

INSTITUTO FEDERAL
ESPIRITO SANTO

PADRÕES DE PROJETO



INSTITUTO FEDERAL
ESPIRITO SANTO

PADRÕES DE PROJETO

- **Padrão Comportamental:** diz respeito a algoritmos e a atribuição de responsabilidades e comportamentos entre objetos.



PADRÕES DE PROJETO

Escopo	Propósito		
	Criativo	Estrutural	Comportamental
Classe	Método-Fábrica	Adaptador (classe)	Interpretador Método Modelo
Objeto	Construtor Fábrica Abstrata Protótipo Singular	Adaptador (objeto) Composto Decorador Fachada Peso-Mosca Ponte Procurador	Cadeia de Responsabilidade Comando Iterador Mediador Memorial Observador Estado Estratégia Visitador

PADRÕES DE PROJETO

- Padrões Comportamentais:
 - **Cadeia de responsabilidade:** evita o acoplamento de um objeto “sender” do objeto “receiver”. Dessa forma, diversos objetos podem responder o sender;
 - **Comando:** encapsula uma requisição como um objeto, deixando a parametrização do cliente com diferentes requisições;
 - **Iterator:** prover um meio de acesso sequencial aos elementos de uma estrutura, sem informar a representação da estrutura;
 - **Memento:** Captura e externalizar o estado de um objeto sem violar o encapsulamento e, dessa forma, prover um meio de restaurar o estado inicial do objeto.

PADRÕES DE PROJETO

- Padrões Comportamentais:
 - **Observer:** define a dependência de um objeto perante outros objetos e, assim, quando o estado de um objeto é mudado, os objetos dependentes são notificados e atualizados automaticamente.
 - **State:** permite que um objeto mude o seu comportamento, quando o seu estado interno muda;
 - **Strategy:** permite que o cliente mude o algoritmo de um objeto que utiliza uma função;
 - **Template Method:** define o “esqueleto” de um algoritmo em um método, utilizando as subclasses como etapas para o algoritmo;
 - **Visitor:** simula a adição de um método a uma classe sem precisar mudar a classe;

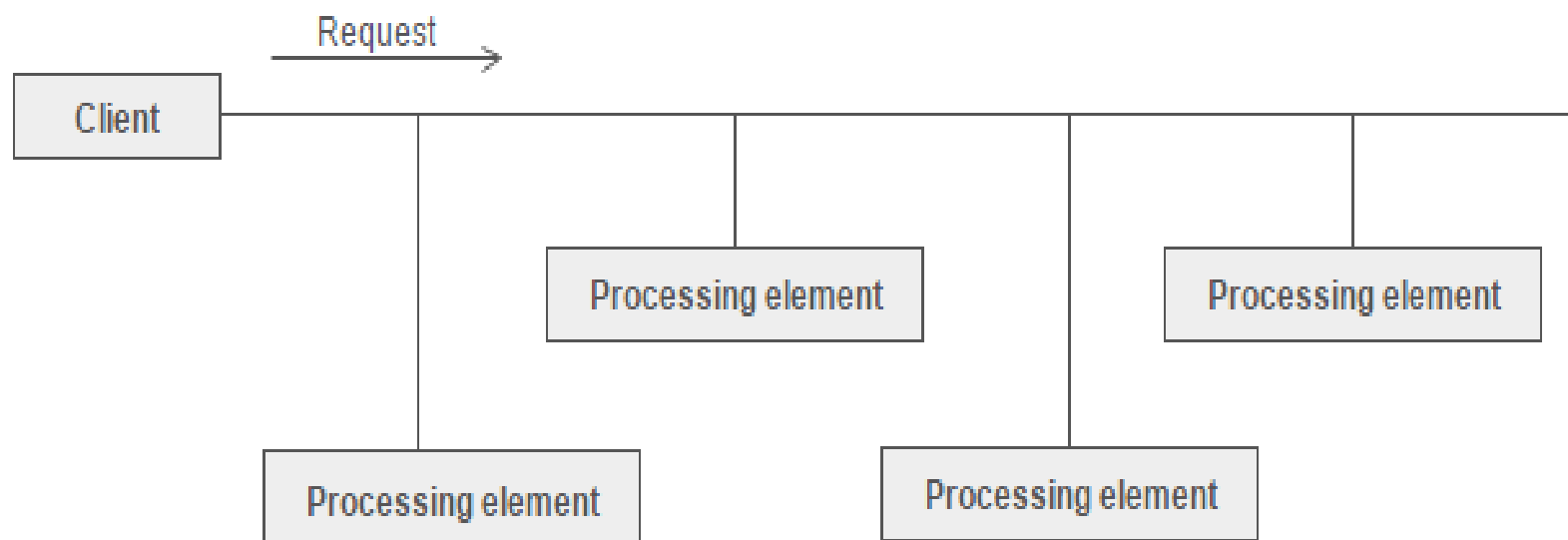
PADRÕES COMPORTAMENTAIS

Cadeia de responsabilidade

PADRÕES DE PROJETO

- Cadeia de Responsabilidade:
 - **Propósito:** evitar o acoplamento do *sender* de uma requisição com o receiver que irá tratar a requisição. A cadeia de receiver irá passar a requisição até um objeto tratar a requisição;
 - **Problema:**
 - A Motores SA recebe muitos emails por dia, incluindo requisições de serviços, requisição de vendas, reclamações e spam;
 - Ela deseja um sistema que trate o email, dependendo do tipo;

Cadeia de Responsabilidade

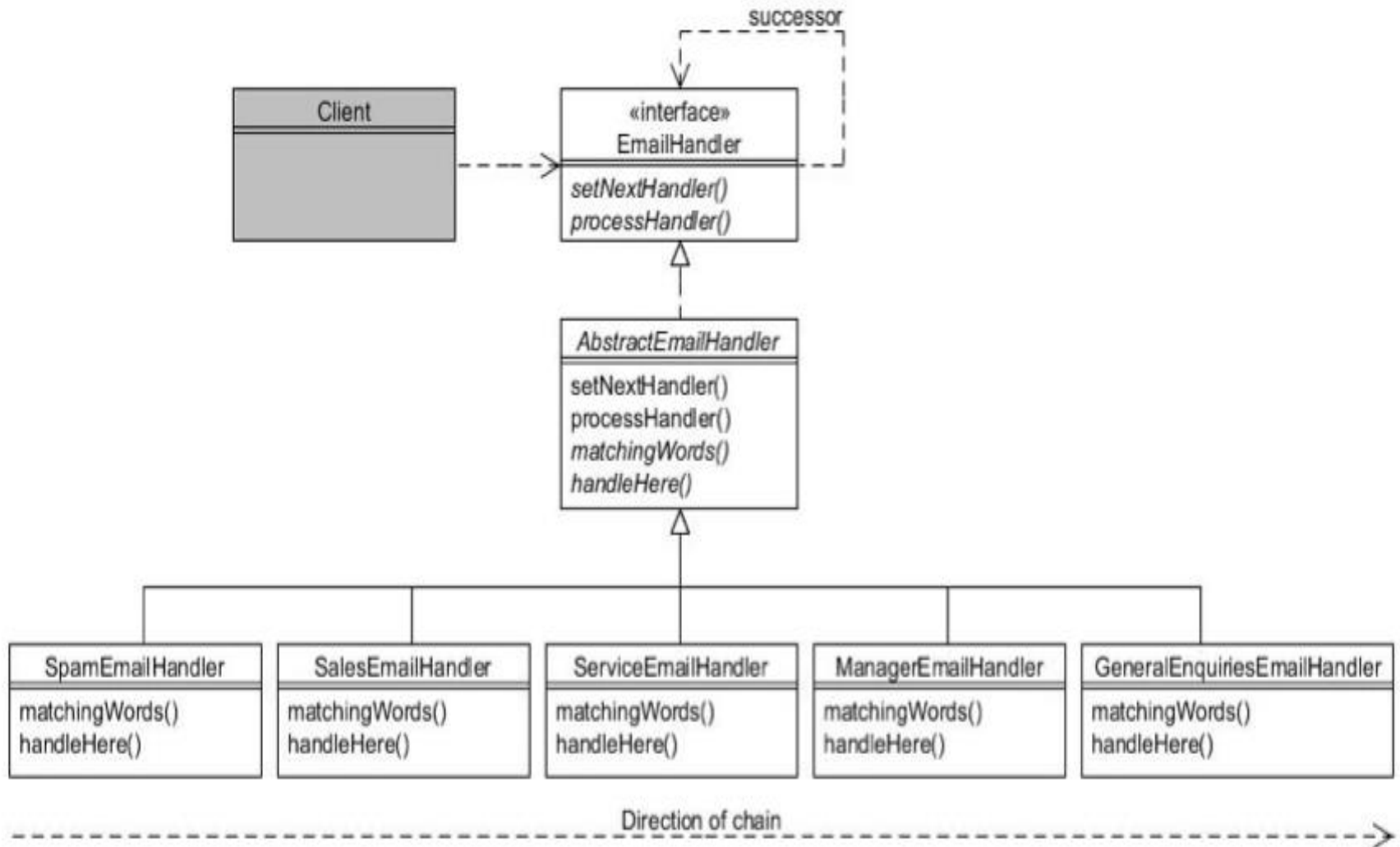


PADRÕES DE PROJETO

- Cadeia de Responsabilidade:
 - Regras:
 - Se o e-mail conter “viagra”, “pílulas” ou “medicina”, então encaminhe para o tratador de spam;
 - Se o e-mail conter “comprar” ou “venda”, então encaminhe para o departamento de vendas;
 - Se o e-mail conter “serviço” ou “reparar”, então encaminhe para o departamento de serviço gerais;
 - Se o e-mail conter “reclamação”, ou “ruim”, então encaminhe para o gerente;
 - Se o e-mail não conter nenhuma das palavras citadas acima, então encaminhe para o call center;

PADRÕES DE PROJETO

Cadeia de Responsabilidade:



PADRÕES DE PROJETO

Cadeia de Responsabilidade:

```
public interface EmailHandler {  
    public void setNextHandler(EmailHandler handler);  
    public void processHandler(String email);  
}
```

```
public abstract class AbstractEmailHandler implements EmailHandler {
    private EmailHandler nextHandler;

    public void setNextHandler(EmailHandler handler) {
        nextHandler = handler;
    }

    public void processHandler(String email) {
        boolean wordFound = false;

        // If no words to match against then this object can handle
        if (matchingWords().length == 0) {
            wordFound = true;

        } else {
            // Look for any of the matching words
            for (String word : matchingWords()) {
                if (email.indexOf(word) >= 0) {
                    wordFound = true;
                    break;
                }
            }
        }

        // Can we handle email in this object?
        if (wordFound) {
            handleHere(email);
        } else {
            // Unable to handle here so forward to next in chain
            nextHandler.processHandler(email);
        }
    }

    protected abstract String[] matchingWords();
    protected abstract void handleHere(String email);
}
```

Se achou ele para de procurar
e executa o handler

PADRÕES DE PROJETO

```
public class SalesEmailHandler extends AbstractEmailHandler {
    protected String[] matchingWords() {
        return new String[]{"buy", "purchase"};
    }

    protected void handleHere(String email) {
        System.out.println("Email handled by sales department.");
    }
}

public class ServiceEmailHandler extends AbstractEmailHandler {
    protected String[] matchingWords() {
        return new String[]{"service", "repair"};
    }

    protected void handleHere(String email) {
        System.out.println("Email handled by service department.");
    }
}
```



PADRÕES DE PROJETO

É necessário definir a sequência dos chamados:

```
public static void handle(String email) {  
    // Create the handler objects...  
    EmailHandler spam = new SpamEmailHandler();  
    EmailHandler sales = new SalesEmailHandler();  
    EmailHandler service = new ServiceEmailHandler();  
    EmailHandler manager = new ManagerEmailHandler();  
    EmailHandler general = new GeneralEnquiriesEmailHandler();  
    // Chain them together...  
    spam.setNextHandler(sales);  
    sales.setNextHandler(service);  
    service.setNextHandler(manager);  
    manager.setNextHandler(general);  
    // Start the ball rolling...  
    spam.processHandler(email);  
}
```

AbstractEmailHandler



PADRÕES DE PROJETO

```
String email = "I need my car repaired.";  
AbstractEmailHandler.handle(email);
```



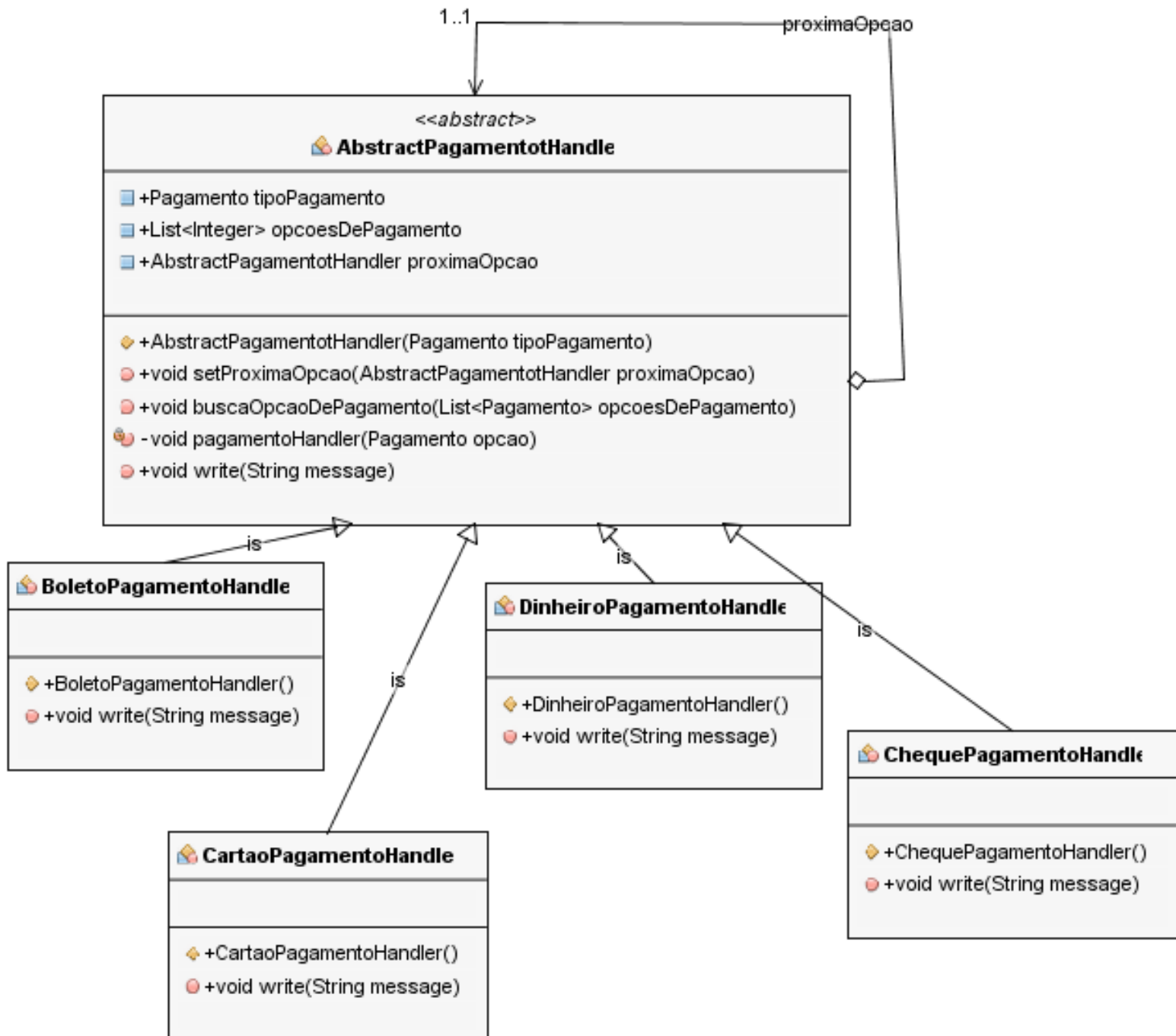
```
Email handled by service department.
```

Exemplo - Pagamentos

Em um sistema de pagamentos, o cliente pode pagar com cartão, cheque, boleto e dinheiro ou uma combinação deles. Vamos utilizar o padrão cadeia de responsabilidade para resolver esse problema.

Ex. Valor do pagamento R\$100,00

Cliente paga com R\$50,00 em dinheiro, parcela no cartão os outros R\$20,00 e R\$ 30,00 em cheque .



```

public abstract class AbstractPagamentoHandler {
    public Pagamento tipoPagamento;
    public List<Integer> opcoesDePagamento;
    public AbstractPagamentoHandler proximaOpcao;
    public enum Pagamento { CARTAO, CHEQUE, DINHEIRO, BOLETO }
    public AbstractPagamentoHandler(Pagamento tipoPagamento) {
        this.tipoPagamento = tipoPagamento;
    }
    public void setProximaOpcao(AbstractPagamentoHandler proximaOpcao) {
        this.proximaOpcao = proximaOpcao;
    }
    public void buscaOpcaoDePagamento(List<Pagamento> opcoesDePagamento) {
        for (Pagamento opcao : opcoesDePagamento) {
            pagamentoHandler(opcao);
        }
    }
    private void pagamentoHandler(Pagamento opcao) {
        if (tipoPagamento == opcao) {
            write("Opção escolhida:");
        } else {
            proximaOpcao.pagamentoHandler(opcao);
        }
    }
    public abstract void write(String message);
}

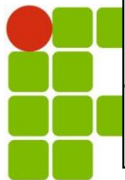
```

Lista com opções
de pagamentos



```
public class BoletoPagamentoHandler extends AbstractPagamentotHandler{  
  
    public BoletoPagamentoHandler() {  
        super(Pagamento.BOLETO);  
    }  
  
    @Override  
    public void write(String message) {  
        System.out.println(message);  
        System.out.println("Pagamento recebido.\nForma: Boleto.\n");  
    }  
}
```

```
public class CartaoPagamentoHandler extends AbstractPagamentotHandler{  
  
    public CartaoPagamentoHandler() {  
        super(Pagamento.CARTAO);  
    }  
  
    @Override  
    public void write(String message) {  
        System.out.println(message);  
        System.out.println("Pagamento recebido.\nForma: Cartão.\n");  
    }  
}
```



```
public class ChequePagamentoHandler extends AbstractPagamentotHandler{

    public ChequePagamentoHandler() {
        super(Pagamento.CHEQUE);
    }

    @Override
    public void write(String message) {
        System.out.println(message);
        System.out.println("Pagamento recebido.\nForma: Cheque.\n");
    }
}
```

```
public class DinheiroPagamentoHandler extends AbstractPagamentotHandler{

    public DinheiroPagamentoHandler() {
        super(Pagamento.DINHEIRO);
    }

    @Override
    public void write(String message) {
        System.out.println(message);
        System.out.println("Pagamento recebido.\nForma: Dinheiro.\n");
    }
}
```



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

```
AbstractPagamentotHandler cartao = new CartaoPagamentoHandler();  
AbstractPagamentotHandler boleto = new BoletoPagamentoHandler();  
AbstractPagamentotHandler dinheiro = new DinheiroPagamentoHandler();  
AbstractPagamentotHandler cheque = new ChequePagamentoHandler();
```

```
cartao.setProximaOpcao(boleto);  
boleto.setProximaOpcao(dinheiro);  
dinheiro.setProximaOpcao(cheque);
```

Encadeamento das
possibilidades requisições

```
List<AbstractPagamentotHandler.Pagamento> opcoesDePagamento = new LinkedList<>();
```

```
opcoesDePagamento.add(AbstractPagamentotHandler.Pagamento.DINHEIRO);  
opcoesDePagamento.add(AbstractPagamentotHandler.Pagamento.CARTAO);  
opcoesDePagamento.add(AbstractPagamentotHandler.Pagamento.CHEQUE);
```

```
cartao.buscaOpcaoDePagamento(opcoesDePagamento);
```

Lista de opções
selecionadas pelo usuário

Início do processamento dado
uma lista de opções.

