlador para cadastrar, atualizar, excluir e consultar produtos e, através da Classe Program.java, demonstre a execução.