

SOLID no Android

Android Dev Conference 2016

Esses são os slides que apresentei no
Android Dev Conference 2016.

Acreditei que por não conter nenhum
“voice over” seria justo fazer pequenas
adaptações para melhor entendimento,
espero que seja útil para você!

Contatos.

- Twitter: <http://twitter.com/marcellogalhard>
- LinkedIn: <https://www.linkedin.com/in/marcellogalhardo>
- Github: <https://github.com/marcellogalhardo>
- E-mail: marcello.galhardo@gmail.com
- Medium: <https://medium.com/@marcellogalhardo>

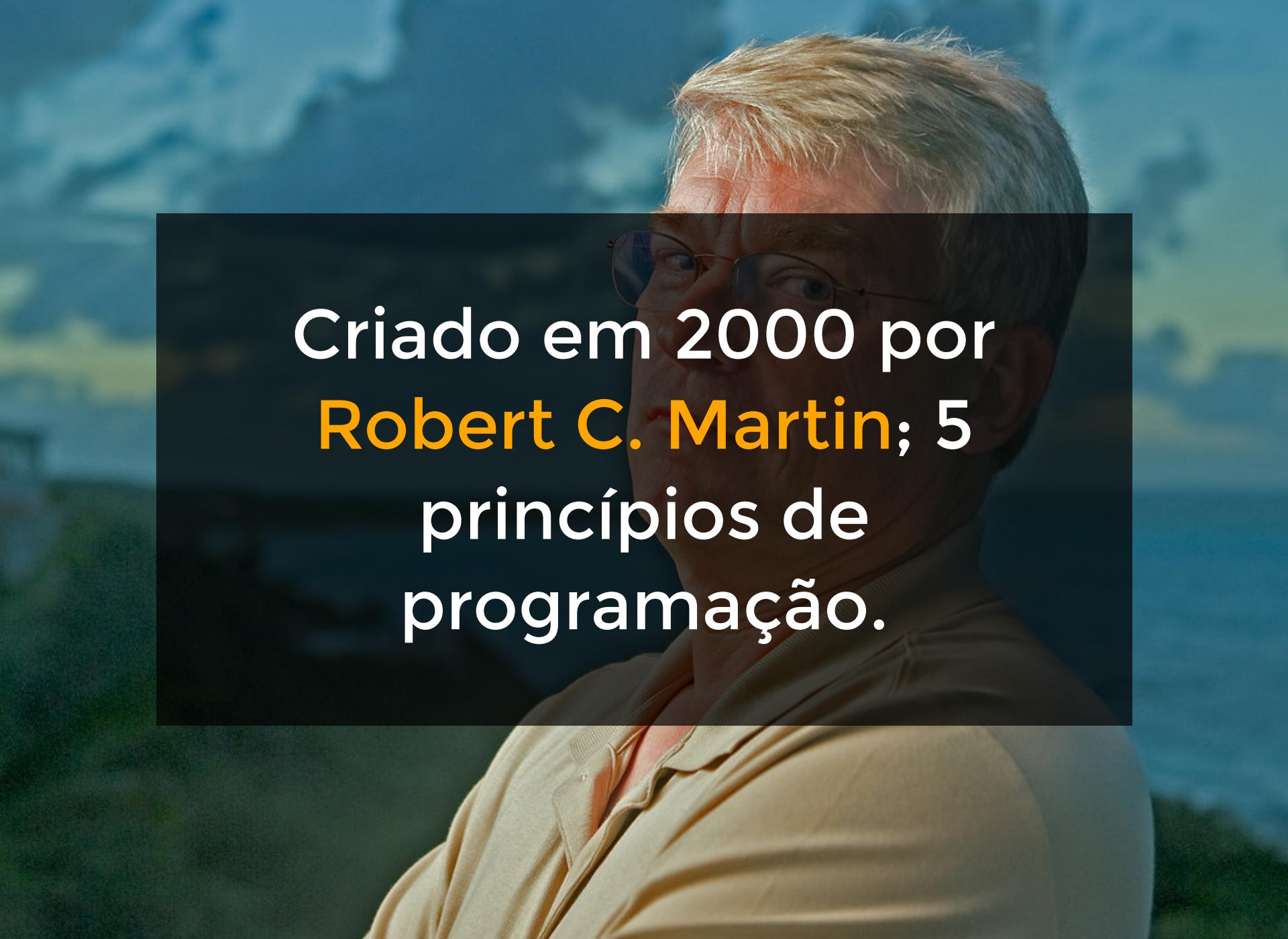


Marcello Galhardo

Desenvolvedor Android





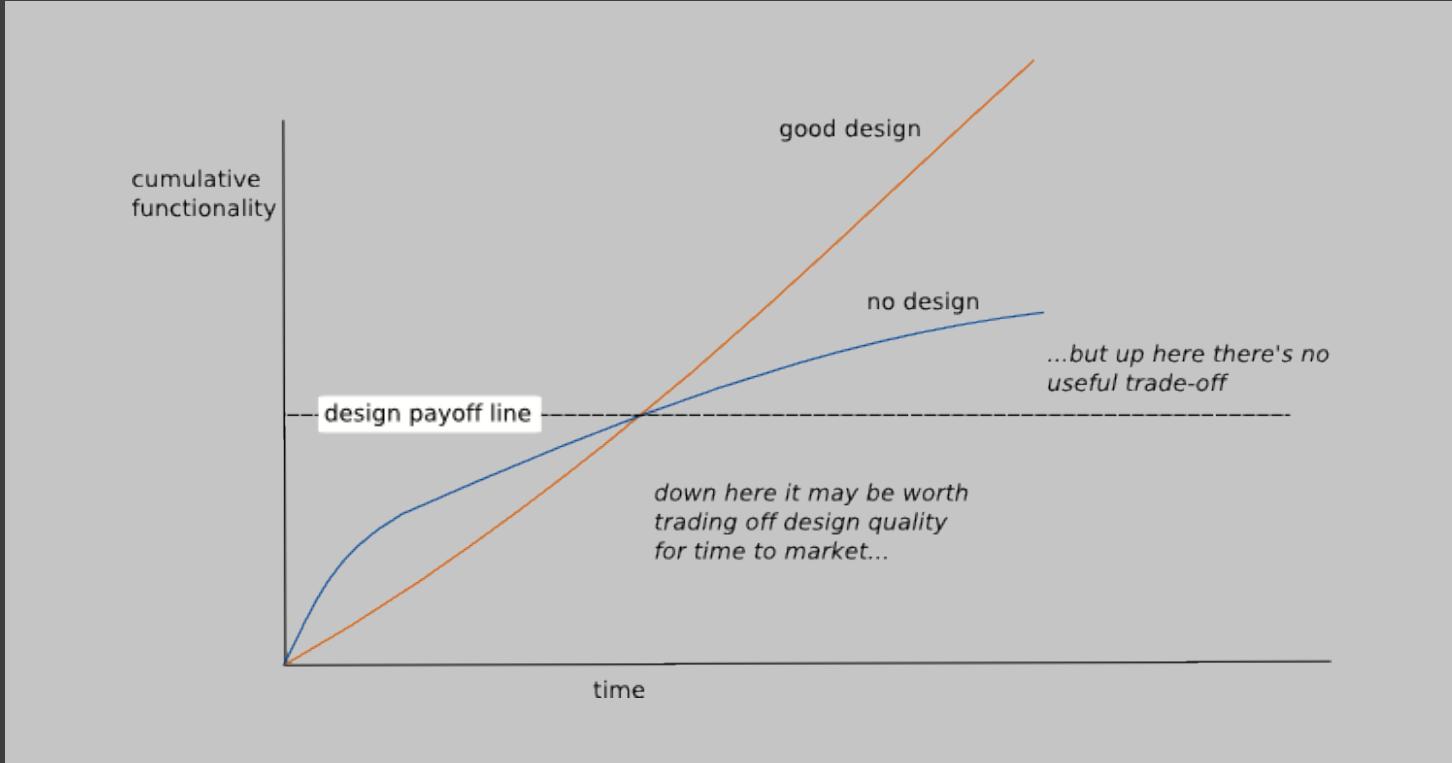


Criado em 2000 por
Robert C. Martin; 5
princípios de
programação.

