Prática com padrões de projeto

Chauã Queirolo

chaua.c.queirolo@gmail.com

1. Introdução

Este trabalho apresenta a documentação do problema proposto em sala de aula. A descrição do problema proposto é apresentado na Figura 1. Para solução do problema foram utilizados três padrões de projetos: *Strategy*, *Observer* e *Singleton*.

> Suppose a system that generates “safe routes” for pedestrians. However, the system must adapt itself in runtime as the person walk for streets. As the intention is to generate safe routes, the context can change in runtime, for example, if an attack or robbery happened in the generated route. When the context change, the system must be smart enough for proposing a new route.
>
> There are different ways (algorithms) for generating routes:
> 1 - generates a not-to-save but a shortest route
> 2 - generate a safe route in an acceptable distance
> 3 - generate a super-safe-route, but usually the distance is the largest
> Depending on the context, the generation of the route can change in runtime.

Figura 1: Enunciado do problema proposto.

O padrão *Strategy* foi utilizado para a escolha do método de busca no grafo de localidades: (1) menor caminho, (2) maior segurança, e (3) combinação de menor caminho e maior segurança. A utilização deste padrão aumenta a flexibilidade do sistema para a inclusão e exclusão das estratégias de busca.

O padrão *Observer* foi utilizado para atualizar o cálculo do percurso toda vez que um novo evento altere o grau de segurança de alguma localidade. Este padrão garante que a rota será sempre recalculada automaticamente sempre que o grafo de localidades for atualizado.

O padrão *Singleton* foi utilizado para garantir no sistema apenas uma instância do grafo de localidades. Como o grafo é imutável (novos vértices e arestas não são adicionados após a sua criação) e a construção do grafo pode ter alto custo computacional dependendo do número de vértices e arestas, a utilização deste padrão simplifica o seu acesso e evita criações desnecessárias.

Este trabalho está organizado como segue. A Seção 2 apresenta o diagrama de classes do sistema proposto. A Seção 3 apresenta o código-fonte do sistema proposto. O projeto completo pode ser encontrado no GitHub no endereço: https://github.com/chaua/mba-machine-learning.

2. Diagrama de classes

A Figura 2 apresenta o digrama de classes do sistema.

Figura 2: Diagrama de classes do sistema.

3. Código-fonte

3.1 Controladores

```
public class MainController {
    private final MainView mView = new MainView(this);
    private final EventoController mEventoController = new EventoController();
    private final PercursoController mPercursoController = new PercursoController();

    public void executar() {
        mView.exibirTelaPrincipal();
    }

    public void gerenciarPercurso() {
        mPercursoController.executar();
    }

    public void gerenciarEvento() {
        mEventoController.executar();
    }
}
```

Código 1: MainController

```
public class PercursoController implements MapaObserver {
    private final Mapa mMapa = MapaSingleton.getInstance();
    private final PercursoView mView = new PercursoView(this);

    private Coordenada mOrigem = new Coordenada();
    private Coordenada mDestino = new Coordenada();

    private PercursoStrategy mStrategy = new PercursoNormal();

    public PercursoController() {
        mMapa.registrarObservador(this);
    }

    public void executar() {
        mView.exibirTelaPrincipal();
    }

    public void setOrigem(final Coordenada origem) {
        mOrigem = origem;
    }

    public void setDestino(final Coordenada destino) {
        mDestino = destino;
    }

    public void setTipoPercurso(final TipoPercurso tipo) {
        switch (tipo) {
            case CURTO -> mStrategy = new PercursoCurto();
            case NORMAL -> mStrategy = new PercursoNormal();
            case SEGURO -> mStrategy = new PercursoSeguro();
        }
    }

    public List<Endereco> calcularPercurso() {
        return mStrategy.calcularPercurso(mOrigem, mDestino, mMapa);
    }

    @Override
    public void onMapaUpdate() {
        mView.exibirTelaAtualizacaoPercurso();
    }
}
```