PADRÕES COMPORTAMENTAIS

Observador

PADRÕES DE PROJETO

- Observador:
 - **Propósito:** define uma relação de notificação de um para muitos, ou seja, quando um objeto muda seu estado, outros objetos são notificados de forma automática.
 - **Problema:**
 - A Motores SA decidiu que um alerta sonoro será emitido quando o motorista exceder uma certa velocidade;
 - Outras coisas também podem acontecer dependendo da velocidade: redução de velocidade automática, por exemplo.
 - É importante que se tenha um baixo acoplamento entre os componentes;

Padrão Observer

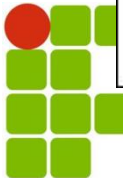
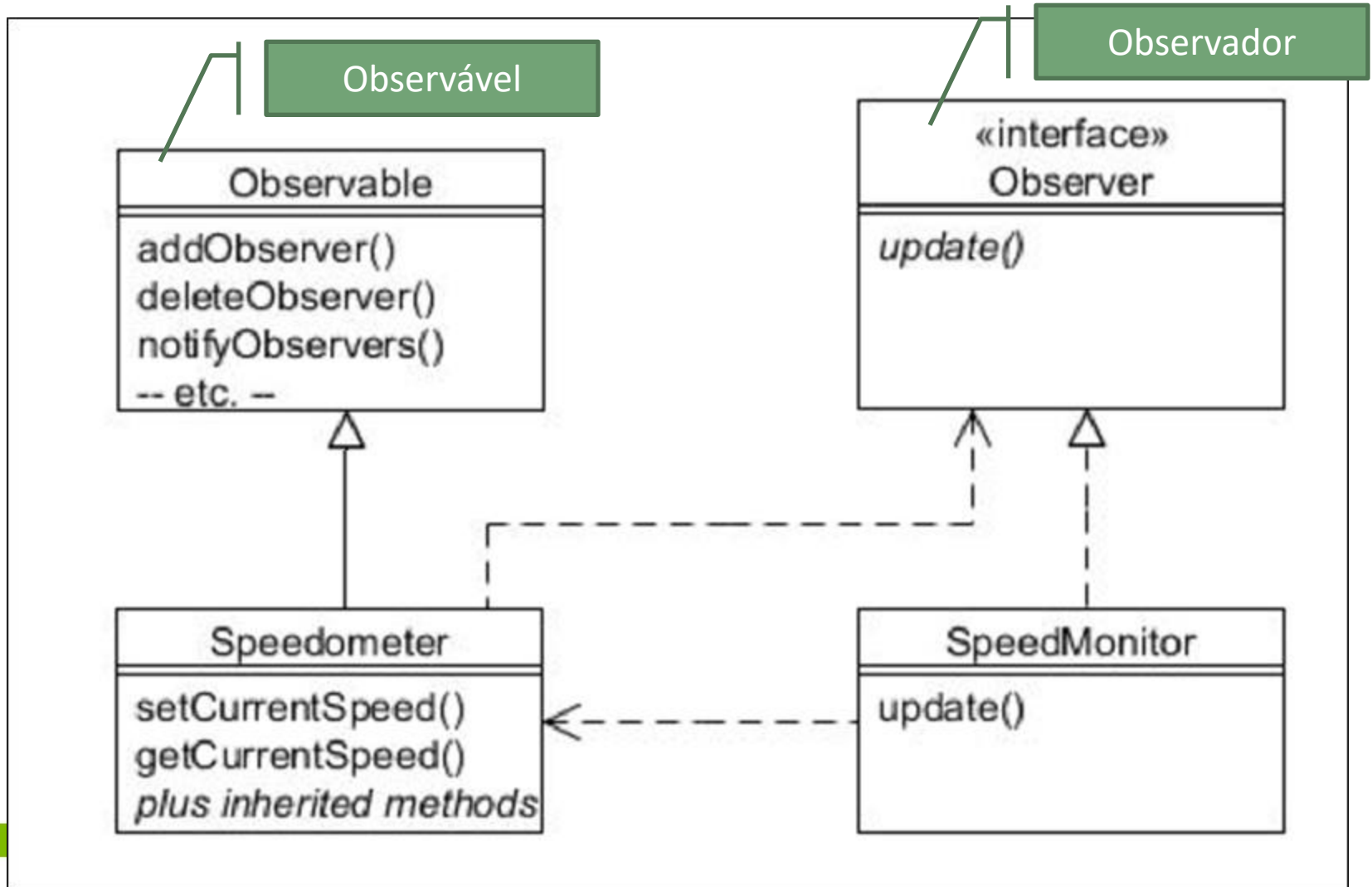
Quando chegar a 70km/hora emitir um aviso sonoro ou desacelerar



Posso também programar uma troca de marcha automática

Velocidade	Marcha
10km/h <=	1
20km/h <=	2
30km/h <=	3
30km/h >	4

PADRÕES DE PROJETO



```
public class Observavel {  
  
    private ArrayList<Observer> monitores = new ArrayList();  
  
    public void addObservador(Observer ob) {  
        this.monitores.add(ob);  
    }  
  
    public void deleteObservador(Observer ob) {  
        monitores.remove(ob);  
    }  
  
    public void notificarObservadores() {  
        for(Observer ob : monitores){  
            ob.update(this);  
        }  
    }  
}
```



Atualiza o observador



PADRÕES DE PROJETO

```
public class Speedometer extends Observable {  
    private int currentSpeed;  
  
    public Speedometer() {  
        speed = 0;  
    }  
  
    public void setCurrentSpeed(int speed) {  
        currentSpeed = speed;  
  
        // Tell all observers so they know speed has changed...  
  
        notifyObservers();  
    }  
  
    public int getCurrentSpeed() {  
        return currentSpeed;  
    }  
}
```



PADRÕES DE PROJETO

```
public class SpeedMonitor implements Observer {
    public static final int SPEED_TO_ALERT = 70;

    public void update(Observable obs ) {
        Speedometer speedo = (Speedometer) obs;
        if (speedo.getCurrentSpeed() > SPEED_TO_ALERT) {
            System.out.println("** ALERT ** Driving too fast! (" + speedo.getCurrentSpeed() + ")");
        } else {
            System.out.println("... nice and steady ... (" + speedo.getCurrentSpeed() + ")");
        }
    }
}
```

PADRÕES DE PROJETO

```
// Create a monitor...
SpeedMonitor monitor = new SpeedMonitor();

// Create a speedometer and register the monitor to it...
Speedometer speedo = new Speedometer();
speedo.addObserver(monitor);

// Drive at different speeds...
speedo.setCurrentSpeed(50);
speedo.setCurrentSpeed(70);
speedo.setCurrentSpeed(40);
speedo.setCurrentSpeed(100);
speedo.setCurrentSpeed(69);
```



```
... nice and steady ... (50)
... nice and steady ... (70)
... nice and steady ... (40)
** ALERT ** Driving too fast! (100)
... nice and steady ... (69)
```

Chama o `notificarTodos`

PADRÕES DE PROJETO

```
public class AutomaticGearbox implements Observer {  
    public void update(Observable obs,          ) {  
        Speedometer speedo = (Speedometer) obs;  
  
        if (speedo.getCurrentSpeed() <= 10) {  
            System.out.println("Now in first gear");  
  
        } else if (speedo.getCurrentSpeed() <= 20) {  
            System.out.println("Now in second gear");  
  
        } else if (speedo.getCurrentSpeed() <= 30) {  
            System.out.println("Now in third gear");  
  
        } else {  
            System.out.println("Now in fourth gear");  
        }  
    }  
}
```



Padrão observador em Jogos

Nos jogos atuais o padrão observador é aplicado para calcular a posição do jogador a cada momento. A estratégia de ataque dos inimigos é calculada com base nos movimentos do personagem que visam cercá-lo por todos os lados. Um exemplo disso acontece no jogo Resident Evil 5, onde os inimigos, que são zumbis, atacam em bandos, cercam o personagem e tentam evitar seus ataques quando possível

O problema acontece quando os objetos dos inimigos precisam receber as informações de movimentação e ataque do jogador, pois qualquer alteração no personagem precisa ser reportada, além disso, a cada mapa, a quantidade de zumbis varia, portanto, não se tem de maneira simples a quantidade de objetos que precisam ser notificados.

Residente Evil 5





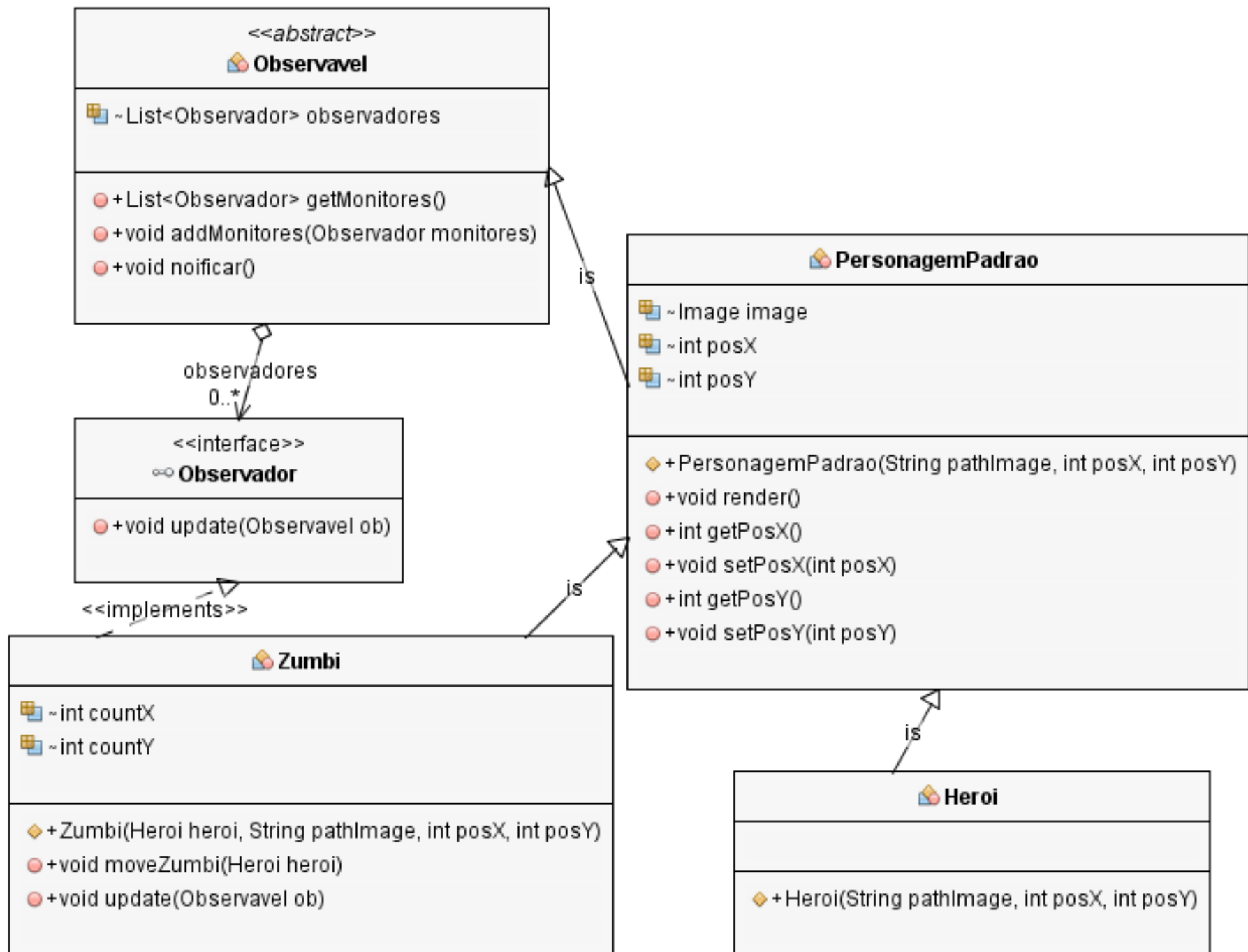
Observadores

Cada movimento do personagem precisa notificar os outros objetos



Observável

Código - Jogo - Padrão Observador



```

public class Observavel {

    List<Observador> observadores = new ArrayList();

    public List<Observador> getMonitores() {
        return observadores;
    }

    public void addMonitores(Observador monitores) {
        this.observadores.add(monitores);
    }

    public void noificarTodos() {
        for(Observador ob : observadores ) {
            ob.update(this);
        }
    }
}

```

Função que notifica todos os seus observadores! Percebam que o update fica por conta do objeto que observa

```
[-] /**
    *
    * @author felipe
    */
public interface Observador {

    public void update(Observavel ob);
}
```

```
class PersonagemPadrao extends Observavel {

    Image image;
    int posX;
    int posY;
    public PersonagemPadrao(String pathImage, int posX, int posY) {
        this.posX = posX;
        this.posY = posY;
        try {
            image = new Image(pathImage);
        } catch (SlickException ex) {
            Logger.getLogger(PersonagemPadrao.class.getName()).log(I
        }
    }
    public Image getImage() {
        return this.image;
    }
    public void render() {
        image.draw(this.posX, this.posY);
    }
    public int getPosX() {
        return posX;
    }
}
```

```
public class Zumbi extends PersonagemPadrao implements Observador{

    public Zumbi(String pathImage, int posX, int posY) {
        super(pathImage, posX, posY);
    }

    @Override
    public void update(Observavel ob) {
        moveZumbi((Heroi) ob);
    }

    public void moveZumbi(Heroi heroi){
        float oposto = heroi.getPosX() - this.getPosX();
        float adjacente = heroi.getPosY() - this.getPosY();
        double hypo = Math.sqrt(Math.pow(oposto, 2) + Math.pow(adjacente, 2));
        float newPosX = (float) (this.getPosX() + 5 * (oposto/hypo));
        float newPosY = (float) (this.getPosY() + 5 * (adjacente/hypo));
        this.setPosX((int) newPosX);
        this.setPosY((int) newPosY );
    }
}
```

Implementa o update
recebendo uma
instância de Heroi

Nova posição baseado
no movimento do heroi

```
/**
 *
 * @author felipe
 */
public class Heroi extends PersonagemPadrao {

    public Heroi(String pathImage, int posX, int posY) {
        super(pathImage, posX, posY);
    }

}
```


@Override

```
public void init(GameContainer gc) throws SlickException {
```

```
    try {
```

```
        File file = new File(".");
```

```
        String filePath = file.getCanonicalPath();
```

```
        land = new Image(filePath + "\\src\\main\\java\\bg.png");
```

```
        heroi = new Heroi(filePath + "\\src\\main\\java\\ator.png",
```

```
        zumbi1 = new Zumbi(filePath + "\\src\\main\\java\\zumbi.png
```

```
        zumbi2 = new Zumbi(filePath + "\\src\\main\\java\\zumbi2.p
```

```
        zumbi3 = new Zumbi(filePath + "\\src\\main\\java\\zumbi1.pn
```

```
        gameover = new Image(filePath + "\\src\\main\\java\\gameove
```

```
        heroi.addMonitores(zumbi1);
```

```
        heroi.addMonitores(zumbi2);
```

```
        heroi.addMonitores(zumbi3);
```

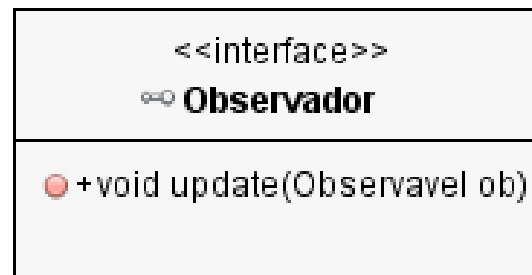
Observadores do heroi
que vão ser atualizados

@Override

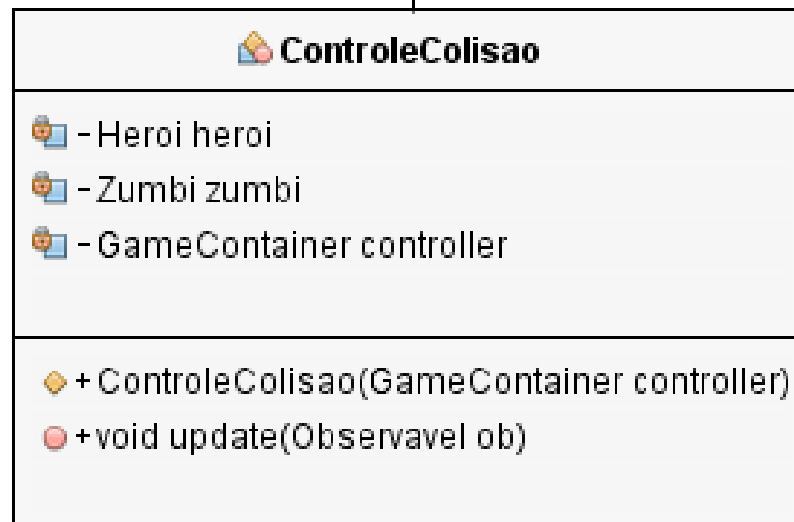
```
public void update(GameContainer gc, int i) throws SlickException {  
    if(!gc.isPaused()){  
        Input input = gc.getInput();  
        if (input.isKeyDown(Input.KEY_UP))  
            this.heroi.setPosY(this.heroi.getPosY() - 1);  
        if (input.isKeyDown(Input.KEY_DOWN))  
            this.heroi.setPosY(this.heroi.getPosY() + 1);  
        if (input.isKeyDown(Input.KEY_LEFT))  
            this.heroi.setPosX(this.heroi.getPosX() - 1);  
        if (input.isKeyDown(Input.KEY_RIGHT))  
            this.heroi.setPosX(this.heroi.getPosX() + 1);  
        timeCount++;  
        if (timeCount % 100 == 0){  
            heroi.noificarTodos();  
        }  
    }  
}
```

Herói notificando
todos os seus
observadores!

Adicionando mais um
Observador para controlar a
colisão e finalizar o jogo caso
isso aconteça!