"Paciência você deve
ter meu jovem
Padawan."

"Quem planta
gambiarras, colhe
bugs."

Trade Off Qualidade



<http://martinfowler.com/bliki/DesignStaminaHypothesis.html>

Princípio da Responsabilidade Única

Single Responsibility Principle

*"Uma classe deve ter
um, e somente um,
motivo para mudar."*

Violação.

```
public class Produto {  
  
    private String descricao;  
    private int quantidade;  
    private long preco;  
  
    // ... getters/setters  
  
}  
  
public class Pedido {  
  
    private int numeroDoPedido;  
    private List<Produto> produtos = new ArrayList<>();  
  
    // ... getters/setters  
  
}
```

```
public class PedidoRecyclerAdapter
    extends RecyclerView.Adapter<PedidoRecyclerAdapter.ViewHolder> {

    private List<Pedido> pedidos;

    public OrderRecyclerAdapter(List<Pedido> pedidos) {
        this.pedidos = pedidos;
    }

    @Override public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(context);
        View v = inflater.inflate(R.layout.item_pedidos, parent, false);
        return new ViewHolder(v);
    }

    @Override public void onBindViewHolder(ViewHolder holder, int position) {
        // TODO: Faz o vínculo entre o modelo e a view
    }

    @Override public int getItemCount() {
        return pedidos.size();
    }

    // ... ViewHolder e métodos
}
```

```
@Override public void onBindViewHolder(ViewHolder holder, int position) {
    Pedido pedido = items.get(position);

    holder.numeroDoPedido.setText(pedido.getNumeroDoPedido().toString());

    long total = 0;
    for (Produto produto : pedido.getProdutos()) {
        total += produto.getPreco();
    }
    NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.US);
    String valorTotal = formatter.format(total);
    holder.valorTotalDoPedido.setText(valorTotal);

    holder.itemView.setTag(pedido);
}

public static class ViewHolder extends RecyclerView.ViewHolder {
    public TextView numeroDoPedido;
    public TextView valorTotalDoPedido;

    // FindViews.
}
```

```
long total = 0;
for (Produto produto : pedido.getProdutos()) {
    total += produto.getPreco();
}
NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.US);
String valorTotal = formatter.format(total);
holder.valorTotalDoPedido.setText(valorTotal);
```

Solução.

```
public class Pedido {  
  
    private int numeroDoPedido;  
    private List<Produto> produtos = new ArrayList<>();  
  
    // ... getters/setters  
  
    public long getValorTotalDoPedido() {  
        long total = 0;  
        for (Produto produto : pedido.getProdutos()) {  
            total += produto.getPreco();  
        }  
        return total;  
    }  
  
    public String getValorTotalDoPedidoFormatado() {  
        long total = getValorTotalDoPedido();  
        NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.US);  
        String valorTotal = formatter.format(total);  
        return valorTotal;  
    }  
}
```

```
@Override  
public void onBindViewHolder(ViewHolder holder, int position) {  
    Pedido pedido = items.get(position);  
    holder.numeroDoPedido.setText(pedido.getNumeroDoPedidoFormatado());  
    holder.valorTotalDoPedido.setText(pedido.getValorTotalDoPedidoFormato());  
    holder.itemView.setTag(pedido);  
}
```

Rigidez.

Princípio do Aberto e Fechado

Open/Closed Principle

*"Você deve ser capaz de
estender um
comportamento de uma
classe, sem modificá-lo."*