Código 2: PercursoController

```java
public class EventoController {
    private final Mapa mMapa = MapaSingleton.getInstance();
    private final EventoView mView = new EventoView(this);

    public void executar() {
        mView.exibirTelaPrincipal();
    }

    public void adicionarEvento(final Evento evento) {
        mMapa.atualizarEndereco(evento.getCoordenada(), evento.getSeguranca());
    }
}
```

Código 3: EventoController

3.2 Apresentação

```java
public class MainView {
    private final MainController mController;

    public MainView(final MainController controller) {
        mController = controller;
    }

    public void exibirTelaPrincipal() {
        Scanner teclado = new Scanner(System.in);
        boolean terminou = false;

        while (!terminou) {
            System.out.println();
            System.out.println("====================");
            System.out.println("= Painel Principal =");
            System.out.println("====================");
            System.out.println("1. Gerenciar percurso");
            System.out.println("2. Gerenciar eventos");
            System.out.println("3. Sair");
            System.out.print(">: ");

            int opcao = teclado.nextInt();
            switch (opcao) {
                case 1 -> mController.gerenciarPercurso();
                case 2 -> mController.gerenciarEvento();
                case 3 -> terminou = true;
            }
        }
    }
}
```

Código 4: MainView

```java
public class PercursoView {
    private final PercursoController mController;

    public PercursoView(PercursoController controller) {
        mController = controller;
    }

    public void exibirTelaPrincipal() {
        Scanner teclado = new Scanner(System.in);
        boolean terminou = false;

        while (!terminou) {
            System.out.println();
            System.out.println("======================");
            System.out.println("= Gerenciar Percurso =");
            System.out.println("======================");
            System.out.println("1. Cadastrar origem e destino");
            System.out.println("2. Visualizar percurso");
```

```
            System.out.println("3. Sair");
            System.out.print(">: ");

            int opcao = teclado.nextInt();
            switch (opcao) {
                case 1 -> cadastrarOrigemDestino();
                case 2 -> exibirTelaPercurso();
                case 3 -> terminou = true;
            }
        }
    }


    private void cadastrarOrigemDestino() {
        System.out.println();
        System.out.println("-------------------------");
        System.out.println("- 1) Cadastrar percurso -");
        System.out.println("-------------------------");

        Scanner teclado = new Scanner(System.in);
        System.out.println("- Digite as coordenadas de origem: ");
        Coordenada origem = new Coordenada(teclado.nextDouble(), teclado.nextDouble());
        mController.setOrigem(origem);

        System.out.println("- Digite as coordenadas de destino: ");
        Coordenada destino = new Coordenada(teclado.nextDouble(), teclado.nextDouble());
        mController.setDestino(destino);

        System.out.println("- Escolha o tipo de percurso: (0) Curto, (1) Normal, (2) Seguro ");
        switch (teclado.nextInt()) {
            case 0 -> mController.setTipoPercurso(TipoPercurso.CURTO);
            case 1 -> mController.setTipoPercurso(TipoPercurso.NORMAL);
            case 2 -> mController.setTipoPercurso(TipoPercurso.SEGURO);
            default -> {
                System.out.println("- Selecionando percurso normal...");
                mController.setTipoPercurso(TipoPercurso.NORMAL);
            }
        }
    }

    public void exibirTelaPercurso() {
        System.out.println();
        System.out.println("--------------------------");
        System.out.println("- 2) Visualizar Percurso -");
        System.out.println("--------------------------");

        List<Endereco> percurso = mController.calcularPercurso();
        for (Endereco endereco : percurso) {
            System.out.println(endereco.getLogradouro());
        }
    }

    public void exibirTelaAtualizacaoPercurso() {
        System.out.println();
        System.out.println("-----------------------------");
        System.out.println("- !! Percurso Atualizado !! -");
        System.out.println("-----------------------------");

        List<Endereco> percurso = mController.calcularPercurso();
        for (Endereco endereco : percurso) {
            System.out.println(endereco.getLogradouro());
        }
    }

}
```