<<implements>>



```
public class ControleColisao implements Observador {

    private Heroi heroi = null;
    private Zumbi zumbi = null;
    private GameContainer controller;

    public ControleColisao(GameContainer controller) {
        this.controller = controller;
    }

    @Override
    public void update(Observavel ob) {
        if (ob instanceof Heroi)
            this.heroi = (Heroi) ob;
        else if (ob instanceof Zumbi)
            this.zumbi = (Zumbi) ob;
        if (this.heroi != null && this.zumbi != null) {
            Rectangle first = new Rectangle(heroi.getPosX(), heroi.getPo
                heroi.getImage().getWidth() - 20, heroi.getImage().g
            Rectangle second = new Rectangle(zumbi.getPosX(), zumbi.getP
                zumbi.getImage().getWidth() - 20, zumbi.getImage().g
            if (first.intersects(second)) {
                controller.pause();
            }
        }
    }
}
```

Função que vai ser chamada a cada nova update objeto observável

Se os objetos colidem GAME OVER!



```
land = new Image(filePath + "\\src\\main\\java\\bg.png");
heroi = new Heroi(filePath + "\\src\\main\\java\\ator.png", 200, 200);
zumbi1 = new Zumbi(filePath + "\\src\\main\\java\\zumbi.png", 500, 0);
zumbi2 = new Zumbi(filePath + "\\src\\main\\java\\zumbi2.png", 0, 0);
zumbi3 = new Zumbi(filePath + "\\src\\main\\java\\zumbi1.png", 400, 400);
gameover = new Image(filePath + "\\src\\main\\java\\gameover.png");

heroi.addMonitores(zumbi1);
heroi.addMonitores(zumbi2);
heroi.addMonitores(zumbi3);

ControleColisao controleColisao = new ControleColisao(gc);
heroi.addMonitores(controleColisao);
zumbi1.addMonitores(controleColisao);
zumbi2.addMonitores(controleColisao);
zumbi3.addMonitores(controleColisao);
```

Adicionando o observador a todos os objetos da tela

```
@Override
public void update(GameContainer gc, int i) throws SlickException {
    if(!gc.isPaused()){
        Input input = gc.getInput();
        if (input.isKeyDown(Input.KEY_UP))
            this.heroi.setPosY(this.heroi.getPosY() - 1);
        if (input.isKeyDown(Input.KEY_DOWN))
            this.heroi.setPosY(this.heroi.getPosY() + 1);
        if (input.isKeyDown(Input.KEY_LEFT))
            this.heroi.setPosX(this.heroi.getPosX() - 1);
        if (input.isKeyDown(Input.KEY_RIGHT))
            this.heroi.setPosX(this.heroi.getPosX() + 1);
        timeCount++;
        if (timeCount % 100 == 0){
            heroi.noificarTodos();
            zumbi1.noificarTodos();
            zumbi2.noificarTodos();
            zumbi3.noificarTodos();
        }
    }
}
```

Notificando todos os observadores dos objetos.

Percebam que do heroi continuou a mesma coisa, Por quê?

```
@Override
public void render(GameContainer gc, Graphics g) {
    g.drawImage(land, 0, 0);
    heroi.render();
    zumbi2.render();
    zumbi3.render();
    zumbi1.render();

    if(gc.isPaused())
        gameover.draw(200,200);
}
```

Show Game Over

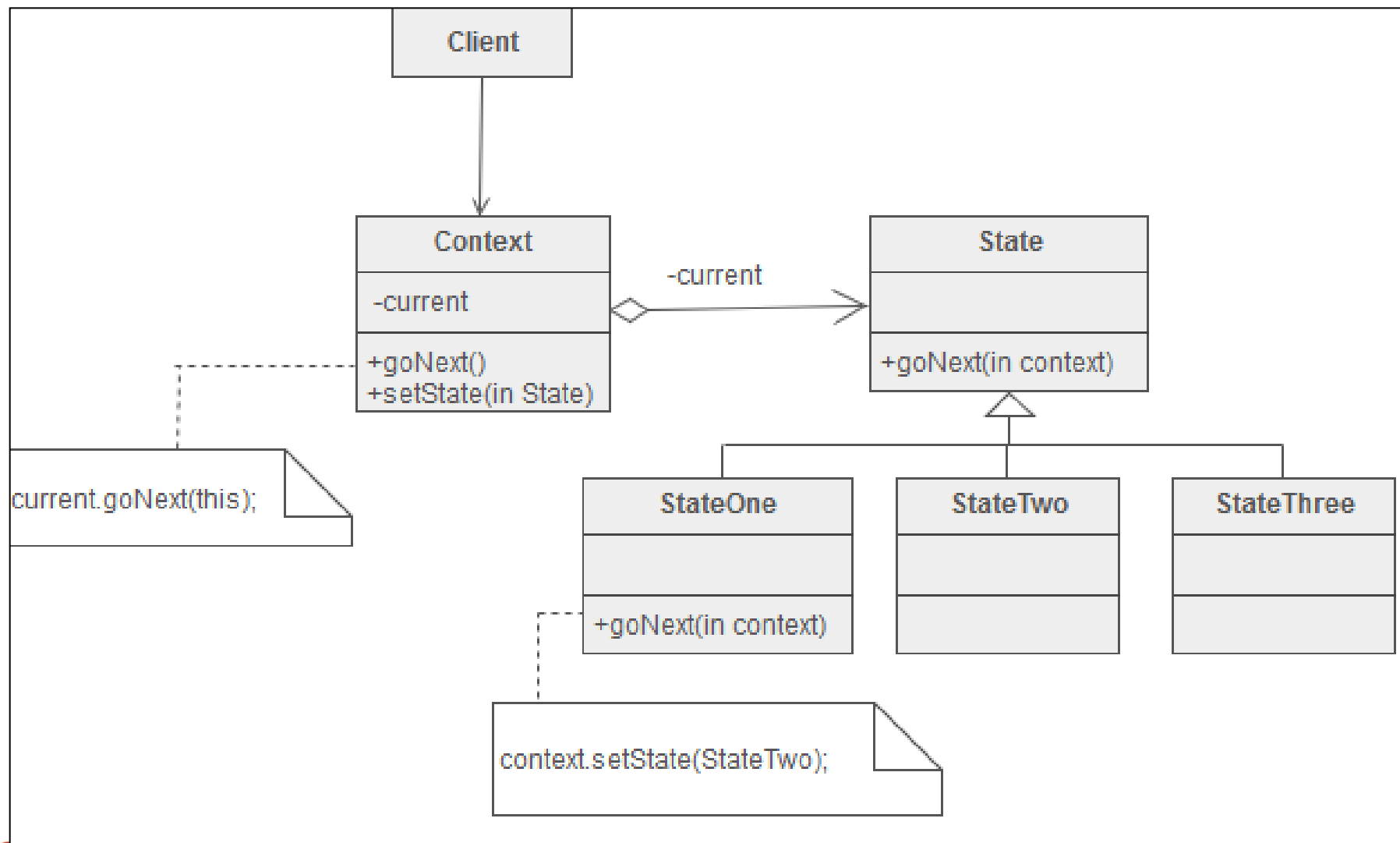
Outro Exemplo de Jogo

[https://github.com/felipefo/poo2/tree/master/Padroes](https://github.com/felipefo/poo2/tree/master/Padroes_de_Projeto/Comportamental/Observer/GameChuven)
[de Projeto/Comportamental/Observer/GameChuven](https://github.com/felipefo/poo2/tree/master/Padroes_de_Projeto/Comportamental/Observer/GameChuven)
[doMonstros](https://github.com/felipefo/poo2/tree/master/Padroes_de_Projeto/Comportamental/Observer/GameChuven)

PADRÕES COMPORTAMENTAIS

PADRÕES DE PROJETO

- State:
 - **Propósito:** permite que o objeto mude o seu comportamento, conforme o seu estado.
 - **Problema:**
 - Todos carros da Motores SA possuem um display que mostra a hora e o data corrente ;
 - O display possui uma alavanca que permite modificar a hora e a data;



PADRÕES DE PROJETO

- State:

- Processo:

- Quando a alavanca é empurrada pela primeira vez, é habilitado a modificação do **ano**;
 - Quando a alavanca é rodada para a esquerda o **ano** é deduzido de uma unidade;
 - Quando a alavanca é rodada para a direita o **ano** é acrescido de uma unidade;
 - Quando a alavanca é empurrada novamente o **ano** é salvo e display habilita a modificação do **mês**;

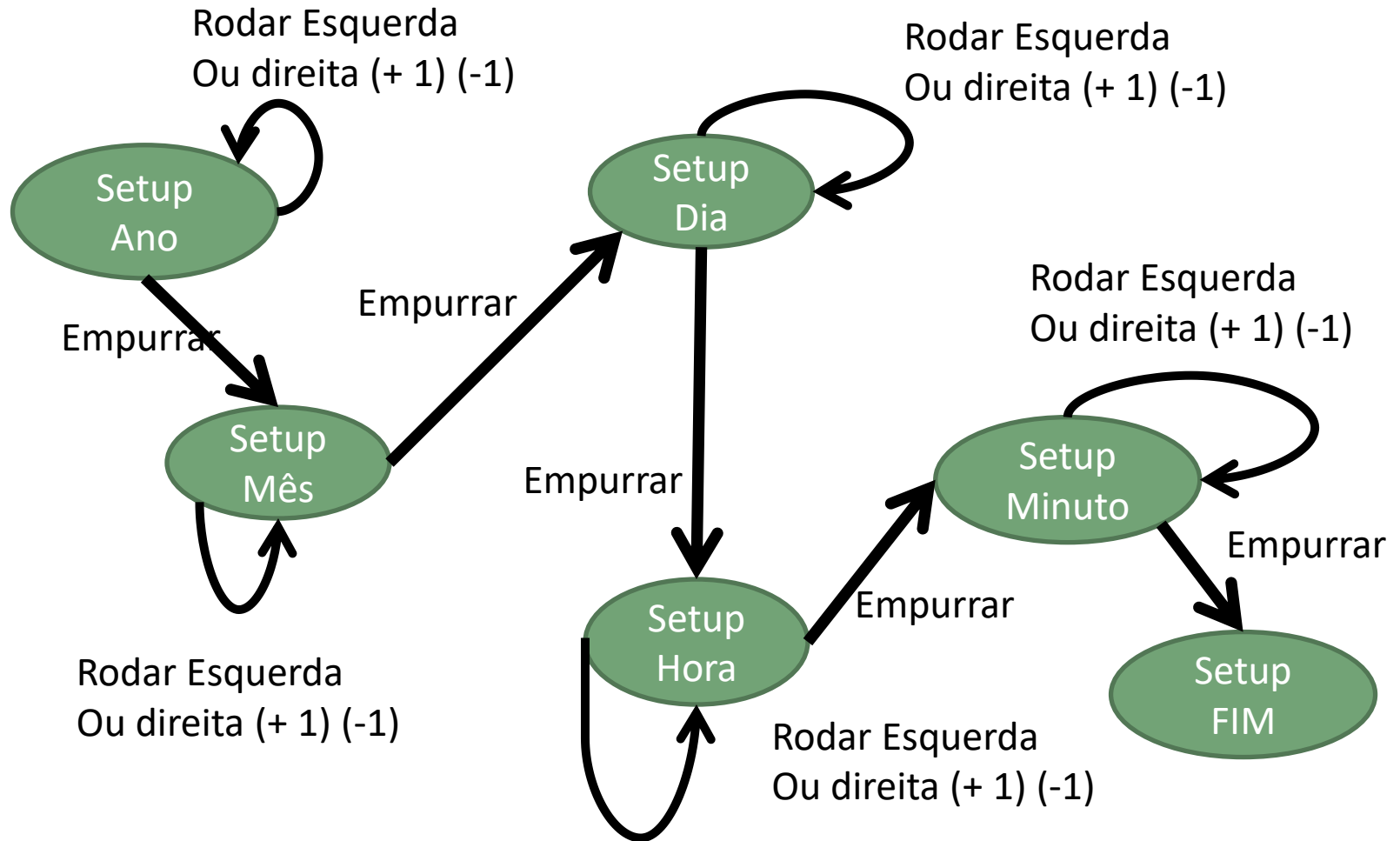
PADRÕES DE PROJETO

- State:
 - Processo:
 - Quando a alavanca é rodada para a esquerda o **mês** é deduzido de uma unidade;
 - Quando a alavanca é rodada para a direita o **mês** é acrescido de uma unidade;
 - Quando a alavanca é empurrada novamente o **mês** é salvo e display habilita a modificação do **dia**;
 - Quando a alavanca é rodada para a esquerda o **dia** é deduzido de uma unidade;
 - Quando a alavanca é rodada para a direita o **dia** é acrescido de uma unidade;

PADRÕES DE PROJETO

- State:
 - Processo:
 - Quando a alavanca é empurrada novamente o **dia** é salvo e display habilita a modificação da **hora**;
 - Quando a alavanca é rodada para a esquerda a **hora** é deduzida de uma unidade;
 - Quando a alavanca é rodada para a direita a **hora** é acrescida de uma unidade;
 - Quando a alavanca é empurrada novamente a **hora** é salvo e display habilita a modificação da **minuto**;
 - Quando a alavanca é rodada para a esquerda o **minuto** é deduzido de uma unidade;
 - Quando a alavanca é rodada para a direita o **minuto** é acrescido de uma unidade;

Estados

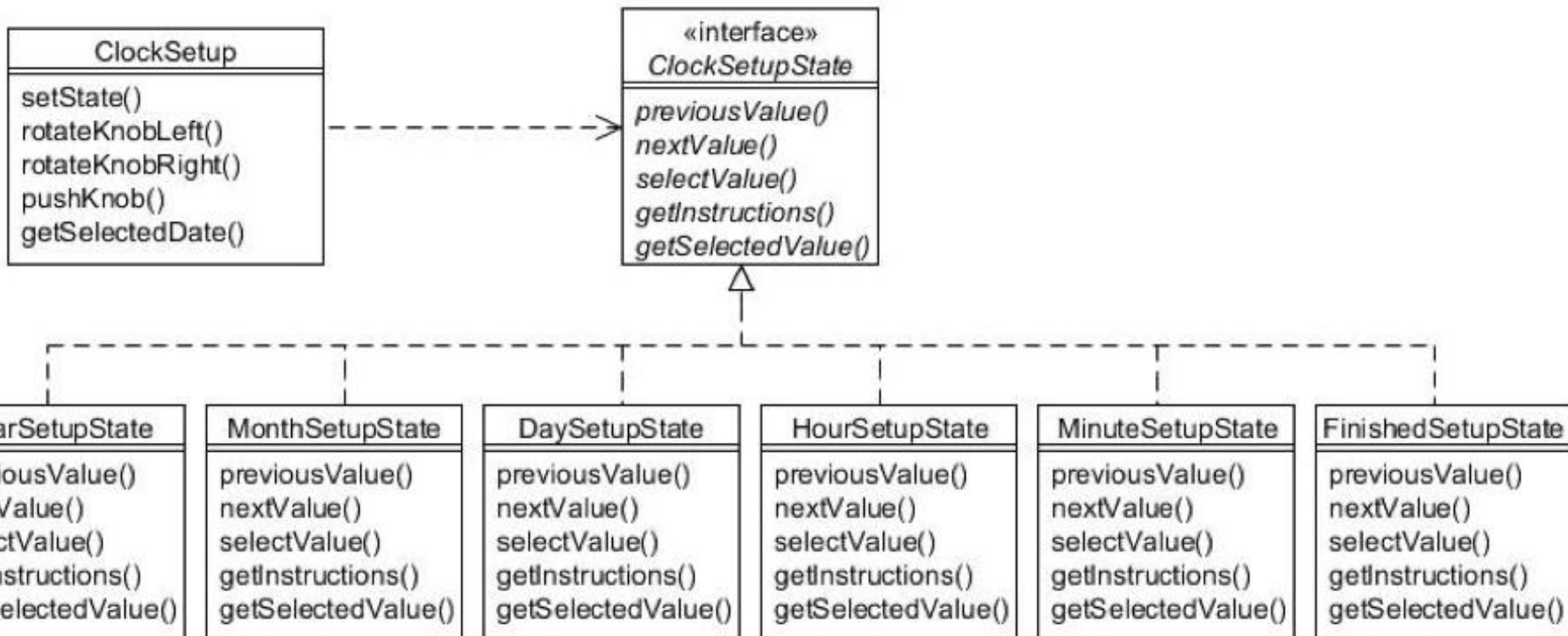


PADRÕES DE PROJETO

- State:
 - **Processo:**
 - Quando a alavanca é puxada novamente os dados são salvos, e a data e as horas são apresentadas no display;
 - **Como fazemos isso tudo sem usar If e else?**

PADRÕES DE PROJETO

- State:



```
public class YearSetupState implements ClockSetupState {
    private ClockSetup clockSetup;
    private int year;

    public YearSetupState(ClockSetup clockSetup) {
        this.clockSetup = clockSetup;
        year = Calendar.getInstance().get(Calendar.YEAR);
    }

    public void previousValue() {
        year--;
    }

    public void nextValue() {
        year++;
    }
```

Rodar a
alavanca

```
public void selectValue() {
    System.out.println("Year set to " + year);
    clockSetup.setState(clockSetup.getMonthSetupState());
}
```

```
public String getInstructions() {
    return "Please set the year...";
}

public int getSelectedValue() {
    return year;
}
```

Empurra a
alavanca



```

public class DaySetupState implements ClockSetupState {
    private ClockSetup clockSetup;
    private int day;

    public DaySetupState(ClockSetup clockSetup) {
        this.clockSetup = clockSetup;
        day = Calendar.getInstance().get(Calendar.DAY_OF_MONTH);
    }

    public void previousValue() {
        if (day > 1) {
            day--;
        }
    }

    public void nextValue() {
        if (day < Calendar.getInstance().getActualMaximum(Calendar.DAY_OF_MONTH)) {
            day++;
        }
    }

    public void selectValue() {
        System.out.println("Day set to " + day);
        clockSetup.setState(clockSetup.getHourSetupState());
    }

    public String getInstructions() {
        return "Please set the day...";
    }

    public int getSelectedValue() {
        return day;
    }
}

```

Rodar a
alavanca

Empurra a
alavanca

```

public ClockSetup() {
    yearState = new YearSetupState(this);
    monthState = new MonthSetupState(this);
    dayState = new DaySetupState(this);
    hourState = new HourSetupState(this);
    minuteState = new MinuteSetupState(this);
    finishedState = new FinishedSetupState(this);

    // Initial state is to set the year
    setState(yearState);
}

public void setState(ClockSetupState state) {
    currentState = state;
    System.out.println(currentState.getInstructions());
}

public void rotateKnobLeft() {
    currentState.previousValue();
}

public void rotateKnobRight() {
    currentState.nextValue();
}

public void pushKnob(){
    currentState.selectedValue();
}

```

Modifica os
valores do estado
atual

Empurra a
alavanca

```
ClockSetup clockSetup = new ClockSetup();
```

```
// Setup starts in 'year' state  
clockSetup.rotateKnobRight();  
clockSetup.pushKnob(); // 1 year on
```

```
// Setup should now be in 'month' state  
clockSetup.rotateKnobRight();  
clockSetup.rotateKnobRight();  
clockSetup.pushKnob(); // 2 months on
```

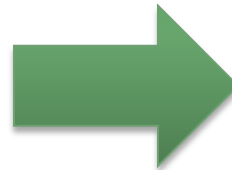
```
// Setup should now be in 'day' state  
clockSetup.rotateKnobRight();
```

```
clockSetup.rotateKnobRight();  
clockSetup.rotateKnobRight();  
clockSetup.pushKnob(); // 3 days on
```

```
// Setup should now be in 'hour' state  
clockSetup.rotateKnobLeft();  
clockSetup.rotateKnobLeft();  
clockSetup.pushKnob(); // 2 hours back
```

```
// Setup should now be in 'minute' state  
clockSetup.rotateKnobRight();  
clockSetup.pushKnob(); // 1 minute on
```

```
// Setup should now be in 'finished' state  
clockSetup.pushKnob(); // to display selected date
```



Please set the year...

