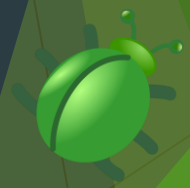
Orientação a Objetos para Testadores

@BrunoMurawski

Muitas vezes aquele que mais precisa é o que menos se empenha



Porque utilizar O.O.?

Validar CEP em vários Formulários

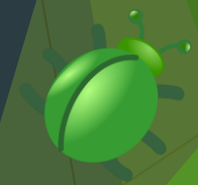
Qual a chance de esquecer de validar eles?

Vantagens

Não espalhar a responsabilidade de validar em todo o seu código!

Centralizar essa responsabilidade em um só lugar!

Diminui as chances de erros!

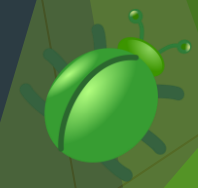


Vantagens

- Organiza o código
- Diminui o Retrabalho (código duplicado e correções)
- Encapsula a lógica dos Negócios
- Responsabilidades centralizadas

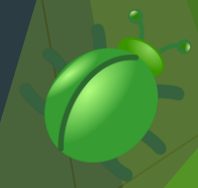
Vantagens

- Diminui código escrito – c/ o Polimorfismo



O que é um Objeto?

- Características
- Comportamento
- Estado



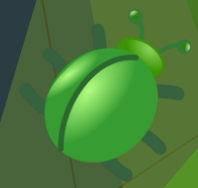
Estado

É como ele está em um determinado momento

Ex: o objeto televisão está ligado ou desligado?

Características

Características dizem o que o objeto é

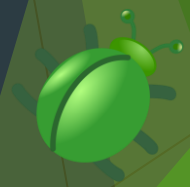


Precisamos saber disso?

Precisamos para que possamos fazer a abstração dos objetos que iremos trabalhar

Abstração de Objetos

Abstração de Dados

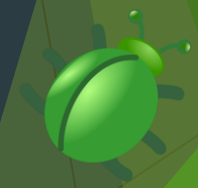


Pensando em Pessoas

Objeto: pessoa

Contexto: comprar um produto em um e-commerce

O que toda pessoa tem?



O que toda pessoa tem?

Características Físicas:

- Cabeça
- Olhos
- Boca
- Braços
- Mãos
- ...

O que toda pessoa tem?

Características Sociais:

- Nome
- Data de Nascimento
- Sexo
- CPF
- ...

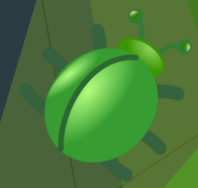
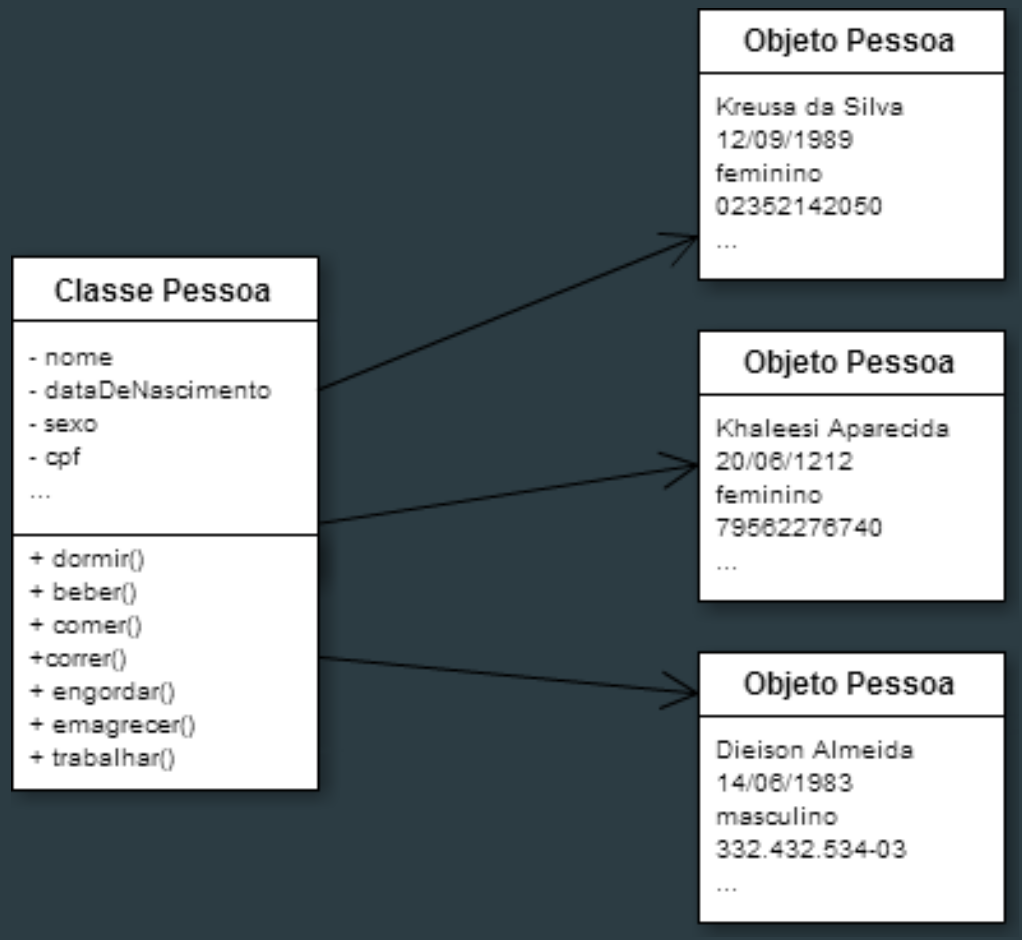


Diagrama de Classe



Instâncias

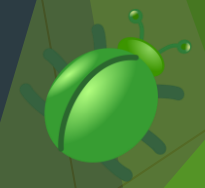
Criamos “instâncias” de pessoas

Podem ter ações e comportamentos

Declaramos o que cada pessoa tem no projeto(classe), mas as instâncias é que podem armazenar esses dados realmente!