Código 5: PercursoView

```java
public class EventoView {
    private final EventoController mController;

    public EventoView(final EventoController controller) {
        mController = controller;
    }

    public void exibirTelaPrincipal() {
        Scanner teclado = new Scanner(System.in);
        boolean terminou = false;

        while (!terminou) {
            System.out.println();
            System.out.println("======================");
            System.out.println("= Gerenciar Eventos =");
            System.out.println("======================");
            System.out.println("1. Cadastrar novo evento");
            System.out.println("3. Sair");
            System.out.print(">: ");

            int opcao = teclado.nextInt();
            switch (opcao) {
                case 1 -> cadastrarEvento();
                case 3 -> terminou = true;
            }
        }
    }

    private void cadastrarEvento() {
        System.out.println();
        System.out.println("------------------------");
        System.out.println("- 1) Cadastrar evento -");
        System.out.println("------------------------");

        Scanner teclado = new Scanner(System.in);
        System.out.println("- Digite o coordenadas do evento: ");
        Coordenada coordenada = new Coordenada(teclado.nextDouble(), teclado.nextDouble());

        System.out.println("- Escolha o tipo do evento: (0) Roubo, (1) Assalto, (2) Tiroteio");
        switch (teclado.nextInt()) {
            case 0 -> mController.adicionarEvento(new EventoRoubo(coordenada));
            case 1 -> mController.adicionarEvento(new EventoAssalto(coordenada));
            case 2 -> mController.adicionarEvento(new EventoTiroteio(coordenada));
            default -> {
                System.out.println("- Evento inválido!!!");
            }
        }
    }
}
```

Código 6: PercursoView

3.3 Entidades

```java
public class Coordenada {
    private double mLatitude;
    private double mLongitude;

    public Coordenada() {
        this(0, 0);
    }

    public Coordenada(double latitude, double longitude) {
        this.mLatitude = mLatitude;
        this.mLongitude = mLongitude;
    }

    public double getLatitude() {
        return mLatitude;
    }
```

```
    public double getLongitude() {
        return mLongitude;
    }
}
```

Código 7: Coordenada

```
public class Endereco {
    private final Coordenada mInicio;
    private final Coordenada mFim;

    private final String mLogradouro;
    private final int mNumeroMin;
    private final int mNumeroMax;

    private TipoSeguranca mTipoSeguranca;

    public Endereco(final Coordenada inicio, final Coordenada fim, final String logradouro,
                    final int numeroMax, final int numeroMin, final TipoSeguranca tipoSeguranca) {
        mInicio = inicio;
        mFim = fim;
        mLogradouro = logradouro;
        mNumeroMin = numeroMax;
        mNumeroMax = numeroMin;
        mTipoSeguranca = tipoSeguranca;
    }

    public String getLogradouro() {
        return mLogradouro;
    }

    public Coordenada getCoordInicio() {
        return mInicio;
    }

    public Coordenada getCoordFim() {
        return mFim;
    }

    public int getNumeroMin() {
        return mNumeroMin;
    }

    public int getNumeroMax() {
        return mNumeroMax;
    }

    public TipoSeguranca getSeguranca() {
        return mTipoSeguranca;
    }

    public void setSeguranca(final TipoSeguranca tipoSeguranca) {
        mTipoSeguranca = tipoSeguranca;
    }
}
```

Código 8: Endereço

```
public abstract class Evento {

    private final Coordenada mCoordenada;
    private final TipoSeguranca mSeguranca;

    public Evento(final Coordenada coordenada, final TipoSeguranca seguranca) {
        mCoordenada = coordenada;
        mSeguranca = seguranca;
    }

    public Coordenada getCoordenada() {
        return mCoordenada;
    }
```

```
    public TipoSeguranca getSeguranca() {
        return mSeguranca;
    }

}
public class EventoAssalto extends Evento {
    public EventoAssalto(final Coordenada coordenada) {
        super(coordenada, TipoSeguranca.PERIGO_ALTO);
    }
}
public class EventoRoubo extends Evento {
    public EventoRoubo(final Coordenada coordenada) {
        super(coordenada, TipoSeguranca.PERIGO_MEDIO);
    }
}

public class EventoTiroteio extends Evento {
    public EventoTiroteio(final Coordenada coordenada) {
        super(coordenada, TipoSeguranca.PERIGO_ALTO);
    }
}
```