Year set to 2013

Please set the month...

Month set to 10

Please set the day...

Day set to 25

Please set the hour...

Hour set to 0

Please set the minute...

Minute set to 4

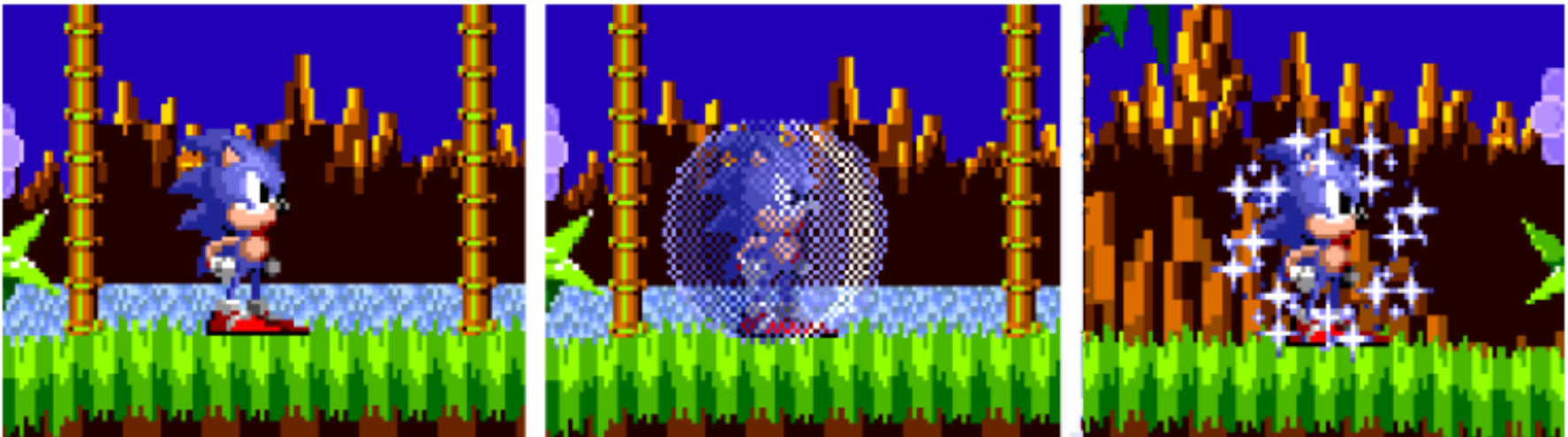
Press knob to view selected date...

Date set to: Mon Nov 25 04:17:00 GMT 2013

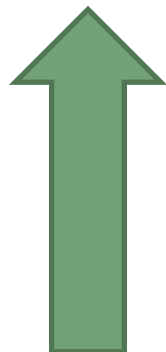


Estado em Jogos

No jogo *Sonic*, o personagem principal ao esbarrar em um inimigo poderá ter diferentes consequências de acordo com sua característica atual.



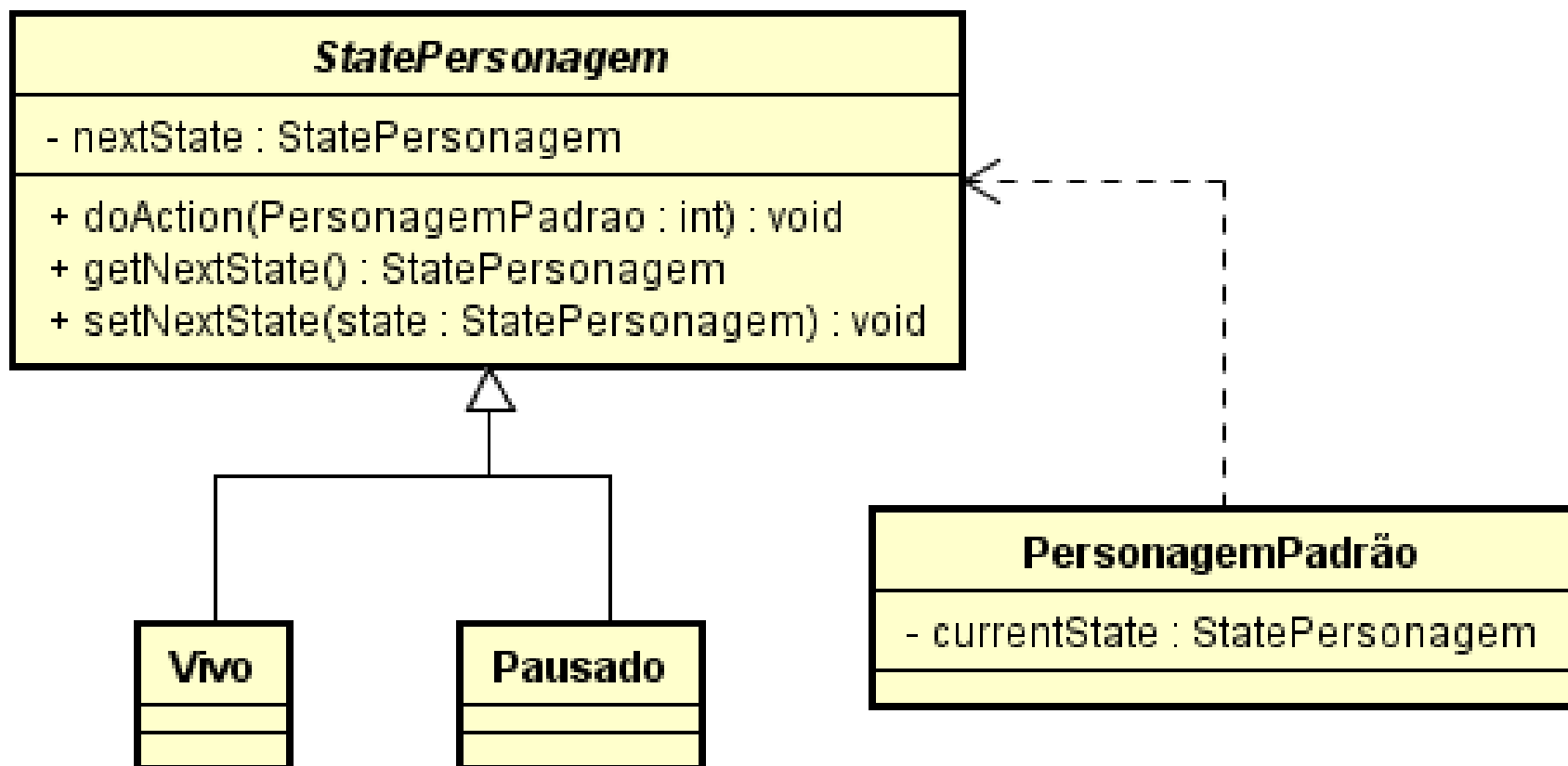
Figura| 5-49: (a) Sonic normal (b) Sonic com escudo (c) Sonic com invencibilidade temporária



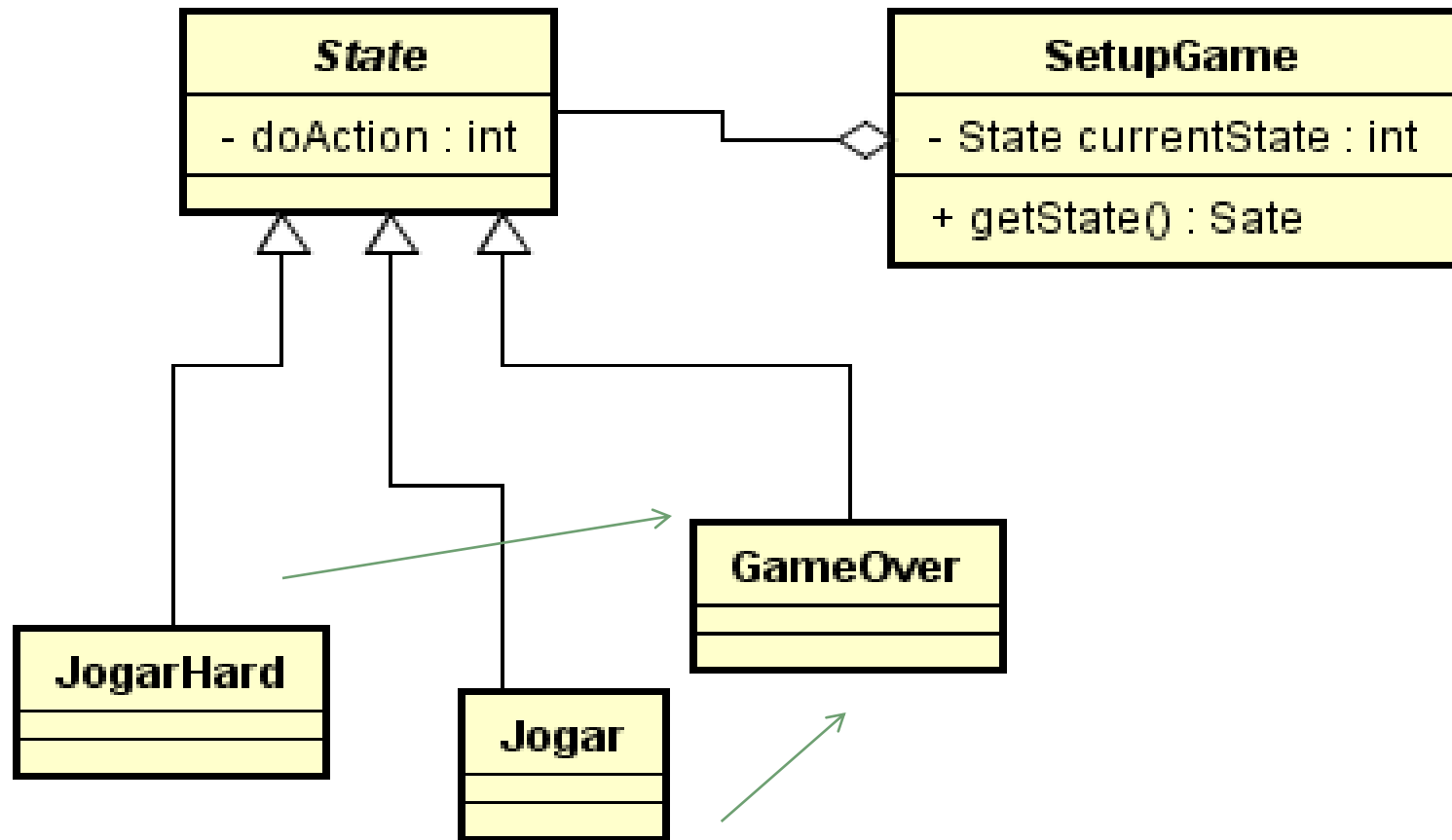
Se ele encostar no
zumbi do lado
direito o zumbi vai
pausar, ou seja seu
estado vai mudar



Código Jogo - Padrão Estado



Estado geral do Jogo



```
public abstract class StatePersonagem {  
    StatePersonagem nextState;  
  
    public abstract void doAction(PersonagemPadrao personagem);  
    public StatePersonagem getNextState() {  
        return nextState;  
    }  
    public void setNextState(StatePersonagem nextState) {  
        this.nextState = nextState;  
    }  
}
```

```
public class Pausado extends StatePersonagem {  
    private int count = 0;  
    private int x = -1;  
    private int y = -1;  
    @Override  
    public void doAction(PersonagemPadrao personagem) {  
        if (x == -1) {  
            this.x = personagem.getPosX();  
            this.y = personagem.getPosY();  
        }  
        count++;  
        if (count == 5000) {  
            //saindo do modo pausado.  
            personagem.goNextState();  
            count = 0;  
        }  
        personagem.getImage().draw(this.x, this.y);  
    }  
}
```



```

/**
 *
 * @author felip
 */
public class Vivo extends StatePersonagem {

    @Override
    public void doAction(PersonagemPadrao personagem) {
        personagem.getImage().draw(personagem.getPosX(), personagem.getPosY())
    }
}

```

Classe Controle de Colisão

```
if (this.heroi != null && this.zumbi != null) {  
    Rectangle first = new Rectangle(heroi.getPosX() + heroi.getImage().getWidth() - 10,  
        heroi.getImage().getWidth() - 5, heroi.getImage().getHeight());  
  
    Rectangle second = new Rectangle(zumbi.getPosX() + zumbi.getImage().getWidth()/2, z  
        zumbi.getImage().getWidth(), zumbi.getImage().getHeight());  
    if (first.intersects(second)) {  
        if(! (zumbi.getCurrentState().instanceof Pausado))  
            zumbi.goNextState();  
        return;  
    }  
}  
  
if (this.heroi != null && this.zumbi != null) {  
    Rectangle first = new Rectangle(heroi.getPosX(), heroi.getPosY(),  
        heroi.getImage().getWidth(), heroi.getImage().getHeight());  
    Rectangle second = new Rectangle(zumbi.getPosX() - 10, zumbi.getPosY(),  
        zumbi.getImage().getWidth() - zumbi.getImage().getWidth()/2 - 10, zumbi.getI  
    if (first.intersects(second)) {  
        gameState.setNextState();  
    }  
}
```

No controle de colisão,
colidindo pelo lado direito =
Pausar Zumbi

No controle de colisão,
colidindo pelo lado
esquerdo = GameOver

```

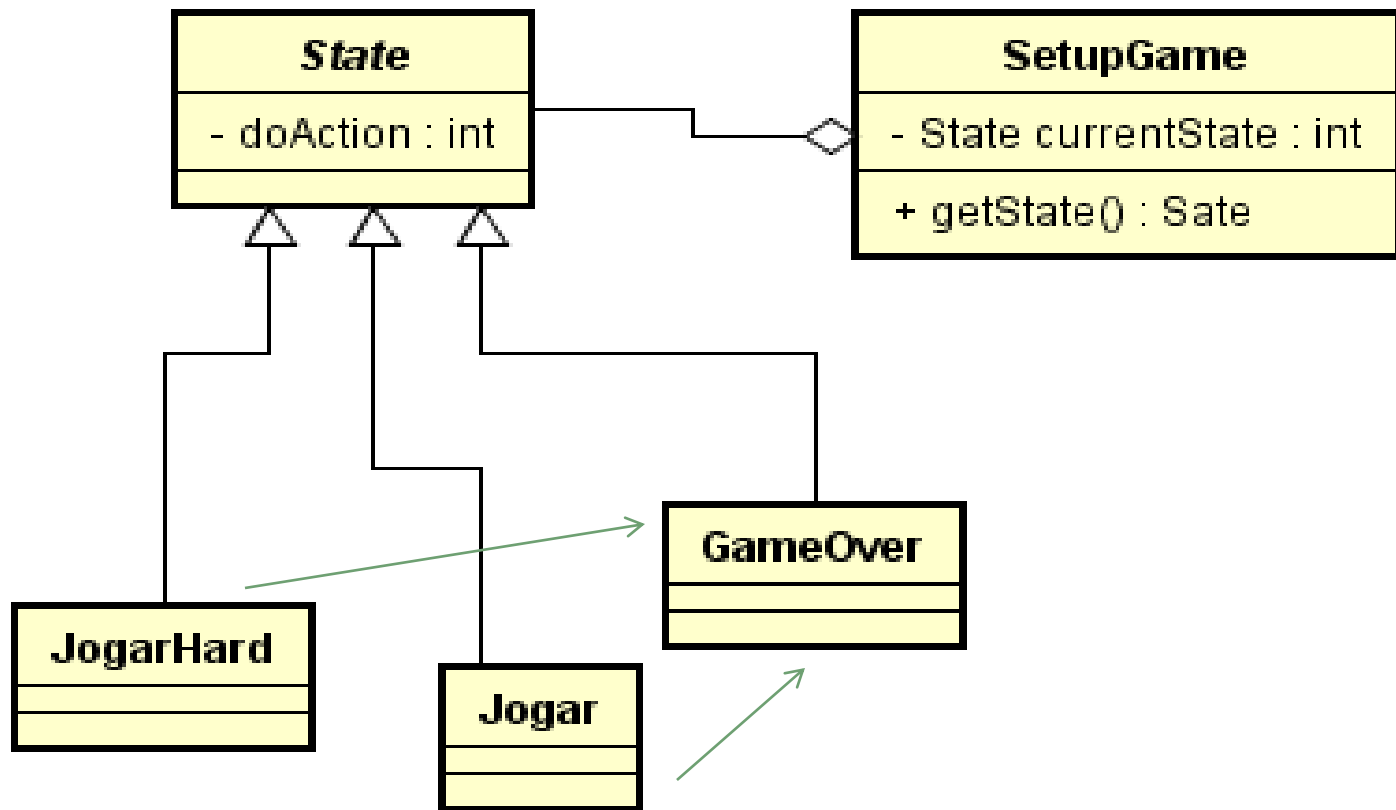
@SuppressWarnings("OverridableMethodCallInConstructor")
public PersonagemPadrao(String pathImage, int posX, int posY){
    this.posX = posX;
    this.posY = posY;
    try {
        image = new Image(pathImage);
    } catch (SlickException ex) {
        Logger.getLogger(PersonagemPadrao.class.getName()).log(Level.SEVERE, null, ex);
    }
    vivo.setNextState(pausado);
    pausado.setNextState(vivo);
    setCurrentState( vivo );
}

```

Definindo os
estados do meu
personagem

Estado geral do Jogo

No nosso jogo de Zumbis



```
public class Jogar extends State{
```

```
    private int timeInterval = 100;
```

```
    private int timeCount=0;
```

```
    @Override
```

```
    public void doAction(GameContainer gc, SimpleSlickGame sl, int i) {
```

```
        Input input = gc.getInput();
```

```
        if (input.isKeyDown(Input.KEY_UP))
```

```
            sl.heroi.setPosY(sl.heroi.getPosY() - 1);
```

```
        if (input.isKeyDown(Input.KEY_DOWN))
```

```
            sl.heroi.setPosY(sl.heroi.getPosY() + 1);
```

```
        if (input.isKeyDown(Input.KEY_LEFT))
```

```
            sl.heroi.setPosX(sl.heroi.getPosX() - 1);
```

```
        if (input.isKeyDown(Input.KEY_RIGHT))
```

```
            sl.heroi.setPosX(sl.heroi.getPosX() + 1);
```

```
        if (timeCount == timeInterval){
```

```
            sl.heroi.noificarTodos();
```

```
            sl.zumbi1.noificarTodos();
```

```
            sl.zumbi2.noificarTodos();
```

```
            sl.zumbi3.noificarTodos();
```

```
            timeCount =0;
```

```
        }
```

```
        timeCount++;
```

```
    }
```