Projeto de Pessoa -> Classe

Pessoas em si -> objetos



Instâncias!

Pessoa loide;

loide = new Pessoa( );

Classe Pessoa

- nome
- dataDeNascimento
- sexo
- cpf
- ...

+ dormir()
+ beber()
+ comer()
+correr()
+ engordar()
+ emagrecer()
+ trabalhar()

Pessoa debi;

debi = new Pessoa( );

Sintaxe - Variáveis

tipo_variável nome_variável;

Exemplo:

int numero;

string nome;

Atribuição de valores para uma variável:

nome_variável = valor;

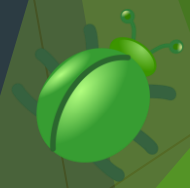
Int numero = 30;

Mão na Massa

Vamos construir uma pessoa!

O que uma pessoa tem?

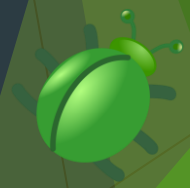
C#



O que toda pessoa faz?

Ações e Comportamentos:

- Dorme
- Bebe
- Come
- Trabalha
- Estuda



O que toda pessoa faz?

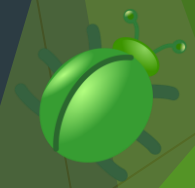
Ações e Comportamentos:

- Dorme
- Bebe
- Come
- Trabalha
- Estuda
- Cuida da vida alheia

Pessoa faça algo!

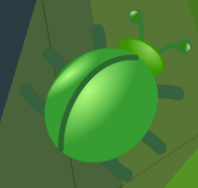
Podemos perguntar a essa pessoa qual é a sua idade?

Essa pessoa pode dançar, dormir ou fazer qualquer outra coisa?



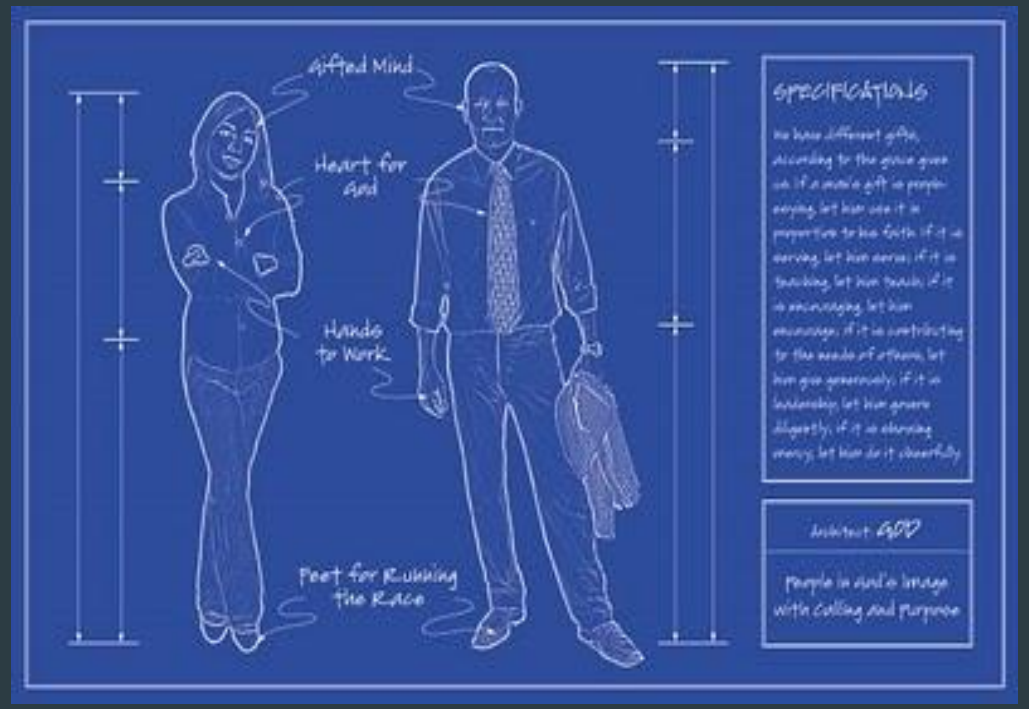
Pessoa faça algo!

NÃO!



Blueprint de Pessoa

O que fizemos até agora foi projetar como uma pessoa seria:

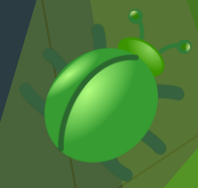


Primeiro

Primeiro precisamos “criar” uma pessoa...

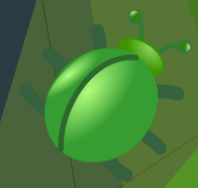
Atividade

Acessar o que ele “tem” e “pedir” para ela fazer algo...