Violação.

```
public class Pedido {  
  
    private int numeroDoPedido;  
    private List<Produto> produtos = new ArrayList<>();  
  
    // ... getters/setters  
  
    public long getValorTotalDoPedido() {  
        long total = 0;  
        for (Produto produto : pedido.getProdutos()) {  
            total += produto.getPreco();  
        }  
        return total;  
    }  
  
    public String getValorTotalDoPedidoFormatado() {  
        long total = getValorTotalDoPedido();  
        NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.US);  
        String valorTotal = formatter.format(total);  
        return valorTotal;  
    }  
}
```

```
private static final int SEM_DESCONTO = 0;
private static final int DESCONTO_10_POR_CENTO = 1;
private static final int DESCONTO_15_POR_CENTO = 2;

private int tipoDeDesconto;

public long getValorTotalDoPedido() {
    long total = 0;
    for (Produto produto : pedido.getProdutos()) {
        total += produto.getPreco();
    }
    if (tipoDeDesconto == DESCONTO_10_POR_CENTO) {
        long descontoDe10PorCento = total * 0.1;
        total += descontoDe10PorCento;
    } else if (tipoDeDesconto == DESCONTO_15_POR_CENTO ) {
        long descontoDe15PorCento = total * 0.15;
        total += descontoDe15PorCento;
    }
    return total;
}
```

```
public class Pedido {  
  
    private static final int SEM_DESCONTO = 0;  
    private static final int DESCONTO_10_POR_CENTO = 1;  
    private static final int DESCONTO_15_POR_CENTO = 2;  
  
    private int numeroDoPedido;  
    private List<Produto> produtos = new ArrayList<>();  
    private int tipoDeDesconto;  
  
    // ... getters/setters  
  
    public long getValorTotalDoPedido() {  
        long total = 0;  
        for (Produto produto : pedido.getProdutos()) {  
            total += produto.getPreco();  
        }  
        if (tipoDeDesconto == DESCONTO_10_POR_CENTO) {  
            long descontoDe10PorCento = total * 0.1;  
            total += descontoDe10PorCento;  
        } else if (tipoDeDesconto == DESCONTO_15_POR_CENTO ) {  
            long descontoDe15PorCento = total * 0.15;  
            total += descontoDe15PorCento;  
        }  
        return total;  
    }  
  
    public String getValorTotalDoPedidoFormatado() {  
        long total = getValorTotalDoPedido();  
        NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.US);  
        String valorTotal = formatter.format(total);  
        return valorTotal;  
    }  
}
```

```
private static final int SEM_DESCONTO = 0;
private static final int DESCONTO_10_POR_CENTO = 1;
private static final int DESCONTO_15_POR_CENTO = 2;
private static final int DESCONTO_20_POR_CENTO = 3;
private static final int DESCONTO_25_POR_CENTO = 4;

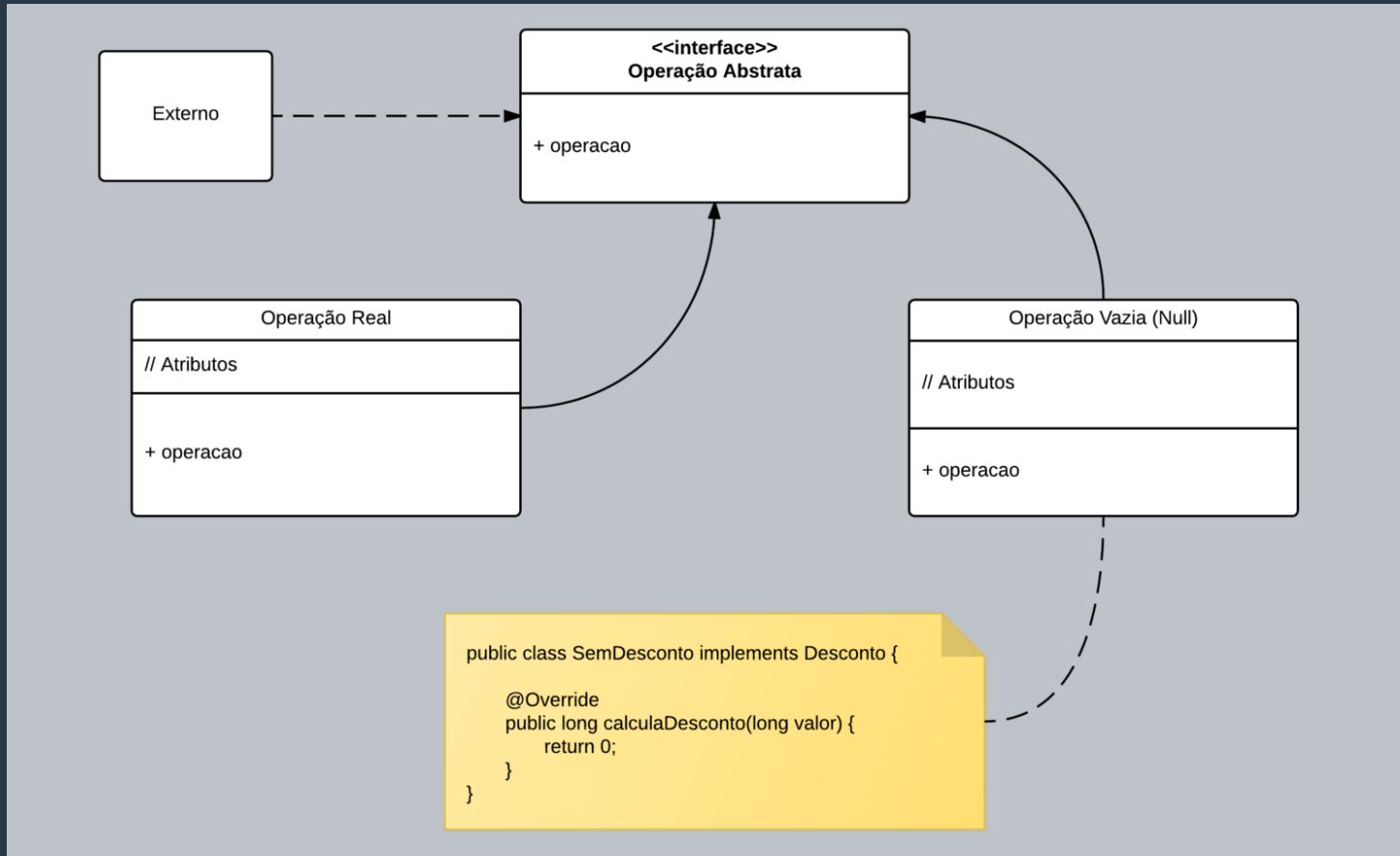
private int tipoDeDesconto;

public long getValorTotalDoPedido() {
    long total = 0;
    for (Produto produto : pedido.getProdutos()) {
        total += produto.getPreco();
    }
    if (tipoDeDesconto == DESCONTO_10_POR_CENTO) {
        long descontoDe10PorCento = total * 0.1;
        total += descontoDe10PorCento;
    } else if (tipoDeDesconto == DESCONTO_15_POR_CENTO) {
        long descontoDe15PorCento = total * 0.15;
        total += descontoDe15PorCento;
    } else if (tipoDeDesconto == DESCONTO_20_POR_CENTO) {
        long descontoDe20PorCento = total * 0.20;
        total += descontoDe20PorCento;
    } else if (tipoDeDesconto == DESCONTO_25_POR_CENTO) {
        long descontoDe25PorCento = total * 0.25;
        total += descontoDe25PorCento;
    }
    return total;
}
```