Velocidade do jogo



```
public abstract class State {  
  
    State nextState;  
  
    public abstract void doAction(GameContainer gc, SimpleSlickGame sl, int i);  
    public State getNextState() {  
        return this.nextState;  
    }  
  
    public void setNextState(State state) {  
        this.nextState = state;  
    }  
}
```



```
public class JogarHard extends State{
```

```
    private int timeInterval = 10;  
    private int timeCount=0;
```

Velocidade do jogo Hard

```
@Override
```

```
public void doAction(GameContainer gc, SimpleSlickGame sl, int i) {
```

```
    Input input = gc.getInput();  
    if (input.isKeyDown(Input.KEY_UP))  
        sl.heroi.setPosY(sl.heroi.getPosY() - 1);  
    if (input.isKeyDown(Input.KEY_DOWN))  
        sl.heroi.setPosY(sl.heroi.getPosY() + 1);  
    if (input.isKeyDown(Input.KEY_LEFT))  
        sl.heroi.setPosX(sl.heroi.getPosX() - 1);  
    if (input.isKeyDown(Input.KEY_RIGHT))  
        sl.heroi.setPosX(sl.heroi.getPosX() + 1);  
  
    if (timeCount == timeInterval){  
        sl.heroi.noificarTodos();  
        sl.zumbi1.noificarTodos();  
        sl.zumbi2.noificarTodos();  
        sl.zumbi3.noificarTodos();  
        timeCount = 0;  
    }  
    timeCount++;
```



```
/**
 *
 * @author felipe
 */
public class GameOver extends State {

    @Override
    public void doAction(GameContainer gc, SimpleSlickGame sl, int i) {
        gc.pause();
    }

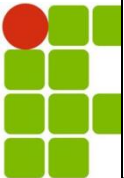
}
```

Fim de jogo

Como era antes (SimpleSlickGame)

```
@Override
public void update(GameContainer gc, int i) throws SlickException {
    if(!gc.isPaused()){
        Input input = gc.getInput();
        if (input.isKeyDown(Input.KEY_UP))
            this.heroi.setPosY(this.heroi.getPosY() - 1);
        if (input.isKeyDown(Input.KEY_DOWN))
            this.heroi.setPosY(this.heroi.getPosY() + 1);
        if (input.isKeyDown(Input.KEY_LEFT))
            this.heroi.setPosX(this.heroi.getPosX() - 1);
        if (input.isKeyDown(Input.KEY_RIGHT))
            this.heroi.setPosX(this.heroi.getPosX() + 1);
        timeCount++;
        if (timeCount % 100 == 0){
            heroi.noificarTodos();
            zumbi1.noificarTodos();
            zumbi2.noificarTodos();
            zumbi3.noificarTodos();
        }
    }
}
```

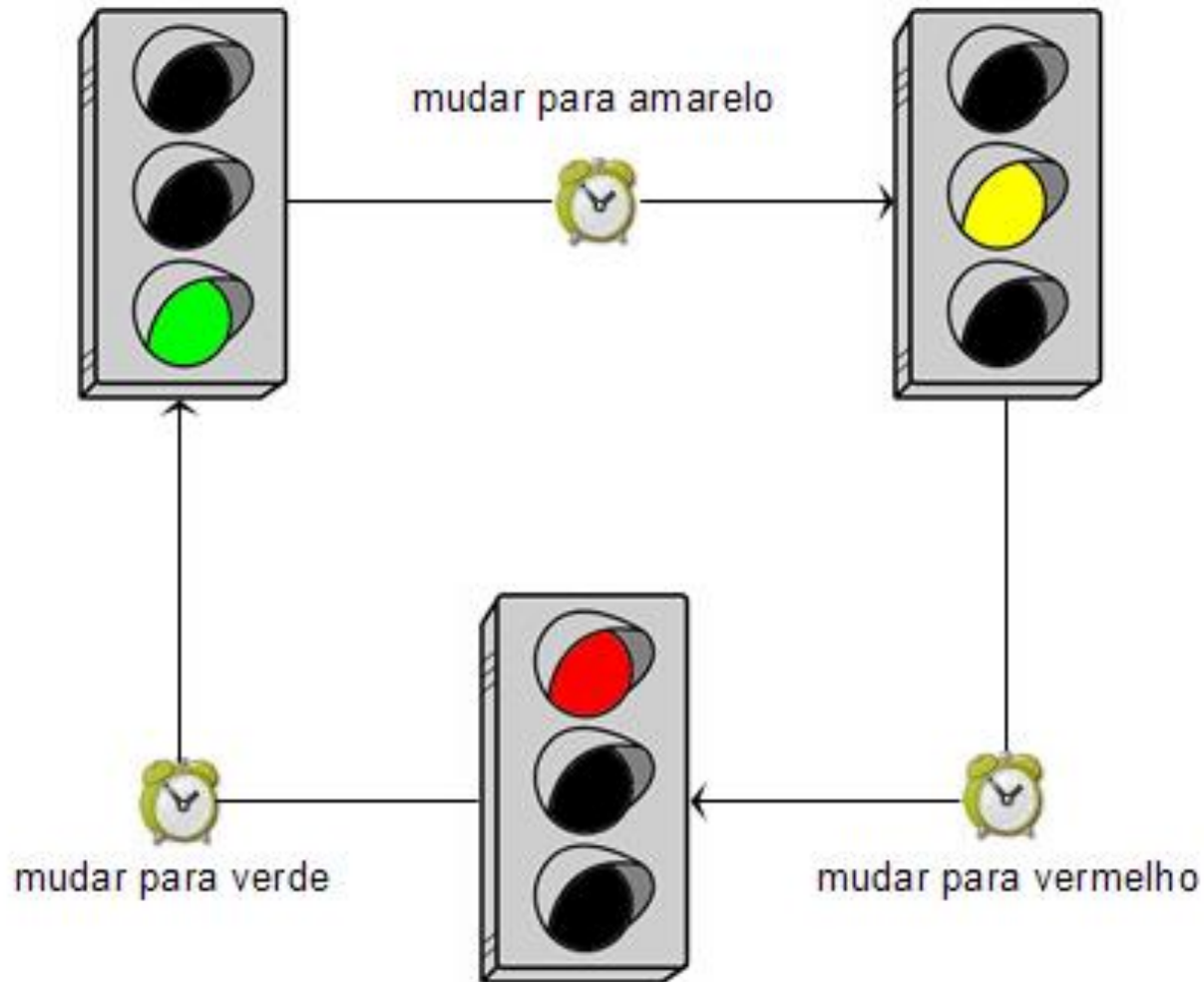
Ação



Como ficou....

```
@Override
public void update(GameContainer gc, int i) throws SlickException {
    if (state != null) {
        if (state.getState() != null) {
            state.getState().doAction(gc, this, i);
        }
    }
}
```

Implemente problema de semáforos abaixo utilizando o padrão State. Faça também o diagrama de classe



PADRÕES COMPORTAMENTAIS

Comando

PADRÕES DE PROJETO

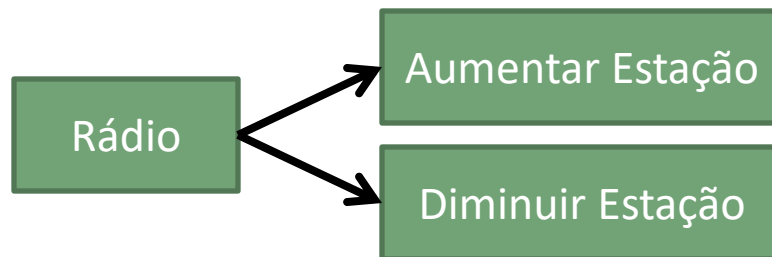
- Comando:
 - **Propósito:** encapsula uma requisição como um objeto, deixando que o cliente parametrize as requisições. Facilita a adição de novos comandos e permite guardar um histórico de execução.
 - **Problema:**
 - Todos os carros da Motores SA possuem rádio instalado e vidro elétricos.
 - A empresa decidiu adicionar o controle de voz nos veículos de luxo;

PADRÕES DE PROJETO

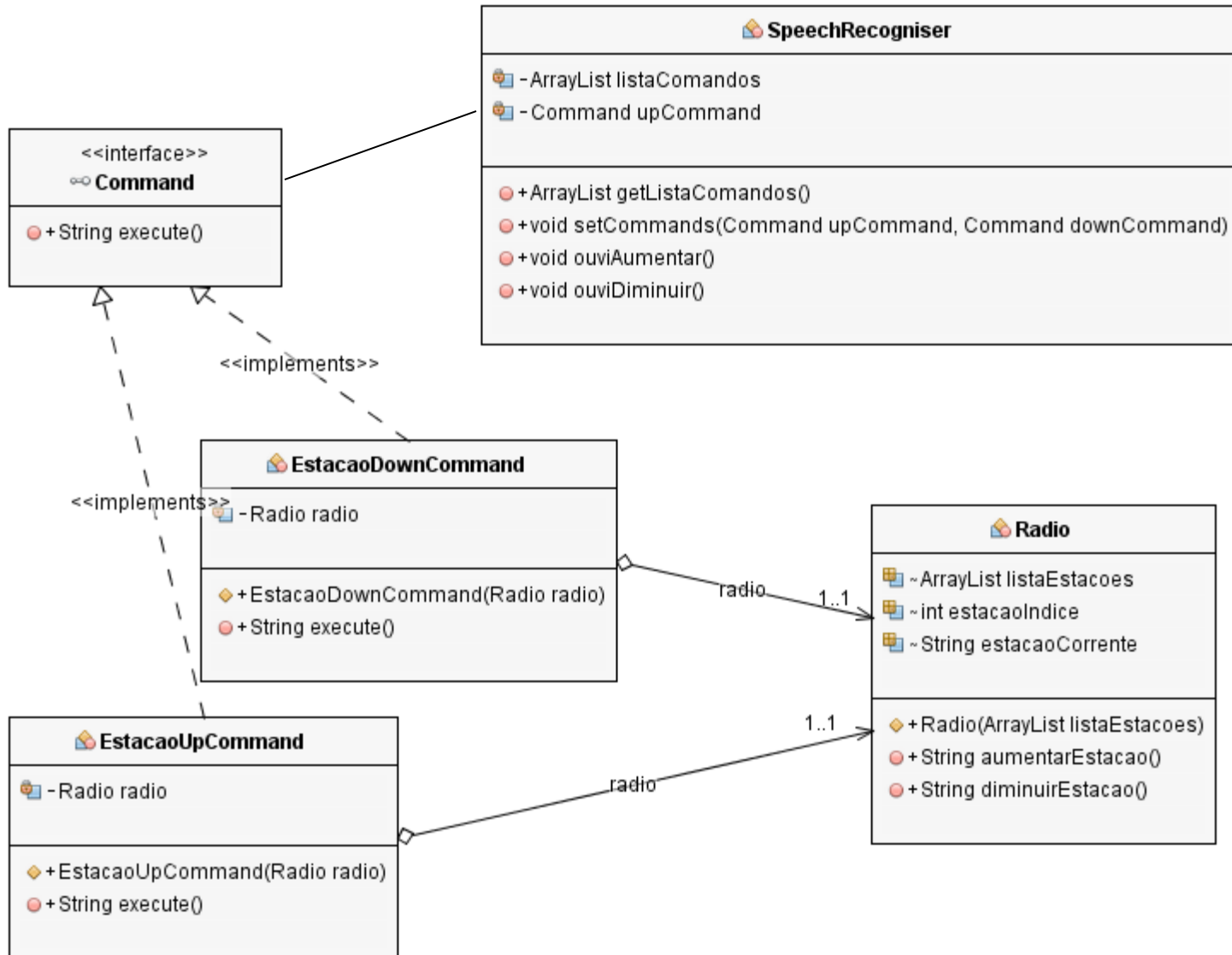
Comando:

– Problema:

- A empresa decidiu adicionar o controle de voz nos veículos de luxo;
- Se o usuário falar “Rádio” e falar “aumentar” e “diminuir” o sistema deve aumentar a estação ou diminuir a estação do rádio, respectivamente;



PADRÕES DE PROJETO



PADRÕES DE PROJETO

```
public class Radio {  
    ArrayList listaEstacoes = new ArrayList();  
    int estacaoIndice = 0;  
    String estacaoCorrente;  
    int volume = 0;  
    public Radio(ArrayList listaEstacoes) {  
        this.listaEstacoes = listaEstacoes;  
    }  
    public void aumentarEstacao() {  
        estacaoIndice++;  
        estacaoCorrente = (String) listaEstacoes.get(estacaoIndice);  
    }  
    public void diminuirEstacao() {  
        estacaoIndice--;  
        estacaoCorrente = (String) listaEstacoes.get(estacaoIndice);  
    }  
    public void aumentarVolume() { volume++; }  
    public void diminuirVolume() { volume--; }  
}
```

Aumentando uma estação de rádio

Diminuindo uma estação de rádio

Aumentar/Diminuir Volume



PADRÕES DE PROJETO

```
public class EstacaoUpCommand implements Command {  
  
    private Radio radio;  
    public EstacaoUpCommand(Radio radio) {  
        this.radio = radio;  
    }  
    @Override  
    public void execute() {  
        this.radio.aumentarEstacao();  
    }  
}
```

Comando padrão
execute

```
public class EstacaoDownCommand implements Command {  
  
    private Radio radio;  
    public EstacaoDownCommand(Radio radio) {  
        this.radio = radio;  
    }  
    @Override  
    public void execute() {  
        this.radio.diminuirEstacao();  
    }  
}
```

Comando padrão
execute

PADRÕES DE PROJETO

Classe que
implementar o
ouvir!

Histórico de
Comandos
Executados

Guardando o
histórico de
comandos
executados

```
public class SpeechRecogniser {  
    private ArrayList listaComandos = new ArrayList();  
    private Command upCommand, downCommand;  
  
    public ArrayList getListaComandos() {  
        return this.listaComandos;  
    }  
  
    public void setCommands(Command upCommand, Command downCommand) {  
        this.upCommand = upCommand;  
        this.downCommand = downCommand;  
    }  
  
    public void ouviAumentar() {  
        this.listaComandos.add(upCommand); {  
            this.upCommand.execute();  
        }  
    }  
  
    public void ouviDiminuir() {  
        this.listaComandos.add(downCommand); {  
            this.downCommand.execute();  
        }  
    }  
}
```

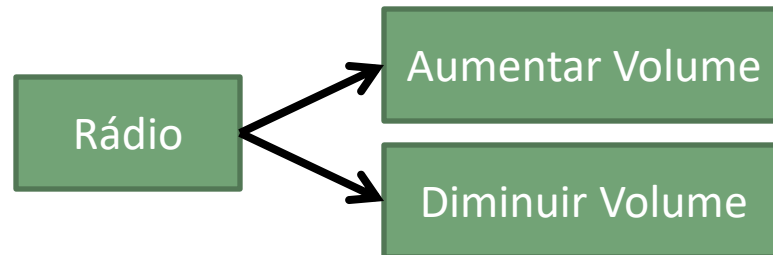


```
public static void main(String[] args) {  
  
    ArrayList listaEstacoes = new ArrayList();  
    listaEstacoes.add("100.1");  
    listaEstacoes.add("92.5");  
    listaEstacoes.add("93.5");  
    Radio radio = new Radio(listaEstacoes);  
    SpeechRecogniser speech = new SpeechRecogniser();  
    speech.setCommands(new EstacaoUpCommand(radio),  
        new EstacaoDownCommand(radio));  
    speech.ouviAumentar();  
    speech.ouviDiminuir();  
    speech.ouviAumentar();  
  
    for(int i=0; i< speech.getListaComandos().size(); i++)  
        System.out.println(speech.getListaComandos().get(i));  
}
```

Comando

Lista de comandos
executados

Aumentar e diminuir volume



```
public class VolumeDownCommand implements Command {  
  
    private Radio radio;  
    public VolumeDownCommand(Radio radio) {  
        this.radio = radio;  
    }  
    @Override  
    public void execute() {  
        this.radio.diminuirVolume();  
    }  
}
```

```
public class VolumeUpCommand implements Command {  
  
    private Radio radio;  
    public VolumeUpCommand(Radio radio) {  
        this.radio = radio;  
    }  
    @Override  
    public void execute() {  
        this.radio.aumetarVolume();  
    }  
}
```

```
public static void main(String[] args) {  
  
    ArrayList listaEstacoes = new ArrayList();  
    listaEstacoes.add("100.1");  
    listaEstacoes.add("92.5");  
    listaEstacoes.add("93.5");  
    Radio radio = new Radio(listaEstacoes);  
    SpeechRecogniser speech = new SpeechRecogniser();  
    speech.setCommands(new VolumeUpCommand(radio),  
        new VolumeDownCommand(radio));  
    speech.ouviAumentar();  
    speech.ouviDiminuir();  
    speech.ouviAumentar();  
  
}
```

Novo comando

PADRÕES COMPORTAMENTAIS

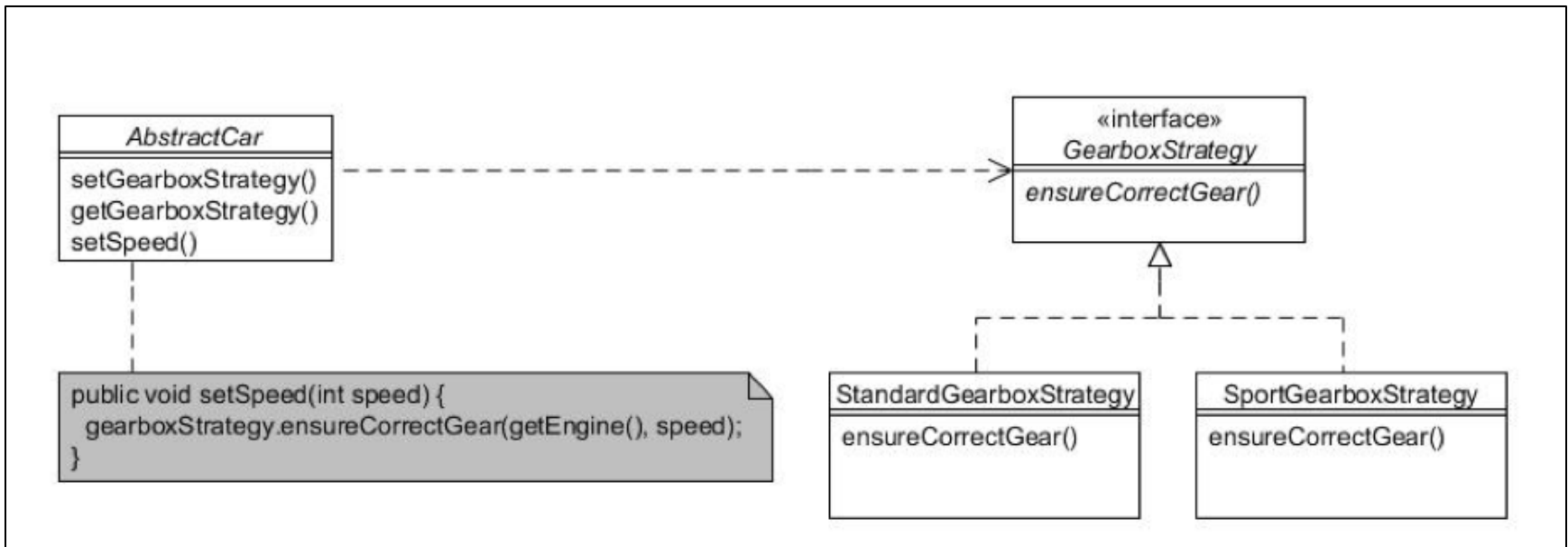
Estratégia

PADRÕES DE PROJETO

- Estratégia:
 - **Propósito:** permite que o cliente mude o algoritmo de um objeto que utiliza uma função
 - **Problema:**
 - Motores SA deseja implementar um novo tipo de câmbio automático em seus carros;
 - O novo câmbio permite trocar do modo padrão para o esporte;
 - A troca do “modo” depende da velocidade do carro, tamanho do motor e se o carro possui turbo;
 - No futuro eles desejam desenvolver outros “modos” como, por exemplo, off-road driving;