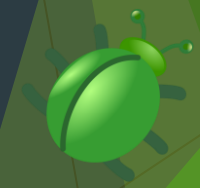
Código Gerado

... Classe PESSOA



Código Gerado

... Objeto KREUSA



Operadores

Tipos de operadores:

Relacionais (==, !=, >, <, >=, <=)

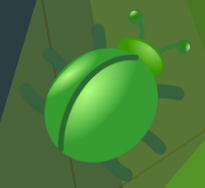
Lógicos (&&, ||, !)

Atribuição (=, +=, *=, /=)

Ternários.

Tipos de Dados

Palavra Chave	Tipo	Faixa de valores
bool	System.Boolean	true ou false
double	System.Double	$\pm 5{,}0 \times 10^{-324}$ a $\pm 1{,}7 \times 10^{308}$
float	System.Single	$\pm 1{,}5 \times 10^{-45}$ a $\pm 3{,}4 \times 10^{38}$
int	System.Int32	-2.147.483.648 a 2.147.483.647
long	System.Int64	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
String	System.String	“Meu nome é Bruno"



Casting e ToString

```
int x = 42;

String numeroEmTexto = x.ToString();
Console.WriteLine(numeroEmTexto );
```

Output: 42

Atributos

- Declaram o que toda a Classe deve ter
- São declarados fora do bloco
- São diferentes de variáveis temporárias
- São variáveis Membros

Métodos

Com ou sem Retorno

São as ações – Comportamento

Métodos - Parâmetros

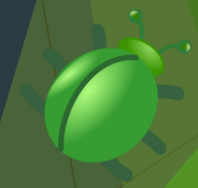
Ou argumentos

São informações passadas aos métodos

Passam o que o método precisa saber para me dar uma resposta ou realizar determinada ação

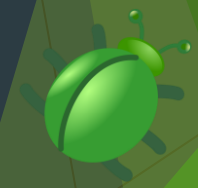
Void

Método SEM retorno



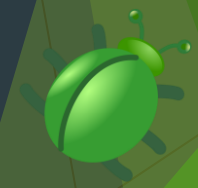
Modificador Private

- Protege o acesso aos atributos da classe
- Permite o acesso somente dentro da classe



Getters e Setters

São métodos responsáveis por Acessar e Modificar os Atributos das classes



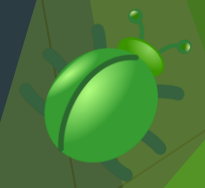
Construtores

Vai inicializar as variáveis, cada um dos atributos da classe

Se fosse não passar nada ele vai inicializar null

Mas podemos criar um Construtor passando Parâmetros para criar o objeto

Devemos fazer o atributo do nosso objeto receber o parâmetro recebido no Construtor



This

This -> palavra reservada

This.nome;

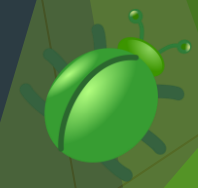
O atributo nome desta Classe

Quando passamos um parâmetro igual ao nome do atributo da classe utilizamos o this para indicar que estamos pegando o atributo desta Classe

This

Pessoa(string nome)

This.nome = nome;



Encapsular

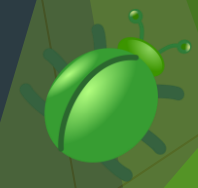
- Esconder todos os membros de uma classe
- Esconder como funcionam seus comportamentos (métodos)
- Fundamental para facilitar a Manutenção
- Não precisaremos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está encapsulada

Encapsular

É sempre bom programar pensando na interface da sua classe, como seus usuários a estarão utilizando, e não somente em como ela vai funcionar. A implementação em si, o conteúdo dos métodos, não tem tanta importância para o usuário dessa classe,

uma vez que ele só precisa saber o que cada método pretende fazer, e não como ele faz, pois isto pode mudar com o tempo.

Essa frase vem do livro Design Patterns, de Eric Gamma et al.



Encapsular

Quando você dirige um carro, o que te importa são os pedais e o volante (interface) e não o motor que você está usando (implementação).

É claro que um motor diferente pode te dar melhores resultados, mas o que ele faz é o mesmo que um motor

menos potente, a diferença está em como ele faz

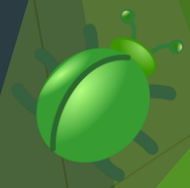
Encapsular

Para trocar um carro a álcool para um a gasolina você não precisa reaprender a dirigir! (trocar a implementação dos métodos não precisa mudar a interface, fazendo com que as outras classes continuem usando eles da mesma maneira). ·

Herança

Devemos deixar as Classes mais Genéricas

Não devemos criar uma outra classe e copiar o código novamente



Herança

Não devemos criar uma outra classe e copiar o código novamente

Se precisarmos adicionar uma nova informação para todas as pessoas, não precisaremos passar por todas as classes de pessoas e adicionar esse atributo

Uma classe herda tudo que a outra tem

Super Classe ou Classe Pai

SubClasse Classe ou Classe Filha

Herança

#Dica:

Devemos sempre fazer a pergunta:

É 1?

O (x) é um pássaro?