Solução.


```
public interface Desconto {  
  
    long calculaDesconto(long valor);  
  
}  
  
// Padrão de Projeto: Objeto Nulo.  
public class SemDesconto implements Desconto {  
  
    public long calculaDesconto(long valor) {  
        return 0;  
    }  
  
}  
  
public class DescontoDe10PorCento implements Desconto {  
  
    public long calculaDesconto(long valor) {  
        return valor * 0.1;  
    }  
  
}  
  
public class DescontoDe15PorCento implements Desconto {  
  
    public long calculaDesconto(long valor) {  
        return valor * 0.15;  
    }  
  
}  
  
// Outros descontos.
```

Objeto Nulo



```
private Desconto desconto = new SemDesconto();

public long getValorTotalDoPedido() {
    long total = 0;
    for (Produto produto : pedido.getProdutos()) {
        total += produto.getPreco();
    }

    // Não é necessário verificar se o desconto é null, pois
    // sempre existirá um desconto vazio: SemDesconto.
    desconto = desconto.calculaDesconto(total);
    return total + desconto;
}
```

```
public class Pedido {  
  
    private int numeroDoPedido;  
    private List<Produto> produtos = new ArrayList<>();  
    private Desconto desconto = new SemDesconto();  
  
    // ... getters/setters  
  
    public long getValorTotalDoPedido() {  
        long total = 0;  
        for (Produto produto : pedido.getProdutos()) {  
            total += produto.getPreco();  
        }  
        desconto = desconto.calculaDesconto(total);  
        return total + desconto;  
    }  
  
    public String getValorTotalDoPedidoFormatado() {  
        long total = getValorTotalDoPedido();  
        NumberFormat formatter = NumberFormat.getCurrencyInstance(Locale.US);  
        String valorTotal = formatter.format(total);  
        return valorTotal;  
    }  
}
```



