JAVA BÁSICO

LABORATÓRIO

1. Abra o **Eclipse** e selecione para “File > New > Java Project”; em **Project Name** coloque “Lab.Polimorfismo” e então confirme.

2. Busque a **pasta** com o nome “src” no lado esquerdo no **Package Explorer**

   (a) Clique com o botão direito na pasta “src” e selecione “New > Interface”

   (b) No campo **Package** coloque “br.com.universidade”

   (c) No campo nome digite “LoginCreatorInterface” e confirme pressionando **Finish**

3. Na interface recém-criada “LoginCreatorInterface” adicione o seguinte método no corpo dessa classe

```
public interface LoginCreatorInterface {

      String Generate(String nome, String sobrenome);
}
```

4. Com essa interface definimos um **contrato** de como gerar um login a partir de um nome e de um sobrenome

5. Em primeiro lugar definiremos uma classe que implementará essa interface chamada “SobrenomeNomeLoginCreator”

6. Busque a **pasta** com o nome “src” no lado esquerdo no **Package Explorer**

(a) Clique com o botão direito na pasta “src” e selecione “New > Class”

(b) No campo **Package** coloque “br.com.universidade”

7. No campo nome digite “SobrenomeNomeLoginCreator” e confirme pressionando **Finish**

8. Com isso agora devemos implementar a interface LoginCreatorInterface

```
public class SobrenomeNomeLoginCreator implements LoginCreatorInterface {


}
```

9. Para de fato implementar a classe temos de produzir uma implementação de cada um dos métodos da interface

(a) A classe “LoginCreatorInterface” tem um método chamado “Generate” que tem de ser sobrescrito

(b) Para sobrescrever este método utilizando a implementação abaixo

```
    @Override
    public String Generate(String nome, String sobrenome) {

          return sobrenome + nome.charAt(0);
    }
```

10. Devemos agora criar uma classe **Main** contendo o método **main** que deve ter como corpo o seguinte código

```
public static void main(String[] args)
{
      LoginCreatorInterface creator = new SobrenomeNomeLoginCreator();
      String resultado = creator.Generate("Jose", "Silva");
      System.out.println(resultado);

}
```

(a) O resultado da execução deve produzir “SilvaJ”

11. Até este momento não houve vantagem alguma de utilizar polimorfismo.
12. Para melhor entender a importância desta técnica criaremos outra classe geradora de logins, chamada de “NomeSepSobrenomeLoginCreator”

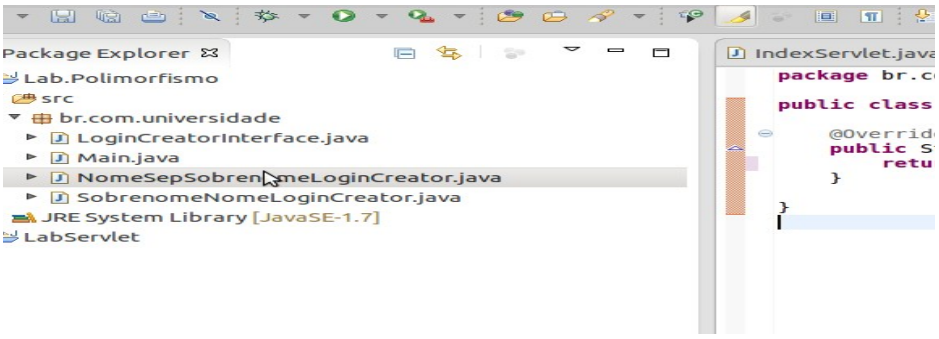
(a) Esta classe também implementará a interface “LoginCreatorInterface”

(b) A implementação desta interface, entretanto, gerará o login da seguinte forma:

```
@Override

public String Generate(String nome, String sobrenome) {
      return nome.toLowerCase() + "." + sobrenome.toLowerCase();
}
```

13. Com essa classe gerada, teremos até então os seguintes arquivos



14. Agora utilizaremos um padrão de projeto denominado de **Factory** para criar objetos do tipo LoginCreator

(a) Este padrão permite delegar a geração do objeto para uma classe específica, que pode determinar qual tipo de objeto criar de acordo com características do objeto

```
public class LoginCreatorFactory {


	public LoginCreatorInterface construct()
	{
		/**
		 * Este dado seria lido do arquivo de configuração por exemplo
		 * configurado de acordo com a universidade
		 */
		String tipoLogin = "nome-sep-sobrenome";

		if (tipoLogin.equals("sobrenome-nome"))
		{
			return new SobrenomeNomeLoginCreator();
		}
		else if (tipoLogin.equals("nome-sep-sobrenome"))
		{
			return new NomeSepSobrenomeLoginCreator();
		}
		else
		{
			throw new RuntimeException("LoginCreator "
					+ tipoLogin + " inválido");
		}
	}

}
```

O último passo agora é alterar o **main** para utilizar a nova estrutura

```
	public static void main(String[] args)

	{
		LoginCreatorFactory factory = new LoginCreatorFactory();
		LoginCreatorInterface loginCreator = factory.construct();

		String login = loginCreator.Generate("Jose", "Silva");
		System.out.println(login);
	}
```

DESAFIO: Como impedir que os usuários dessas classes criem os objetos como “SobrenomeNomeLoginCreator” diretamente?

DESAFIO: O padrão **Factory** permite que objetos relacionados sejam criados de forma uniforme e transparente. O que o padrão **Abstract Factory** poderia permitir?