X = pato

X = ganso

X = cisne

Herança

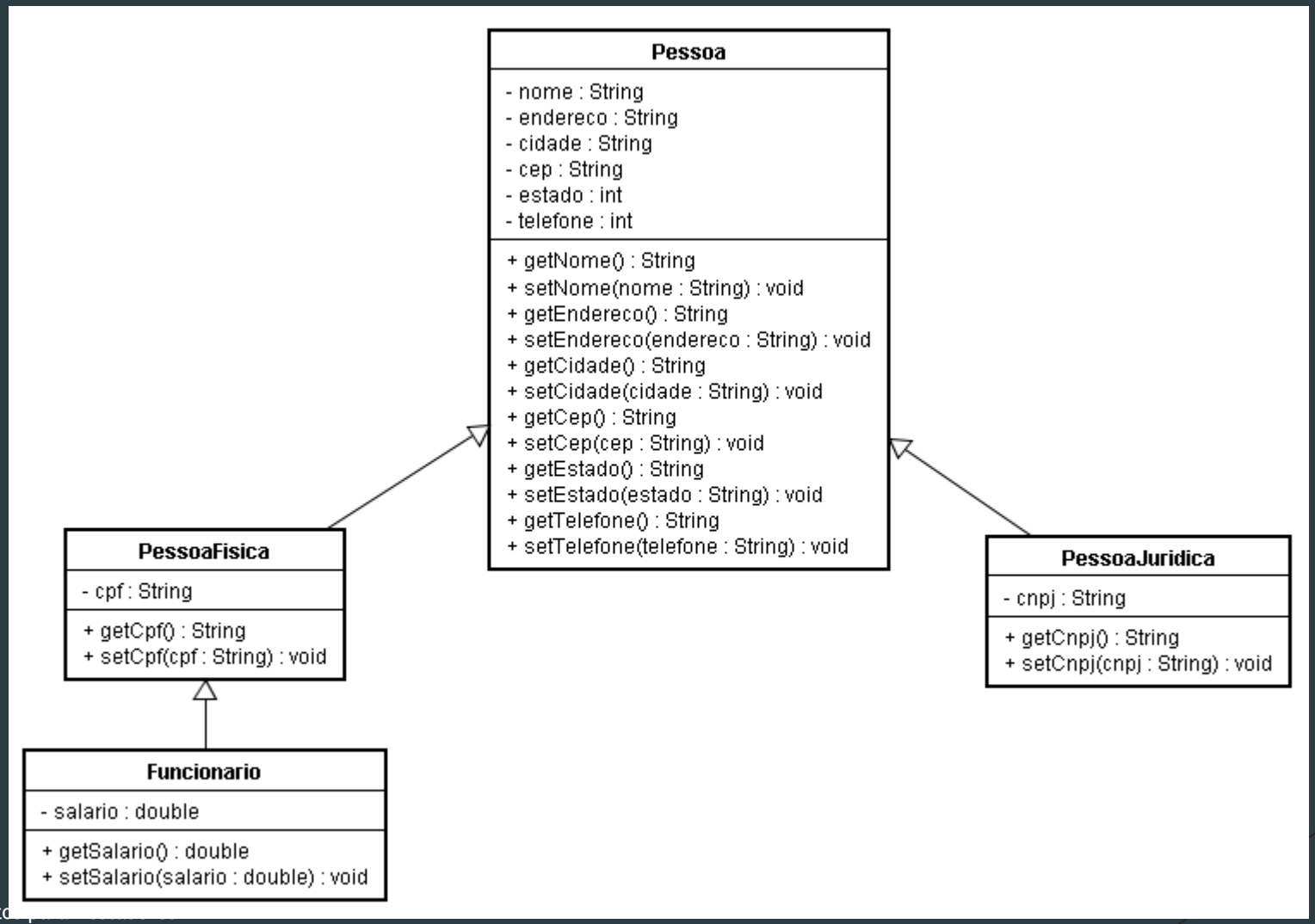
Extensão da Classe Pai

Classe Filha possui possuirá também os atributos de definidos na classe Pai

Herda Todos os Atributos e Métodos da Classe Pai

@Override - deixar explícito no seu código que determinador método é a reescrita de um método da sua classe mãe

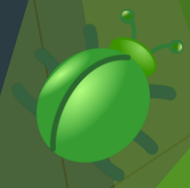
Uma classe pode ter várias filhas, mas pode ter apenas uma mãe, é a chamada herança simples


Pessoa
- nome : String
- endereco : String
- cidade : String
- cep : String
- estado : int
- telefone : int
+ getNome() : String
+ setNome(nome : String) : void
+ getEndereco() : String
+ setEndereco(endereco : String) : void
+ getCidade() : String
+ setCidade(cidade : String) : void
+ getCep() : String
+ setCep(cep : String) : void
+ getEstado() : String
+ setEstado(estado : String) : void
+ getTelefone() : String
+ setTelefone(telefone : String) : void
PessoaFisica
- cpf : String
+ getCpf() : String
+ setCpf(cpf : String) : void
PessoaJuridica
- cnpj : String
+ getCnpj() : String
+ setCnpj(cnpj : String) : void
Funcionario
- salario : double
+ getSalario() : double
+ setSalario(salario : double) : void

Herança

Cuidado: herda os atributos e métodos privados, porém não consegue acessá-los diretamente.

Para acessar um membro privado na filha indiretamente, seria necessário que a mãe expusesse um outro método visível que invocasse esse atributo ou método privado



Associação

Representa um relacionamento entre classes onde uma das classes está presente como atributo da outra

FAZ PARTE?