PADRÕES DE PROJETO

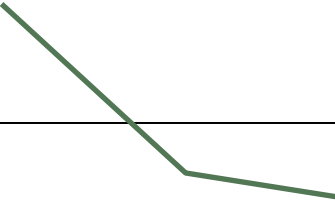
- Estratégia:



PADRÕES DE PROJETO

- Estratégia:

```
public interface GearboxStrategy {  
    public void ensureCorrectGear(Engine engine, int speed);  
}
```



Interface de
padronização

```
public class StandardGearboxStrategy implements GearboxStrategy {
    public void ensureCorrectGear(Engine engine, int speed) {
        int engineSize = engine.getSize();
        boolean turbo = engine.isTurbo();

        // Some complicated code to determine correct gear
        // setting based on engineSize, turbo & speed, etc.
        // ... omitted ...

        System.out.println("Working out correct gear at " + speed + "mph for a
STANDARD gearbox");
    }
}
```



Cambio
Automático Padrão

```
public class SportGearboxStrategy implements GearboxStrategy {
    public void ensureCorrectGear(Engine engine, int speed) {
        int engineSize = engine.getSize();
        boolean turbo = engine.isTurbo();

        // Some complicated code to determine correct gear
        // setting based on engineSize, turbo & speed, etc.
        // ... omitted ...

        System.out.println("Working out correct gear at " + speed + "mph for a SPORT
gearbox");
    }
}
```



Cambio Automático
Esporte

PADRÕES DE PROJETO

```
public abstract class AbstractCar extends AbstractVehicle {
    private GearboxStrategy gearboxStrategy;

    public AbstractCar(Engine engine) {
        this(engine, Vehicle.Colour.UNPAINTED);
    }

    public AbstractCar(Engine engine) {
        super(engine);

        // Starts in standard gearbox mode (more economical)
        gearboxStrategy = new StandardGearboxStrategy();
    }

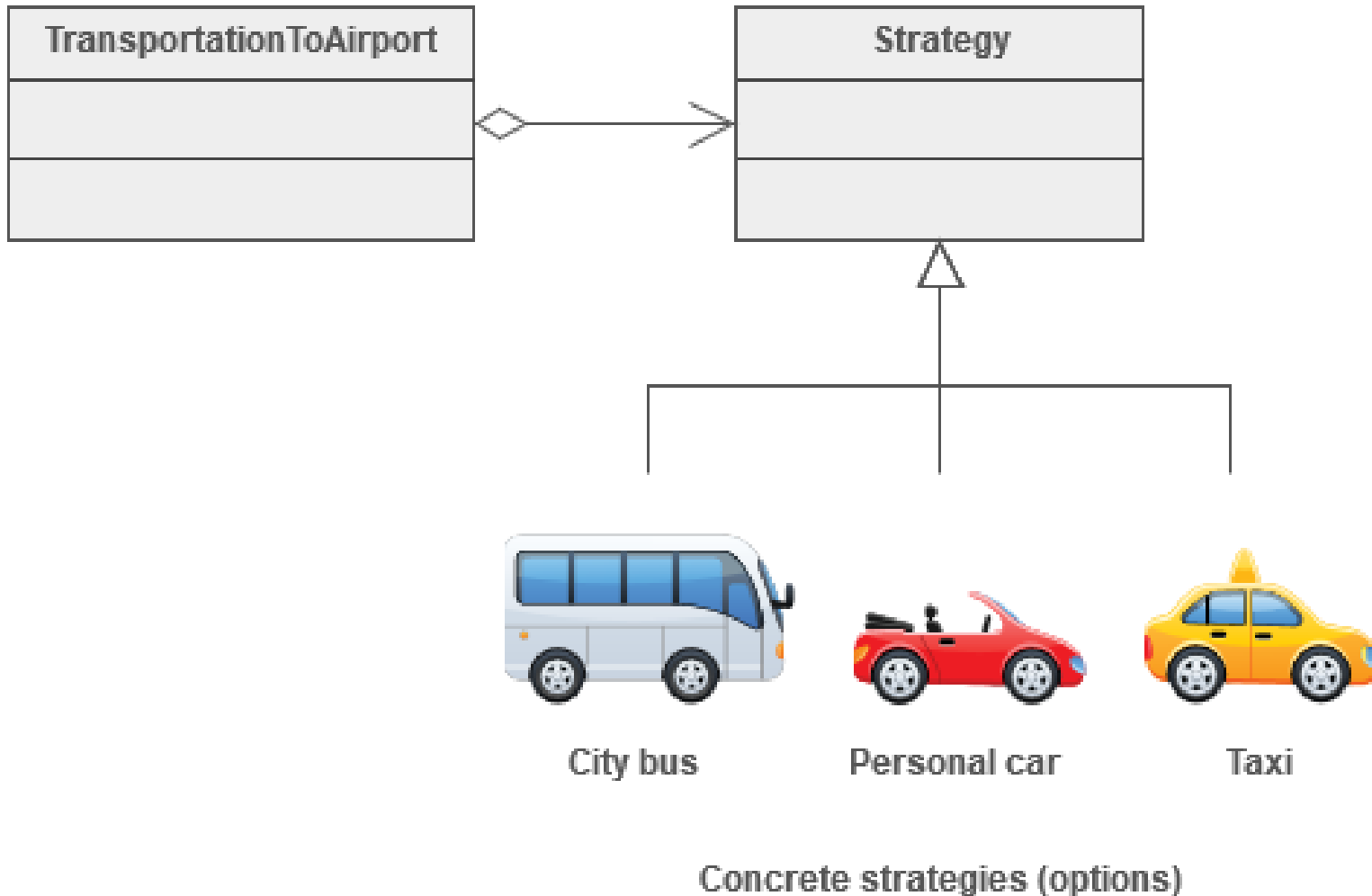
    // Allow the gearbox strategy to be changed...
    public void setGearboxStrategy(GearboxStrategy gs) {
        gearboxStrategy = gs;
    }

    public GearboxStrategy getGearboxStrategy() {
        return getGearboxStrategy();
    }

    public void setSpeed(int speed) {
        // Delegate to strategy in effect...
        gearboxStrategy.ensureCorrectGear(getEngine(), speed);
    }
}
```

Permite mudar a
minha estratégia

Transporte para o Aeroporto



```
public abstract class Strategy {  
  
    protected int distance =10;  
    public Strategy(int distance){  
        this.distance = distance;  
    }  
  
    public abstract double cost();  
}
```

```
public class Car extends Strategy {  
  
    public Car(int distance) {  
        super(distance);  
    }  
  
    @Override  
    public double cost() {  
        return 1000* distance;  
    }  
}
```



```
public class Cab extends Strategy {  
  
    public Cab(int distance) {  
        super(distance);  
    }  
  
    @Override  
    public double cost() {  
        return 1020* distance;  
    }  
  
}
```



```
public class Bus extends Strategy {  
  
    public Bus(int distance) {  
        super(distance);  
    }  
  
    @Override  
    public double cost() {  
        return 10* distance;  
    }  
  
}
```




```
public class TransportToAirPort {  
  
    Strategy transport;  
  
    public TransportToAirPort(Strategy transport) {  
        this.transport = transport;  
    }  
  
    public double calculateCost() {  
        return this.transport.cost();  
    }  
  
}
```

Thread

Java is a *multi threaded programming language* which means we can develop multi threaded program using Java.

A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

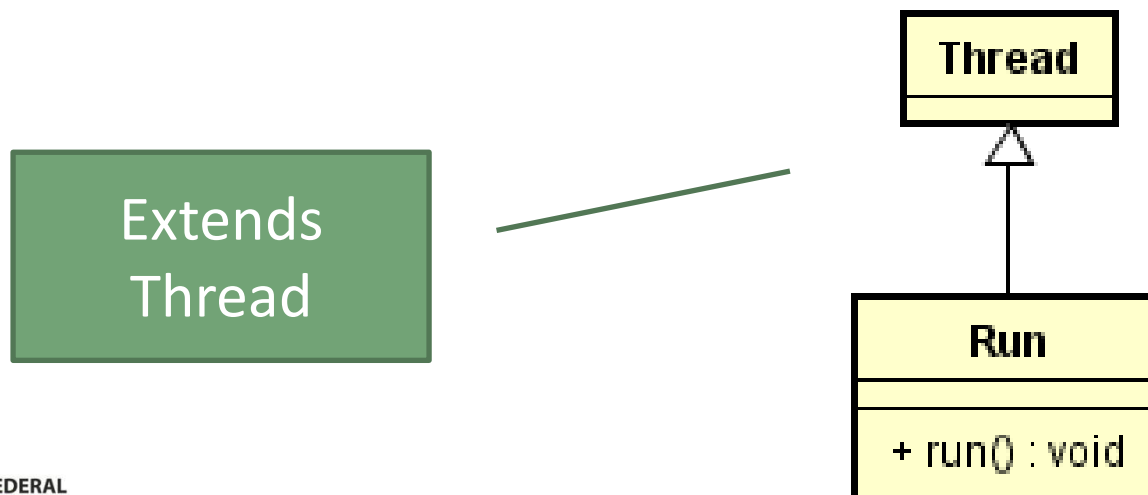
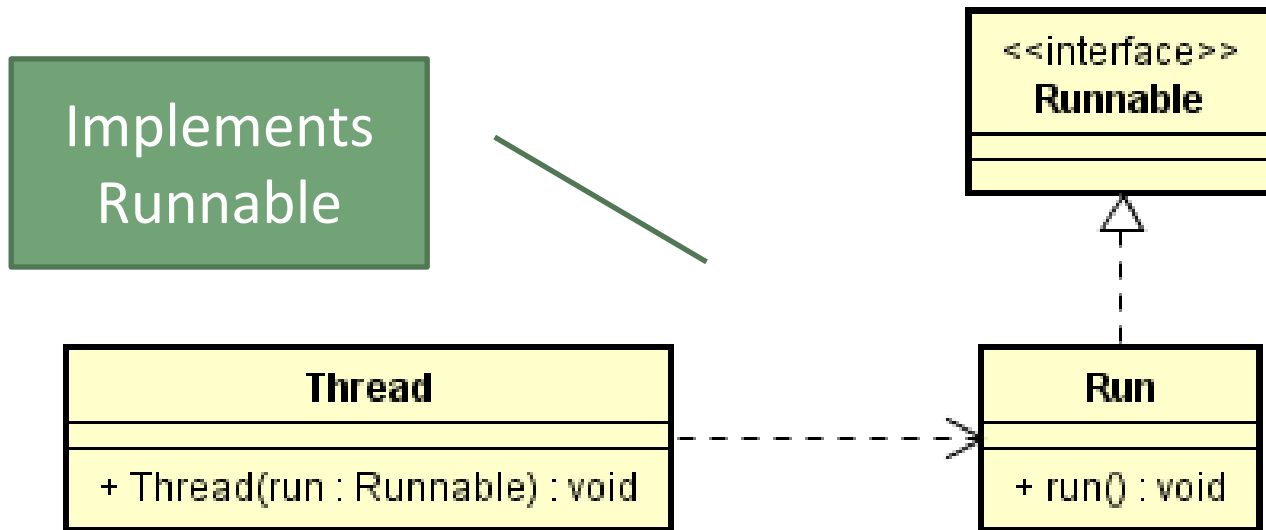
Thread Priority

Toda thread java tem uma prioridade que ajuda o sistema operacional a determinar sua ordem de execução;

MIN_PRIORITY (1) e MAX_PRIORITY (10).

Por padrão, uma thread recebe a prioridade NORM_PRIORITY (5).

Duas maneiras de utilizar Threads



https://github.com/felipefo/poo2/tree/master/Padroes_de_Projeto/Comportamental/Estrategia/Strategy_Threads

Implements Runnable

```
public class DisplayUUIDMainRunnable
```

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

```
{  
    Runnable runUUID1 = new DisplayUUIDRunnable("Thread1");  
    Thread thread1 = new Thread(runUUID1);  
    thread1.start();  
}
```

Setando a minha
estratégia

```
public class DisplayUUIDRunnable implements Runnable
```

```
{  
    String threadName;  
    public DisplayUUIDRunnable(String threadName)  
    {  
        this.threadName = threadName;  
    }  
    public void run()  
    {  
        int i =0;  
        while(i<10)//cada execucao vai imprimir 10 vezes o uuid  
        {  
            try {
```

Implements
Runnable

Seu código

```
        int i =0;  
        while(i<10)//cada execucao vai imprimir 10 vezes o uuid  
        {  
            try {
```

Extends Thread

```
public class DisplayUUIDMainExtend {  
    public static void main(String args[]) {  
        Thread thread1 = new DisplayUUIDExtend("Thread1");  
        thread1.start();  
        Thread thread2 = new DisplayUUIDExtend("Thread2");  
        thread2.start();  
    }  
}
```

Extends a
Thread

```
public class DisplayUUIDExtend extends Thread{  
    private String threadName;  
    public DisplayUUIDExtend(String threadName)  
    {  
        this.threadName = threadName;  
    }  
    public void run()  
    {  
        int i =0;  
        while(i<10)  
        {  
            try {
```

Seu código

Conceitos de Multithread

Synchronized: permite somente a execução de uma thread por vez onde estiver compreendido o trecho do código;

```
public static synchronized meuMetodoSinchronized()  
{  
    //somente uma thread por vez nesse trecho de código  
}
```


Pensando sobre Threads...



Qual a melhor
forma de utilizar
Thread??
Extensão ou
Composição?

Exemplos de utilização Padrão Strategy

- Save files in different formats.
- Compress files using different algorithms
- Capture video data using different compression schemes
- Use different line-breaking strategies to display text data.
- Plot the same data in different formats: line graph, bar chart or pie chart.

PADRÕES COMPORTAMENTAIS

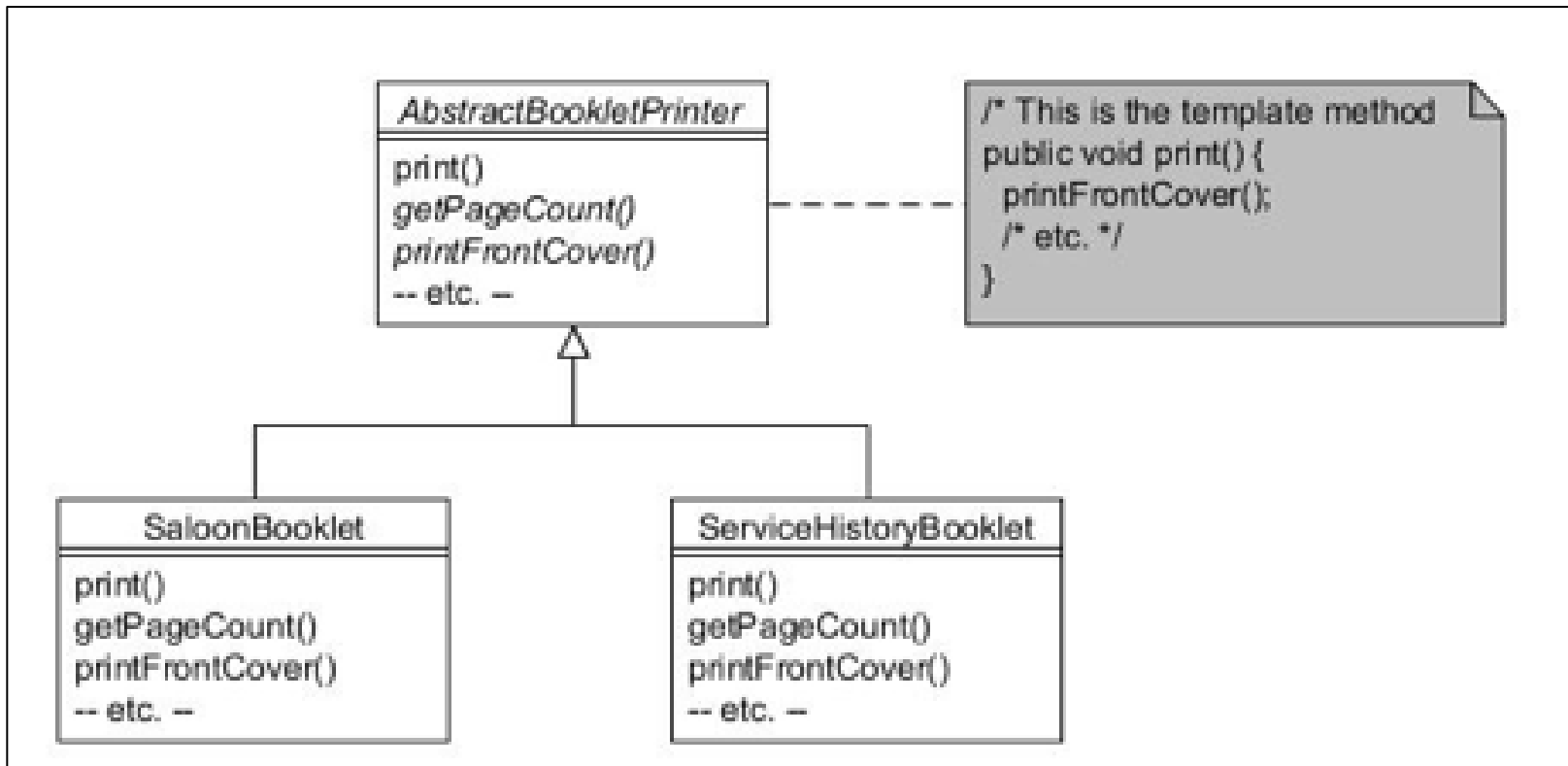
Template Method

PADRÕES DE PROJETO

- Template Method:
 - **Propósito:** define o “esqueleto” de um algoritmo em um método, utilizando as subclasses como etapas para o algoritmo;
 - Define alguns métodos e deixa alguns para a classe concreta implementar;
 - **Problema:**
 - Cada veículo fabricado pela Motores SA precisa dos seguintes documentos que precisam ser produzidos e entregues aos compradores:
 - Manual do proprietário;
 - Catálogo de serviços

PADRÕES DE PROJETO

Template Method:



PADRÕES DE PROJETO

```
public abstract class AbstractBookletPrinter {
    protected abstract int getPageCount();
    protected abstract void printFrontCover();
    protected abstract void printTableOfContents();
    protected abstract void printPage(int pageNumber);
    protected abstract void printIndex();
    protected abstract void printBackCover();

    // This is the 'template method'
    public final void print() {
        printFrontCover();
        printTableOfContents();
        for (int i = 1; i <= getPageCount(); i++) {
            printPage(i);
        }
        printIndex();
        printBackCover();
    }
}
```

Métodos
abstratos

Template
método



```
public class SaloonBooklet extends AbstractBookletPrinter {
    protected int getPageCount() {
        return 100;
    }

    protected void printFrontCover() {
        System.out.println("Printing front cover for Saloon car booklet");
    }

    protected void printTableOfContents() {
        System.out.println("Printing table of contents for Saloon car booklet");
    }

    protected void printPage(int pageNumber) {
        System.out.println("Printing page " + pageNumber + " for Saloon car booklet");
    }

    protected void printIndex() {
        System.out.println("Printing index for Saloon car booklet");
    }

    protected void printBackCover() {
        System.out.println("Printing back cover for Saloon car booklet");
    }
}
```

Implementações!

```
public class ServiceHistoryBooklet extends AbstractBookletPrinter {  
    protected int getPageCount() {  
        return 12;  
    }  
  
    protected void printFrontCover() {  
        System.out.println("Printing front cover for service history booklet");  
    }  
  
    protected void printTableOfContents() {  
        System.out.println("Printing table of contents for service history booklet");  
    }  
  
    protected void printPage(int pageNumber) {  
        System.out.println("Printing page " + pageNumber + " for service history booklet");  
    }  
  
    protected void printIndex() {  
        System.out.println("Printing index for service history booklet");  
    }  
  
    protected void printBackCover() {  
        System.out.println("Printing back cover for service history booklet");  
    }  
}
```

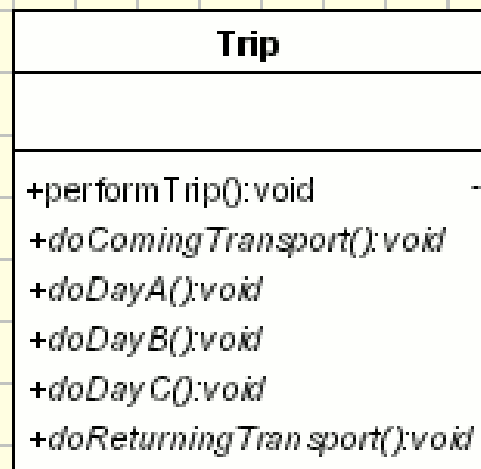
Implementações!