Princípio da Substituição de Liskov

Liskov Substitution Principle

*"Classes derivadas
devem ser
substituíveis por
suas classes base."*

Violação.

```
// Violação do Princípio de Liskov.

public interface Carro {

    public void partidaNoMotor();
}

public class Ferrari implements Carro {

    @Override public double partidaNoMotor() {
        // TODO: Lógica.
    }
}

public class Tesla implements Carro {

    @Override public void ligaCarro() {
        // TODO: Lógica.
    }

    @Override public double partidaNoMotor() {
        // TODO: Lógica.
    }
}

// Iniciar Carro.
public void iniciarPartidaDeCarro(Carro carro) {
    carro.partidaNoMotor();
}
```

```
// "Tentativa de correção" do princípio de Liskov.

public void iniciarPartidaDeCarro(Carro carro) {
    if (carro instanceof Tesla) {
        Tesla tesla = (Tesla) carro;
        tesla.ligaCarro();
    }
    carro.partidaNoMotor();
}
```

Solução.

```
// Correção do Princípio de Liskov

public interface Carro {
    public void partidaNoMotor();
}

public class Ferrari implements Carro {

    @Override public double partidaNoMotor() {
        // TODO: Implementação.
    }
}

public class Tesla implements Carro {

    // Implementação do ligaCarro();

    @Override public double partidaNoMotor() {
        if (!estaDescarregado) {
            ligaCarro();
        }
        // TODO: Implementação.
    }
}

// Iniciar Carro.
public void iniciarPartidaDeCarro(Carro carro) {
    carro.partidaNoMotor();
}
```

Princípio da Segregação de Interface

Interface Segregation Principle

*"Muitas interfaces
específicas são
melhores do que
uma interface
única."*

Violação.


```
// Violação do Princípio da Segregação de Interfaces

Button botao = (Button) findViewById(R.id.botao);
botao.setOnClickListener(new View.OnClickListener {

    public void onClick(View v) {
        // TODO: Faz algo bem legal...
    }

    public void onLongClick(View v) {
        // Não precisamos disso.
    }

    public void onTouch(View v, MotionEvent event) {
        // Disso também não.
    }
});
```

Solução.

```
// Correção do Princípio da Segregação de Interfaces.

public interface OnClickListener {
    void onClick(View v);
}

public interface OnLongClickListener {
    void onLongClick(View v);
}

public interface OnTouchListener {
    void onTouch(View v, MotionEvent event);
}
```

Princípio da Inversão da Dependência

Dependency Inversion Principle

"Dependa de uma
abstração e não de
uma
implementação."

Violação.

```
// violação do Princípio da Inversão de Dependência.

class Presenter {

    private final UsuarioRepositorio usuarioRepositorio
        = new UsuarioRepositorio();

    // TODO: Implementação diversas.

}

class UsuarioRepositorio {

    public getUsuario() {
        // TODO: Recupera o usuário conectado.
    }

}
```

Solução.

```
// Correção do Princípio da Inversão de Dependência.

class Presenter {

    private final UsuarioRepositorio usuarioRepositorio;

    @Inject
    public Presenter(UsuarioRepositorio usuarioRepositorio) {
        this.usuarioRepositorio = usuarioRepositorio;
    }

    // TODO: Implementação diversas.

}

interface UsuarioRepositorio {

    Usuario getUsuario();

}

class UsuarioManager implements UsuarioRepositorio {

    @Override
    public getUsuario() {
        // TODO: Recupera o usuário conectado.
    }

}
```

GAME OVER

CONTINUE

EXIT

**Snake?
...Snake?
SNAAAAKE!!**