Roda FAZ PARTE do Carro

N x N

1 x N

1 x 1

Composição

Informa que uma classe FAZ PARTE de outra de forma EXCLUSIVA

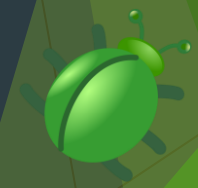
Os objetos são independentes

Um aluno pode existir sem a necessidade de um professor e vice e versa

Indiferente do relacionamento entre eles, ambos podem existir sem o outro

N alunos para N professores

1 Empresa (CWI) para N funcionários



Composição

Se excluir a classe responsável pelo relacionamento, então DEVE excluir a classe que ele possui relacionamento

Agregação

Informa que uma classe FAZ PARTE de outra, mas NÃO de forma EXCLUSIVA

Representa um todo dividido em várias partes

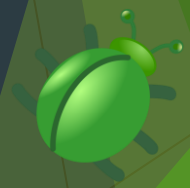
Quando o relacionamento é do tipo Faz parte

1 p/ N

Tester **faz parte** do Time de Desenvolvimento

Agregação

Se excluir a classe responsável pelo relacionamento NÃO DEVE excluir a classe que ele possui relacionamento

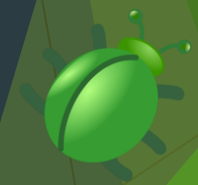


Private

Então porque usar private? Depois de um tempo programando orientado a objetos, você vai começar a sentir que nem sempre é uma boa ideia deixar que a classe filha acesse os atributos da classe mãe, pois isso quebra um pouco a ideia de que só aquela classe deveria manipular seus atributos.

Essa é uma discussão um pouco mais avançada.

O tipo ou membro pode ser acessado somente pelo código na mesma classe



Static

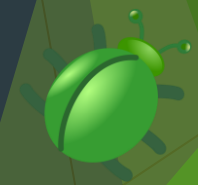
É uma classe ou atributo que é comum a todos os objetos criados a partir dela/dele

Ex: classe de iniciação de projeto

Atributo resultadoSoma

No decorrer do programa podemos ter em diversas partes do sistema uma chamada ao método Somar()

Se ele mandar o resultado da soma para um atributo static o resultado será sempre o mesmo em todo sistema



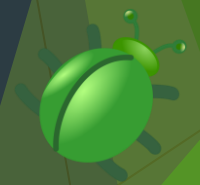
Protected

E se precisamos acessar os atributos que herdamos? Não gostaríamos de deixar os atributos de Pessoa, public, pois dessa maneira qualquer um poderia alterar os atributos dos objetos deste tipo.

Existe um outro modificador de acesso, o

PROTECTED, que fica entre o private e o public.

Um atributo protected só pode ser acessado (visível) pela própria classe e por suas subclasses



Acoplamento

O uso de herança aumenta o acoplamento entre as classes, isto é, o quanto uma classe depende de outra. A relação entre classe mãe e filha é muito forte e isso acaba fazendo com que o programador das classes filhas tenha que conhecer a implementação da classe pai e vice-versa - fica difícil fazer uma mudança pontual no sistema.

Classes Abstratas

Classe que define as características e comportamentos de um conjunto de objetos

Mas, nem todas as classes são feitas para serem instanciadas

Existem classes que são criadas para conter/agrupar características comuns a diversas classes e então ser herdadas por elas

Classes Abstratas

Palavra chave Abstract

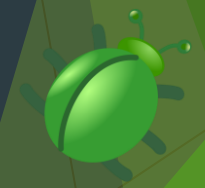
Só consigo instanciar as classes filhas delas

Métodos Abstratos

Classes abstratas podem ser utilizadas para prover a definição de métodos que devem ser implementados em todas as suas classes filhas sem apresentar uma implementação para esses métodos

Toda classe que possui pelo menos um método abstrato é uma classe abstrata, mas uma classe pode ser abstrata sem possuir nenhum método abstrato

Ex: Classe Simular

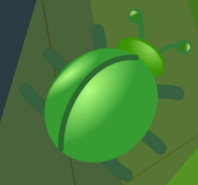


Abstract

Usamos a palavra chave abstract para impedir que a classe possa ser instanciada

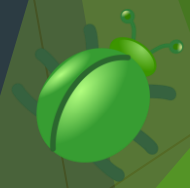
O problema é instanciar a classe - criar referência, você pode. Se ela não pode ser instanciada, para que serve? Serve para o polimorfismo e herança dos atributos e métodos, que são recursos muito poderosos, como já vimos.

Qualquer classe que estender a classe Funcionario será obrigada a reescrever este método, tornando-o "concreto"



Abstract

Como posso acessar o método getBonificacao se ele não existe na classe Pessoa? Já que o método é abstrato, com certeza suas subclasses têm esse método, o que garante que essa invocação de método não vai falhar?



Abstract

Basta pensar que uma referência do tipo Funcionario nunca aponta para um objeto que não tem o método getBonificacao, pois não é possível instanciar uma classe abstrata, apenas as concretas. Um método abstrato obriga a classe em que ele se encontra ser abstrata.