PADRÕES DE PROJETO

```
System.out.println("About to print a booklet for Saloon cars");  
AbstractBookletPrinter saloonBooklet = new SaloonBooklet();  
saloonBooklet.print();
```

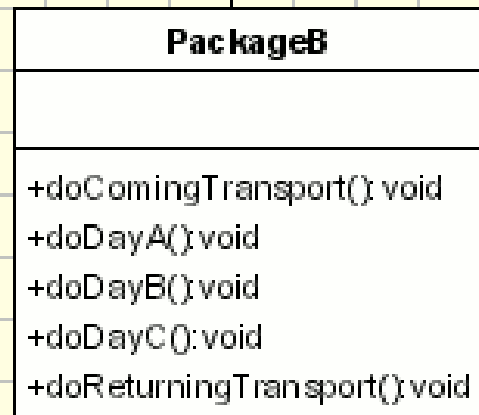
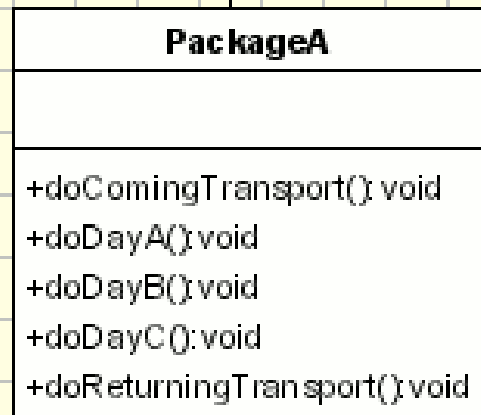
```
System.out.println("About to print a service history booklet");  
AbstractBookletPrinter serviceBooklet = new ServiceHistoryBooklet();  
serviceBooklet.print();
```

Template método



```
public final void performTrip(){
    doComingTransport();
    doDayA();
    doDayB();
    doDayC();
    doReturningTransport();
}
```

Os pacotes são sempre de 3 dias



```
public class Trip {  
    public final void performTrip(){  
        doComingTransport();  
        doDayA();  
        doDayB();  
        doDayC();  
        doReturningTransport  
    }  
    public abstract void doComingTransport();  
    public abstract void doDayA();  
    public abstract void doDayB();  
    public abstract void doDayC();  
    public abstract void doReturningTransport();
```

```
public class PackageA extends Trip {  
    public void doComingTransport() {  
        System.out.println("The turists are comming by air ...");  
    }  
    public void doDayA() {  
        System.out.println("The turists are visiting the aquarium ...");  
    }  
    public void doDayB() {  
        System.out.println("The turists are going to the beach ...");  
    }  
    public void doDayC() {  
        System.out.println("The turists are going to mountains ...");  
    }  
    public void doReturningTransport() {  
        System.out.println("The turists are going home by air ...");  
    }  
}
```

```
public class PackageB extends Trip {  
    public void doComingTransport() {  
        System.out.println("The turists are comming by train ...");  
    }  
    public void doDayA() {  
        System.out.println("The turists are visiting the mountain ...");  
    }  
    public void doDayB() {  
        System.out.println("The turists are going to the beach ...");  
    }  
    public void doDayC() {  
        System.out.println("The turists are going to zoo ...");  
    }  
    public void doReturningTransport() {  
        System.out.println("The turists are going home by train ...");  
    }  
}
```

```
PackageB packageB = new PackageB();  
packageB. performTrip();
```

```
PackageA packageA = new PackageA();  
packageA. performTrip();
```

PADRÕES COMPORTAMENTAIS

Memento

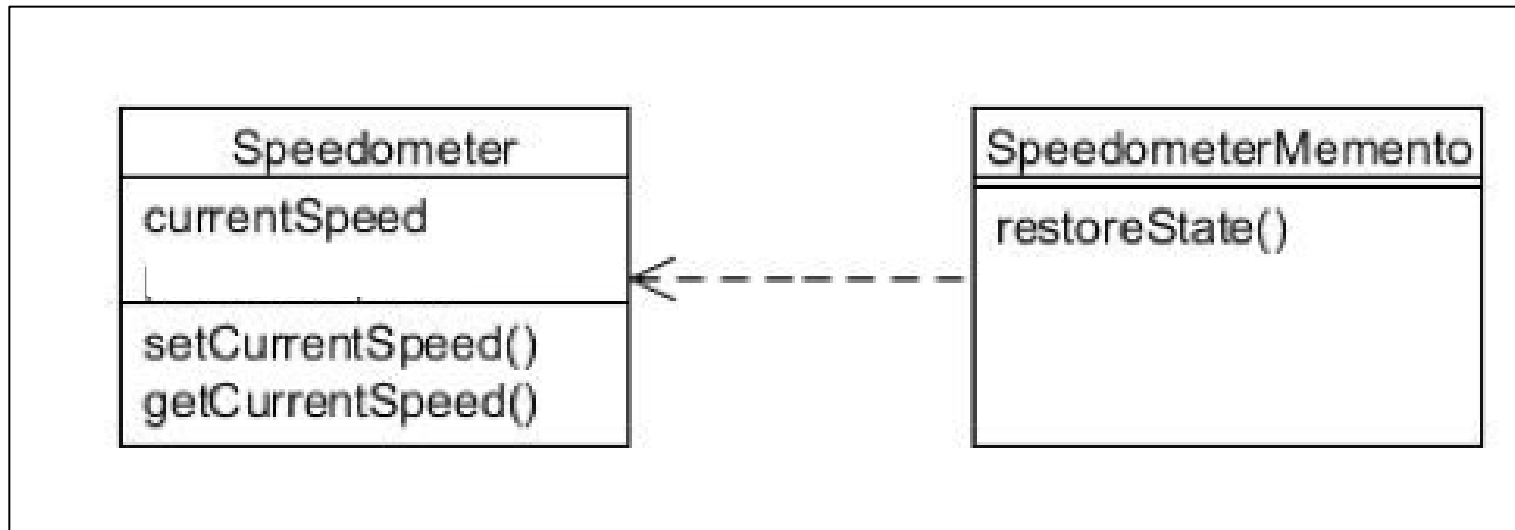
PADRÕES DE PROJETO

Memento:

- **Propósito:** Possibilitar a restauração do estado de um objeto;
- **Problema:**
 - Os veículos da Motores SA possuem um velocímetro no painel.
 - Esse velocímetro registram a velocidade atual e anterior;

PADRÕES DE PROJETO

Memento:



PADRÕES DE PROJETO

- Memento:
 - Formas de implementar:
 1. In memory
 2. Object Serialization;

PADRÕES DE PROJETO

```
public class Speedometer {  
    // Normal private visibility but has accessor method...  
    private int currentSpeed;  
  
    public Speedometer() {  
        currentSpeed = 0;  
    }  
  
    public void setCurrentSpeed(int speed) {  
        currentSpeed = speed;  
    }  
  
    public int getCurrentSpeed() {  
        return currentSpeed;  
    }  
}
```

PADRÕES DE PROJETO

```
package mementosubpackage;

public class SpeedometerMemento {
    private Speedometer speedometer;

    private int copyOfCurrentSpeed;

    public SpeedometerMemento(Speedometer speedometer) {
        this.speedometer = speedometer;
        copyOfCurrentSpeed = speedometer.getCurrentSpeed();
    }

    public void restoreState() {
        speedometer.setCurrentSpeed(copyOfCurrentSpeed);
    }
}
```

PADRÕES DE PROJETO

```
Speedometer speedo = new Speedometer();
```

```
speedo.setCurrentSpeed(100);
```

```
System.out.println("Current speed: " + speedo.getCurrentSpeed());
```

100

```
// Save the state of 'speedo'...
```

```
SpeedometerMemento memento = new SpeedometerMemento(speedo);
```

```
// Change the state of 'speedo'...
```

```
speedo.setCurrentSpeed(80);
```

```
System.out.println("After setting to 80...");
```

```
System.out.println("Current speed: " + speedo.getCurrentSpeed());
```

100

80

```
// Restore the state of 'speedo'...
```

```
System.out.println("Now restoring state...");
```

```
memento.restoreState();
```

```
System.out.println("Current speed: " + speedo.getCurrentSpeed());
```

```
;
```



PADRÕES DE PROJETO

Object Serialization

```
public class Speedometer implements Serializable {  
    private int currentSpeed;  
  
    public Speedometer() {  
        currentSpeed = 0;  
    }  
  
    public void setCurrentSpeed(int speed) {  
        currentSpeed = speed;  
    }  
  
    public int getCurrentSpeed() {  
        return currentSpeed;  
    }  
}
```



PADRÕES DE PROJETO

- Memento:Object Serialization

```
public class SpeedometerMemento {  
    public SpeedometerMemento(Speedometer speedometer) throws IOException {  
        // Serialize...  
        File speedometerFile = new File("speedometer.ser");  
        oos = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(speedometerFile)));  
        oos.writeObject(speedometer);  
        oos.close();  
    }  
  
    public Speedometer restoreState() throws IOException, ClassNotFoundException {  
        // Deserialize...  
        File speedometerFile = new File("speedometer.ser");  
        ois = new ObjectInputStream(new BufferedInputStream(new FileInputStream(speedometerFile)));  
        Speedometer speedo = (Speedometer) ois.readObject();  
        ois.close();  
        return speedo;  
    }  
}
```

PADRÕES COMPORTAMENTAIS

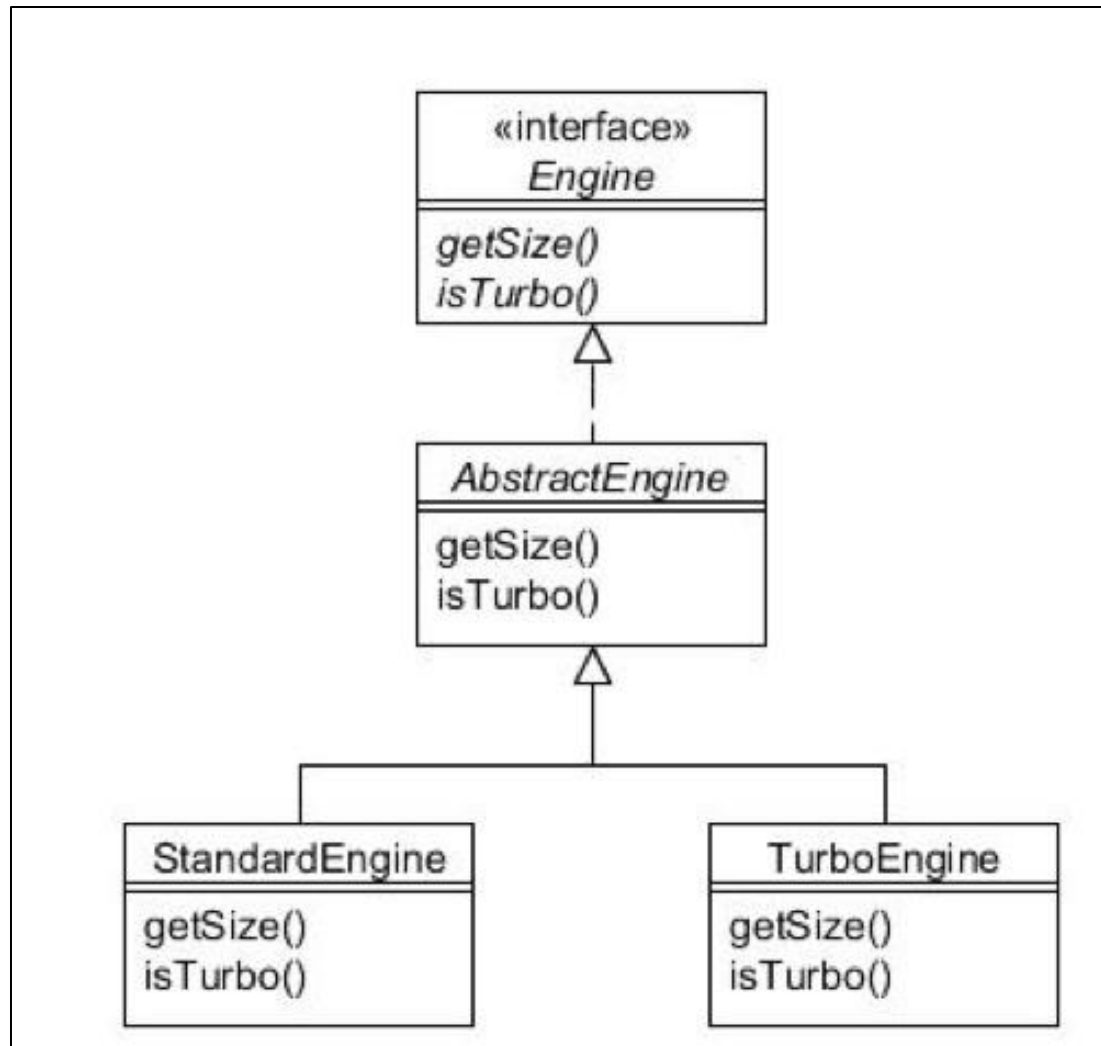
Visitador

PADRÕES DE PROJETO

- Visitador:
 - **Propósito:** **simula** a adição de um método a uma classe sem precisar mudar a classe;
 - **Problema:**
 - Em alguns casos não conhecemos como a classe será modificada.
 - Por exemplo no caso da Engine, podemos adicionar novas funcionalidades.
 - Como fazer isso sem fazer grandes modificações na classe?

PADRÕES DE PROJETO

- Visitador:

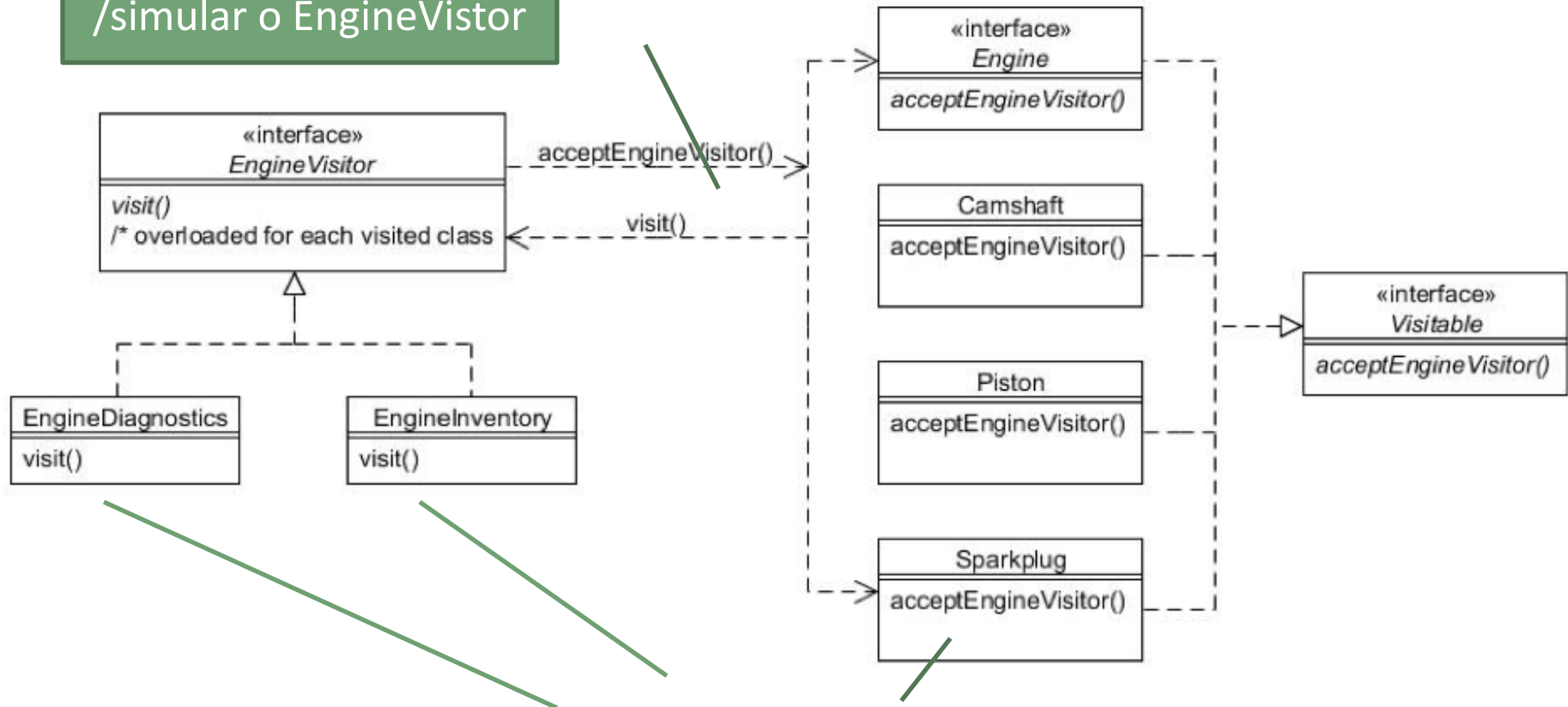


PADRÕES DE PROJETO

- Visitador:
 - Problema:
 - A AbstractEngine é composta por outras classes:
 - Pistão;
 - Vela;
 - Comando de válvula
 - Como fazer essas modificações sem uma grande mudança nas classes?