Abstract

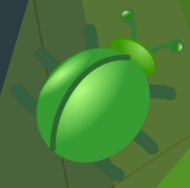
Ou seja, tenho a classe abstrata Funcionario, com o método abstrato getBonificacao; as classes Gerente e Presidente estendendo Funcionario e implementando o método getBonificacao; e, por fim, a classe Diretor, que estende Gerente, mas não implementa o método getBonificacao

Abstract

Essas classes vão compilar? Vão rodar? A resposta é sim. E, além de tudo, farão exatamente o que nós queremos, pois, quando Gerente e Presidente possuem os métodos perfeitamente implementados, a classe Diretor, que não possui o método implementado, vai usar a implementação herdada de Gerente

Abstract

Se eu não reescrever um método abstrato da minha classe mãe, o código não compilará. Mas posso, em vez disso, declarar a classe como abstrata!



Polimorfismo

Significa muitas formas (grego)

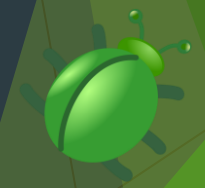
Permite escrever programas que processam objetos que compartilham a mesma classe pai como se todos fossem objetos da superclasse

Simplifica a programação

Ex: método sacar é diferente na conta corrente, na conta poupança e na conta de investimento

Não é sobrecarga de método, mas sim mudar

Executar ações diferentes



Polimorfismo

Não é sobrecarga de método, mas sim sobrescrita (mudar

Executar ações diferentes)

SOBRECARGA

Mesmo nome do método, mas com diferentes quantidades de parâmetros = sobrecarga de método

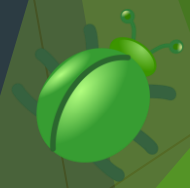
Acontece dentro da mesma classe

Polimorfismo

SOBRESCRITA

Mesmo nome do método, mas em diferentes classes que herdam de uma mesma classe abstrata

Tem que escrever o método totalmente, ou seja, sobrescreve o método da classe pai



Polimorfismo

Ex: Podemos ter Funcionários Vendedor e Testador

Classe Funcionário pode ser abstrata nesse contexto

Podemos ter métodos como CalcularSalario() que está vazio, não implementa nada

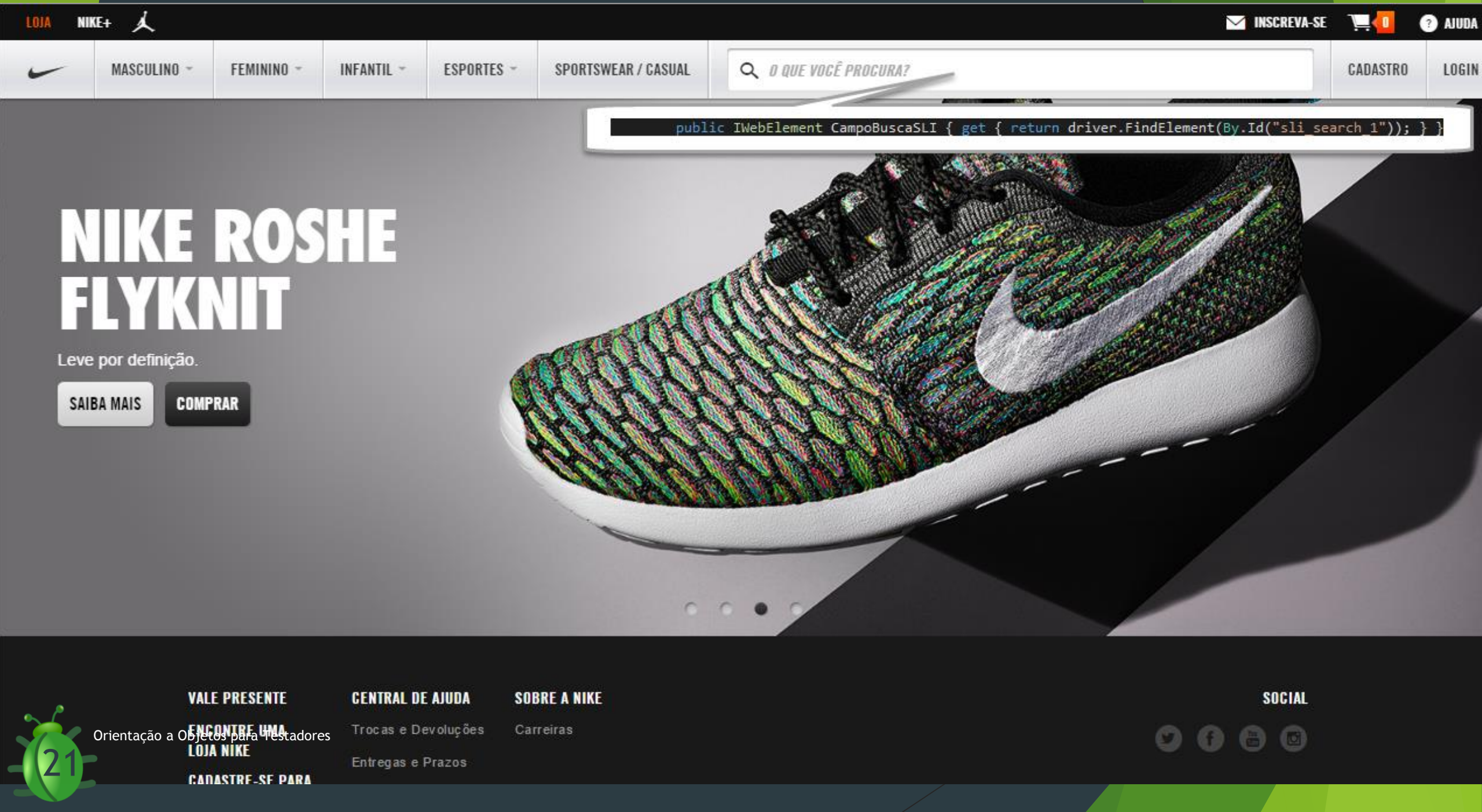
Nas classes filhas, será diferente para cada um dos funcionários