PADRÕES DE PROJETO

Fluxo para registrar
/simular o EngineVistor



Responsável por simular
a adição de um método

PADRÕES DE PROJETO

Interface que define como a visita a classe deve ser implementada

```
public interface EngineVisitor {  
    public void visit(Camshaft camshaft);  
    public void visit(Engine engine);  
    public void visit(Piston piston);  
    public void visit(SparkPlug sparkPlug);  
}
```

```
public interface Visitable {  
    public void acceptEngineVisitor(EngineVisitor visitor);  
}
```

Método que registrar/adiciona o EngineVistor que visitará a classe concreta

PADRÕES DE PROJETO

```
public class Camshaft implements Visitable {  
    public void acceptEngineVisitor(EngineVisitor visitor) {  
        visitor.visit(this);  
    }  
}
```

Método que adiciona a chamada ao EngineVistor que visitará a classe concreta

```
public class Piston implements Visitable {  
    public void acceptEngineVisitor(EngineVisitor visitor) {  
        visitor.visit(this);  
    }  
}
```

```
public class SparkPlug implements Visitable {  
    public void acceptEngineVisitor(EngineVisitor visitor) {  
        visitor.visit(this);  
    }  
}
```



PADRÕES DE PROJETO

```
public interface Engine extends Visitable {  
    public int getSize();  
    public boolean isTurbo();  
}
```

```

public abstract class AbstractEngine implements Engine {
    private int size;
    private boolean turbo;
    private Camshaft camshaft;
    private Piston piston;
    private SparkPlug[] sparkPlugs;
    public AbstractEngine(int size, boolean turbo) {
        this.size = size;
        this.turbo = turbo;

        // Create a camshaft, piston and 4 spark plugs...
        camshaft = new Camshaft();
        piston = new Piston();
        sparkPlugs = new SparkPlug[]{new SparkPlug(), new SparkPlug(), new SparkPlug(), new SparkPlug()};
    }
    public int getSize() {
        return size;
    }

    public boolean isTurbo() {
        return turbo;
    }
}

```


PADRÕES DE PROJETO(Cont. AbstractEngine)

```
public void acceptEngineVisitor(EngineVisitor visitor) {  
    // Visit each component first...  
    camshaft.acceptEngineVisitor(visitor);  
    piston.acceptEngineVisitor(visitor);  
    for (SparkPlug eachSparkPlug : sparkPlugs) {  
        eachSparkPlug.acceptEngineVisitor(visitor);  
    }  
  
    // Now visit the receiver...  
    visitor.visit(this);  
}  
  
public String toString() {  
    return getClass().getSimpleName() + " (" + size + ")";  
}
```

Na própria AbstractEngine

PADRÕES DE PROJETO

Classe que pode visitar
os visitáveis

```
public class EngineDiagnostics implements EngineVisitor {  
    public void visit(Camshaft camshaft) {  
        System.out.println("Diagnosing the camshaft");  
    }  
  
    public void visit(Engine engine) {  
        System.out.println("Diagnosing the unit engine");  
    }  
  
    public void visit(Piston piston) {  
        System.out.println("Diagnosing the pi");  
    }  
  
    public void visit(SparkPlug sparkPlug) {  
        System.out.println("Diagnosing a single spark plug");  
    }  
}
```

Método
simulado/adicionado



PADRÕES DE PROJETO

```
public class EngineInventory implements EngineVisitor {  
    private int camshaftCount;  
    private int pistonCount;  
    private int sparkPlugCount;
```

Classe que pode visitar
os visitáveis

```
    public EngineInventory() {  
        camshaftCount = 0;  
        pistonCount = 0;  
        sparkPlugCount = 0;  
    }
```

```
    public void visit(Camshaft p) {  
        camshaftCount++;  
    }
```

Método
simulado/adicionado

```
    public void visit(Engine e) {  
        System.out.println("The engine has: " + camshaftCount + "  
camshaft(s), " + pistonCount + " piston(s), and " + sparkPlugCount + "  
spark plug(s)");  
    }
```



```
// Create an engine...
Engine engine = new StandardEngine(1300);

// Run diagnostics on the engine...
engine.acceptEngineVisitor(new EngineDiagnostics());
```

Chamando o
novo/outro método

The above will result in the following output:

```
Diagnosing the camshaft
Diagnosing the piston
Diagnosing a single spark plug
Diagnosing a single spark plug
Diagnosing a single spark plug
Diagnosing a single spark plug
Diagnosing the unit engine
```

And to obtain the inventory (using the same `Engine` instance):

```
// Run inventory on the engine...
engine.acceptEngineVisitor(new EngineInventory());
```

Chamando o
novo/outro método

PADRÕES COMPORTAMENTAIS

Iterator

PADRÕES DE PROJETO

- **Iterator:**
 - **Propósito:** prover um meio de acesso sequencial aos elementos de uma estrutura, sem informar a representação da estrutura;
 - **Problema:**
 - A Motores SA quer produzir uma documentação que lista os tipos de veículos;
 - A empresa alocou dois programadores para essa tarefa;
 - O primeiro é responsável por listar os carros e o segundo por listar as vans;

PADRÕES DE PROJETO

- Iterator:

```
public interface Iterator<E> {  
    public boolean hasNext();  
    public E next();  
    public void remove();  
}
```

- **hasNext():** retorna true se ainda existe itens;
- **next():** retorna o próximo item;
- **remove:** remove o item retornado;

PADRÕES DE PROJETO

- Iterator:

```
public class CarRange {  
    private List<Vehicle> cars;  
  
    public CarRange() {  
        cars = new ArrayList<Vehicle>();  
  
        cars.add(new Saloon(new StandardEngine(1300)));  
        cars.add(new Saloon(new StandardEngine(1600)));  
        cars.add(new Coupe(new StandardEngine(2000)));  
        cars.add(new Sport(new TurboEngine(2500)));  
    }  
  
    public List<Vehicle> getRange() {  
        return cars;  
    }  
}
```



PADRÕES DE PROJETO

- Iterator:

```
public class VanRange {  
    private Vehicle[] vans;  
  
    public VanRange() {  
        vans = new Vehicle[3];  
  
        vans[0] = new BoxVan(new StandardEngine(1600));  
        vans[1] = new BoxVan(new StandardEngine(2000));  
        vans[2] = new Pickup(new TurboEngine(2200));  
    }  
  
    public Vehicle[] getRange() {  
        return vans;  
    }  
}
```

VanIterator

```
public class VanIterator implements Iterator{

    Veiculo[] veiculos;
    int position =0;
    public VanIterator(Veiculo[] veiculos){
        this.veiculos = veiculos;
    }
    @Override
    public boolean hasNext() {
        if(position >= veiculos.length || veiculos[position] == null ){
            return false;
        }
        else return true;
    }

    @Override
    public Object next() {
        Veiculo veiculo = veiculos[position];
        position++;
        return veiculo;
    }

}
```



Iterator

```
        Iterator listaCarros = carros.iterator();  
        iteraLista(listaCarros);  
  
        VanIterator vanIterator = new VanIterator(vans);  
        iteraLista(vanIterator);  
  
    }  
  
    public static void iteraLista(Iterator itens){  
  
        while(itens.hasNext()){  
            System.out.println(itens.next());  
        }  
  
    }
```



PADRÕES COMPORTAMENTAIS

Interpreter

PADRÕES DE PROJETO

- Interpretar:
 - **Propósito:** define a representação de uma gramática;
 - **Problema:**

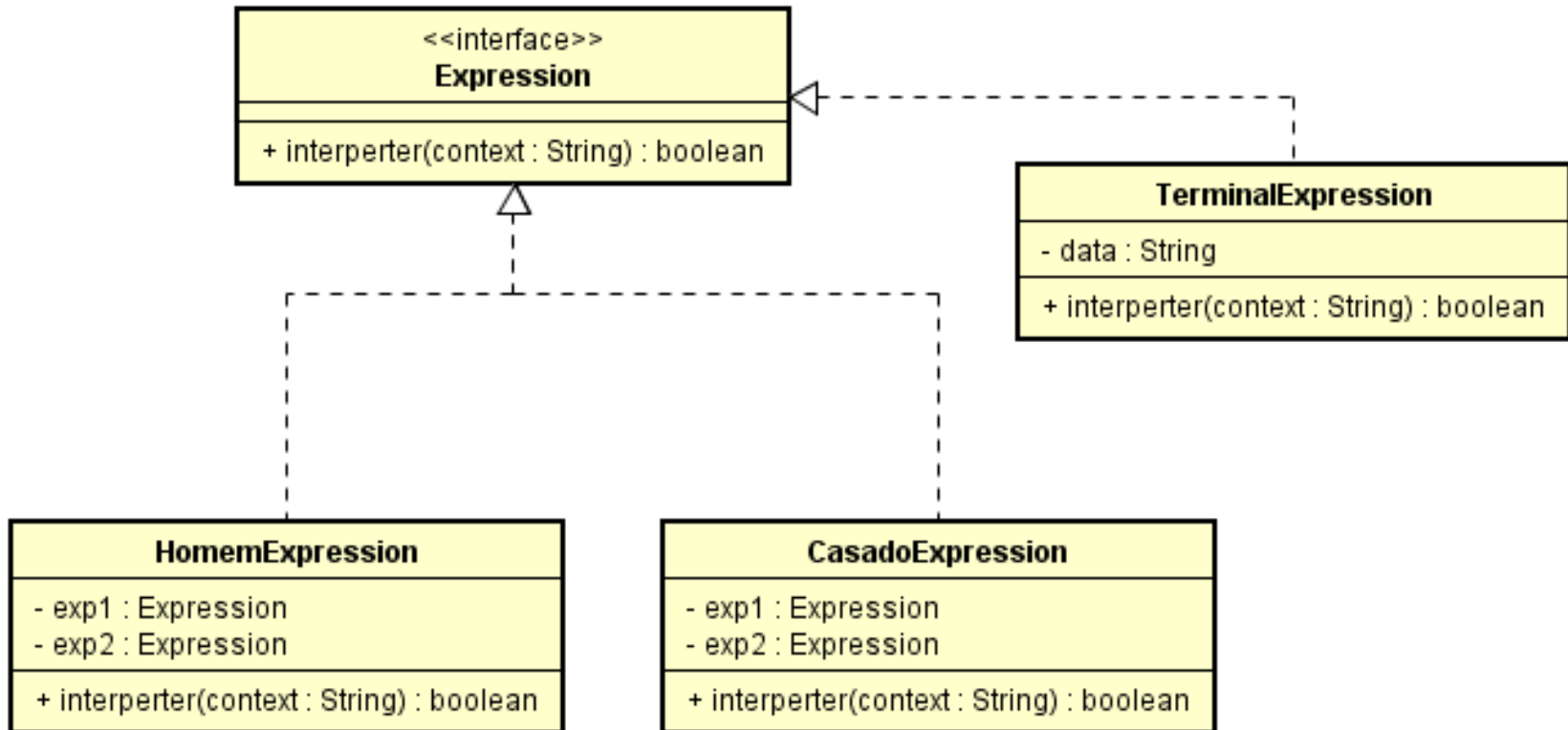
Dado a seguinte expressão:

"João Roberto homem Julia Renata casada";

Julia é casada? true

João é homem? true

PADRÕES DE PROJETO



PADRÕES DE PROJETO

```
public interface Expression {  
    public boolean interpret(String context);  
}
```

PADRÕES DE PROJETO

```
public class HomemExpression implements Expression {  
  
    private Expression expr1 = null;  
    private Expression expr2 = null;  
  
    public HomemExpression(Expression expr1, Expression expr2) {  
        this.expr1 = expr1;  
        this.expr2 = expr2;  
    }  
  
    @Override  
    public boolean interpret(String context) {  
        return (expr1.interpret(context) || expr2.interpret(context))  
            && context.contains("homem");  
    }  
}
```


PADRÕES DE PROJETO

```
public class CasadaExpression implements Expression {  
  
    private Expression expr1 = null;  
    private Expression expr2 = null;  
  
    public CasadaExpression(Expression expr1, Expression expr2) {  
        this.expr1 = expr1;  
        this.expr2 = expr2;  
    }  
  
    @Override  
    public boolean interpret(String context) {  
        return (expr1.interpret(context) || expr2.interpret(context))  
            && context.contains("casada");  
    }  
}
```

PADRÕES DE PROJETO

```
public class TerminalExpression implements Expression {  
  
    private String data;  
  
    public TerminalExpression(String data){  
        this.data = data;  
    }  
  
    @Override  
    public boolean interpret(String context) {  
  
        if(context.contains(data)){  
            return true;  
        }  
        return false;  
    }  
}
```



PADRÕES DE PROJETO

```
public class InterpreterPatternDemo {

    public static void main(String[] args) {
        Stack<Expression> expressions = new Stack();
        String entrada = "João Roberto homem Julia Renata casada";
        StringTokenizer st = new StringTokenizer(entrada);
        while (st.hasMoreTokens()) {
            String currChar = st.nextToken();
            if(currChar.equalsIgnoreCase("homem")){
                expressions.push(new HomemExpression(expressions.pop(), expressions.pop() ));
            }else if(currChar.equalsIgnoreCase("casada")){
                expressions.push(new CasadaExpression(expressions.pop(),expressions.pop() ));
            }else{
                expressions.push(new TerminalExpression(currChar));
            }
        }
        System.out.println("Julia é casada? " + expressions.pop().interpret("casada Julia"));
        System.out.println("João é homem? " + expressions.pop().interpret("homem João"));
    }
}
```



PADRÕES DE PROJETO

- Iterator:
 - Como eu posso acessar os objetos das classes sem saber a representação interna?

PADRÕES DE PROJETO

- Mediador:
 - O padrão Mediador ajuda a resolver esse problema através de uma classe separada (o mediador) que sabe:
 - O comportamento dos objetos;
 - Como combinar o comportamento dos objetos;

PADRÕES DE PROJETO

```
public class Ignition {
    private EngineManagementSystem mediator;
    private boolean on;

    // Constructor accepts mediator as an argument
    public Ignition(EngineManagementSystem mediator) {
        this.mediator = mediator;
        on = false;

        // Register back with the mediator...
        mediator.registerIgnition(this);
    }

    public void start() {
        on = true;
        mediator.ignitionTurnedOn();
        System.out.println("Ignition turned on");
    }

    public void stop() {
        on = false;
        mediator.ignitionTurnedOff();
        System.out.println("Ignition turned off");
    }

    public boolean isOn() {
        return on;
    }
}
```

```

public class EngineManagementSystem {
    private Ignition ignition;
    private Gearbox gearbox;
    private Accelerator accelerator;
    private Brake brake;

    private int currentSpeed;

    public EngineManagementSystem() {
        currentSpeed = 0;
    }

    // Methods that enable registration with this mediator...

    public void registerIgnition(Ignition ignition) {
        this.ignition = ignition;
    }

    public void registerGearbox(Gearbox gearbox) {
        this.gearbox = gearbox;
    }

    public void registerAccelerator(Accelerator accelerator) {
        this.accelerator = accelerator;
    }

    public void registerBrake(Brake brake) {
        this.brake = brake;
    }

    // Methods that handle object interactions...

```

```

public void ignitionTurnedOn() {
    gearbox.enable();
    accelerator.enable();
    brake.enable();
}

public void ignitionTurnedOff() {
    gearbox.disable();
    accelerator.disable();
    brake.disable();
}

public void gearboxEnabled() {
    System.out.println("EMS now controlling the gearbox");
}

public void gearboxDisabled() {
    System.out.println("EMS no longer controlling the gearbox");
}

public void gearChanged() {
    System.out.println("EMS disengaging revs while gear changing");
}

public void acceleratorEnabled() {
    System.out.println("EMS now controlling the accelerator");
}

public void acceleratorDisabled() {
    System.out.println("EMS no longer controlling the accelerator");
}

public void acceleratorPressed() {
    brake.disable();
    while (currentSpeed < accelerator.getSpeed()) {
        currentSpeed ++;
    }
}

```



```

        System.out.println("Speed currently " + currentSpeed)

        // Set gear according to speed...
        if (currentSpeed <= 10) {
            gearbox.setGear(Gearbox.Gear.FIRST);

        } else if (currentSpeed <= 20) {
            gearbox.setGear(Gearbox.Gear.SECOND);

        } else if (currentSpeed <= 30) {
            gearbox.setGear(Gearbox.Gear.THIRD);

        } else if (currentSpeed <= 50) {
            gearbox.setGear(Gearbox.Gear.FOURTH);

        } else {
            gearbox.setGear(Gearbox.Gear.FIFTH);
        }
    }
    brake.enable();
}

public void brakeEnabled() {
    System.out.println("EMS now controlling the brakes");
}

public void brakeDisabled() {
    System.out.println("EMS no longer controlling the brakes");
}

public void brakePressed() {
    accelerator.disable();
    currentSpeed = 0;
}

public void brakeReleased() {
        gearbox.setGear(Gearbox.Gear.FIRST);
        accelerator.enable();
    }
}

```

Referências

- **Bevis, Tony. Java: Design Pattern Essentials – Second Edition:**
 - Capítulos:
 - 14: Cadeia de Responsabilidade;
 - 15: Comando;
 - 16: Interpreter;
 - 17: Iterator;
 - 18: Mediator;
 - 19: Memento;
 - 20: Observer;
 - 21: Estratégia;
 - 23: Template Method;
 - 24: Visitor;

Referências

- https://sourcemaking.com/design_patterns/
- <http://www.oodesign.com/decorator-pattern-gui-example-java-sourcecode.html>
- <https://code.tutsplus.com/pt/tutorials/design-patterns-the-decorator-pattern--cms-22641>
- <https://github.com/chantastic/css-patterns>
- <http://archive.oreilly.com/pub/a/onjava/2003/02/05/decorator.html?page=1>

Licença para Uso e Distribuição

- Este material está disponível para uso não-comercial e pode ser derivado e/ou distribuído, desde que utilizando uma licença equivalente.

- Maiores informações: <http://creativecommons.org/licenses/by-nc-sa/2.5/deed.pt>

- Você pode copiar, distribuir, exibir e executar a obra, além de criar obras derivadas, sob as seguintes condições: (a) você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante; (b) você não pode utilizar esta obra com finalidades comerciais; (c) Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

- By Paulo Sergio Santos Junior