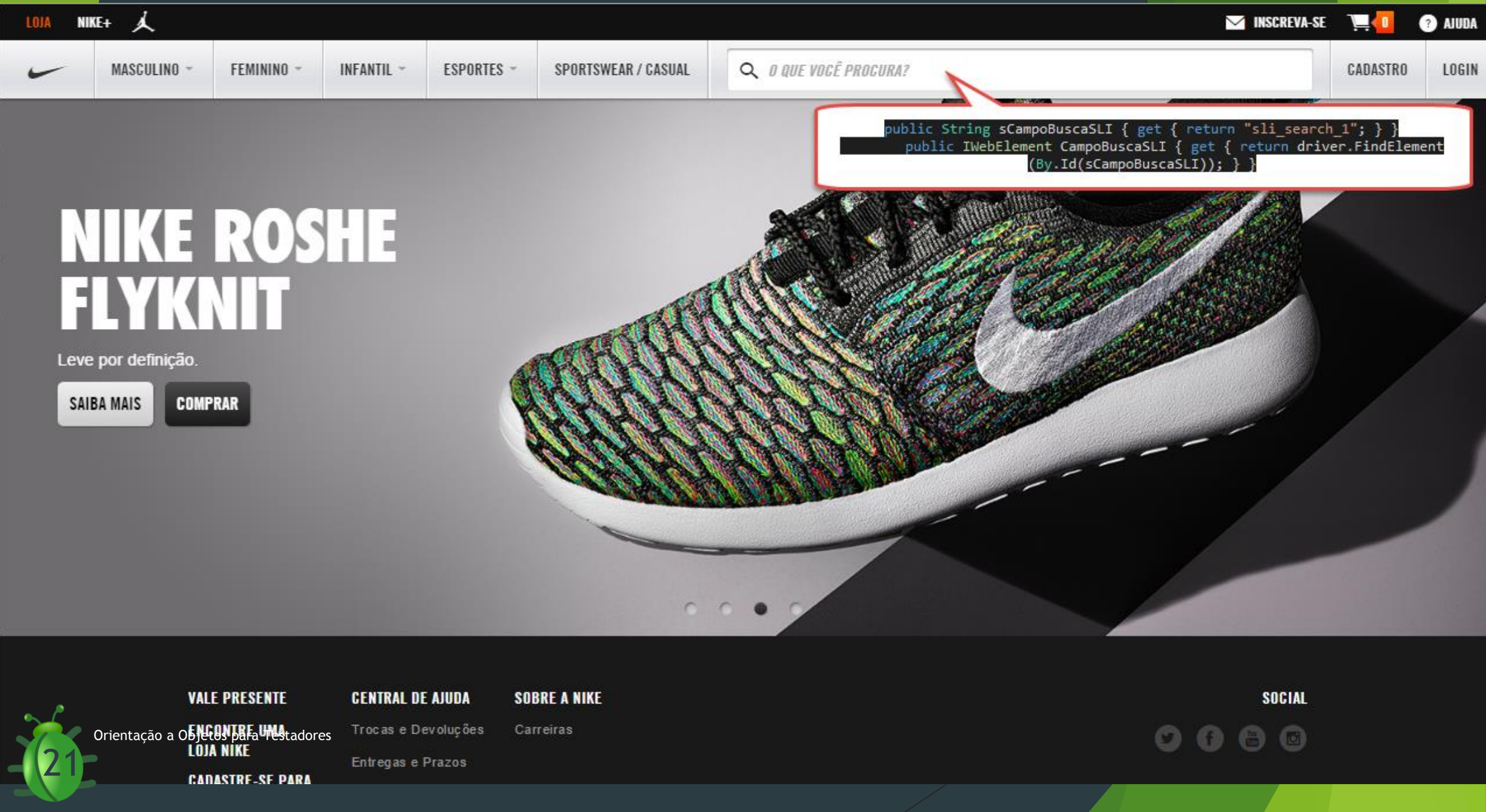
Polimorfismo

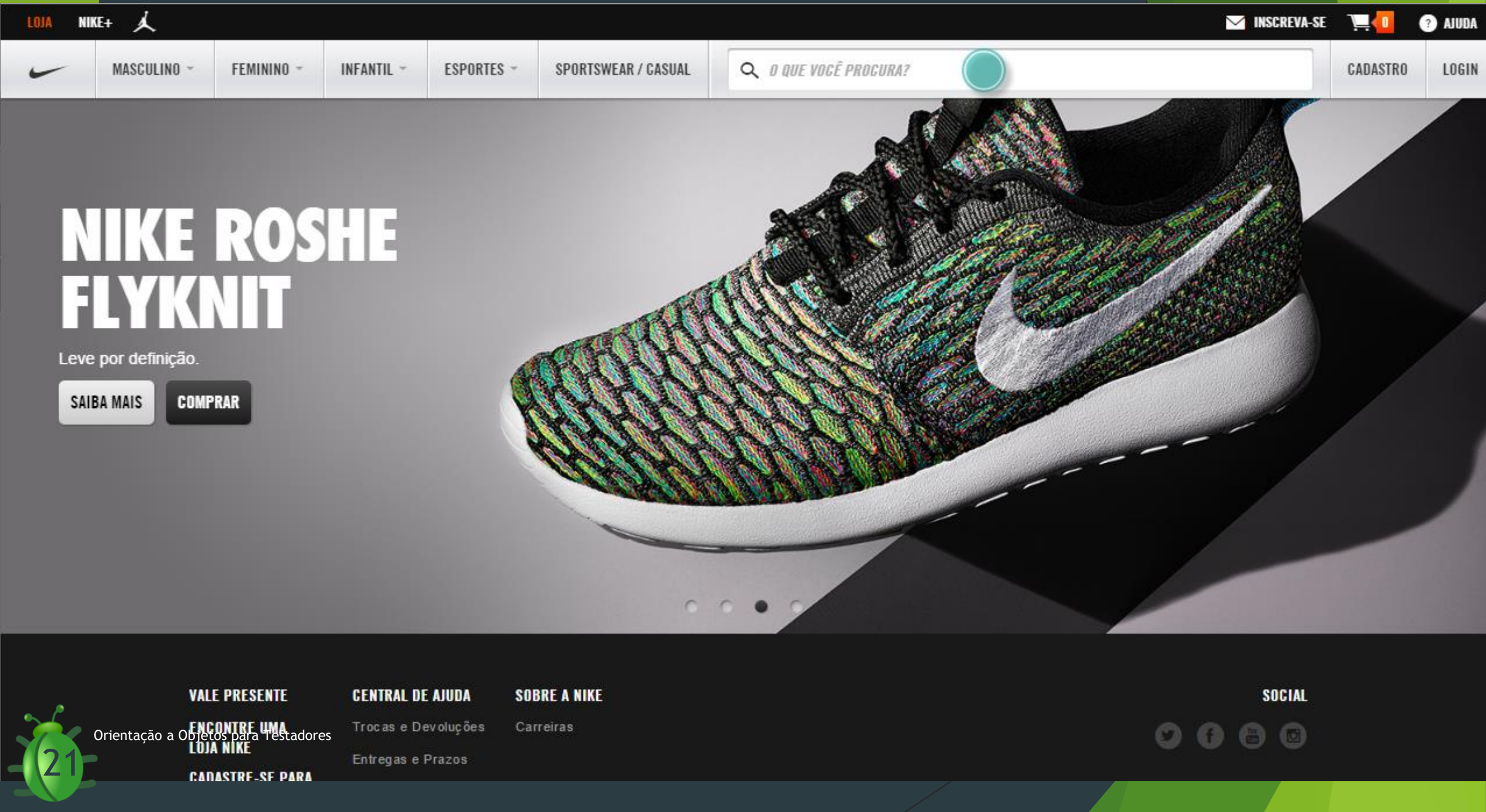
Ações diferentes em métodos herdados das mesmas classes pai

Que são implementados completamente diferentes

Vendedor tem o salário calculado com comissões e outros e testador vocês sabem como é né...


LOJA
NIKE+
INSCREVA-SE
0
AJUDA
MASCULINO
FEMININO
INFANTIL
ESPORTES
SPORTSWEAR / CASUAL
O QUE VOCÊ PROCURA?
CADASTRO
LOGIN
public IWebElement CampoBuscaSLI { get { return driver.FindElement(By.Id("sli_search_1")); } }
NIKE ROSHE FLYKNIT
Leve por definição.
SAIBA MAIS
COMPRAR
VALE PRESENTE
ENCONTRE UMA LOJA NIKE
CADASTRE-SE PARA
CENTRAL DE AJUDA
Trocas e Devoluções
Entregas e Prazos
SOBRE A NIKE
Carreiras
SOCIAL


LOJA
NIKE+
INSCREVA-SE
0
AJUDA
MASCULINO
FEMININO
INFANTIL
ESPORTES
SPORTSWEAR / CASUAL
O QUE VOCÊ PROCURA?
CADASTRO
LOGIN
public String sCampoBuscaSLI { get { return "sli_search_1"; } }
public IWebElement CampoBuscaSLI { get { return driver.FindElement
(By.Id(sCampoBuscaSLI)); } }
NIKE ROSHE
FLYKNIT
Leve por definição.
SAIBA MAIS
COMPRAR
VALE PRESENTE
ENCONTRE UMA
LOJA NIKE
CADASTRE-SE PARA
CENTRAL DE AJUDA
Trocas e Devoluções
Entregas e Prazos
SOBRE A NIKE
Carreiras
SOCIAL


LOJA
NIKE+
INSCREVA-SE
0
AJUDA
MASCULINO
FEMININO
INFANTIL
ESPORTES
SPORTSWEAR / CASUAL
O QUE VOCÊ PROCURA?
CADASTRO
LOGIN
NIKE ROSHE FLYKNIT
Leve por definição.
SAIBA MAIS
COMPRAR
VALE PRESENTE
ENCONTRE UMA LOJA NIKE
CADASTRE-SE PARA
CENTRAL DE AJUDA
Trocas e Devoluções
Entregas e Prazos
SOBRE A NIKE
Carreiras
SOCIAL

Page Object

A página é um objeto!