Código 9: Evento

```
public class Mapa {

    private final Map<Coordenada, List<Endereco>> mGrafo;
    private final List<Endereco> mEnderecos;
    private final List<MapaObserver> mObservadores;

    public Mapa() {
        mGrafo = new HashMap<>();
        mEnderecos = new ArrayList<>();
        mObservadores = new ArrayList<>();
    }

    public void adicionarEndereco(final Endereco endereco) {
        addAresta(endereco.getCoordInicio(), endereco);
        addAresta(endereco.getCoordFim(), endereco);
        notificarObservadores();
    }

    private void addAresta(final Coordenada coordenada, final Endereco endereco) {
        if (mGrafo.containsKey(coordenada)) {
            mGrafo.get(coordenada).add(endereco);
        } else {
            List<Endereco> enderecos = new ArrayList<>();
            enderecos.add(endereco);
            mGrafo.put(coordenada, enderecos);
        }
    }

    private void notificarObservadores() {
        for (MapaObserver observador : mObservadores) {
            observador.onMapaUpdate();
        }
    }

    public void atualizarEndereco(final Coordenada coordenada, final TipoSeguranca seguranca) {
        // TODO: Buscar o endereco mais próximo com base nas coordenadas
        // TODO: Atualizar o status de seguranca do endereco
        notificarObservadores();
    }

    public void registrarObservador(final MapaObserver observador) {
        mObservadores.add(observador);
    }

}
```

Código 9: Mapa

```java
public enum TipoPercurso {
    CURTO,
    NORMAL,
    SEGURO
}
```

Código 10: TipoPercurso

```java
public enum TipoSeguranca {
    SEGURANCA_ALTA,
    SEGURANCA_MEDIA,
    SEGURANCA_BAIXA,
    PERIGO_MEDIO,
    PERIGO_ALTO
}
```

Código 11: TipoSeguranca

3.5 Observer

```java
public interface MapaObserver {
    void onMapaUpdate();
}
```

Código 12: MapaObserver

3.6 Singleton

```java
public class MapaSingleton {
    private static Mapa sMapa;

    private MapaSingleton() {
        // vazio
    }

    public static Mapa getInstance() {
        if (sMapa == null) {
            MapaDAO dao = new MapaDAO();
            sMapa = dao.carregarMapa();
        }
        return sMapa;
    }
}
```

Código 13: MapaSingleton

3.7 Strategy

```java
public interface PercursoStrategy {
    List<Endereco> calcularPercurso(final Coordenada origem, final Coordenada destino,
                                    final Mapa mapa);
}
```

Código 14: PercursoStrategy

```java
public class PercursoCurto implements PercursoStrategy {
    @Override
    public List<Endereco> calcularPercurso(final Coordenada origem, final Coordenada destino, final
Mapa mapa) {
        // TODO: Implementar busca no grafo considerando somente menor distância
        System.out.println(">> Calculando a rota mais curta, mas não tão segura");
        return new ArrayList<>();
    }
}
```

Código 15: PercursoCurto

```java
public class PercursoNormal implements PercursoStrategy {
    @Override
    public List<Endereco> calcularPercurso(final Coordenada origem, final Coordenada destino, final
Mapa mapa) {
        // TODO: Implementar busca no grafo considerando distância e segurança
        System.out.println(">> Calculando a rota segura e distância aceitável");
        return new ArrayList<>();
    }
}
```

Código 16: PercursoNormal

```java
public class PercursoSeguro implements PercursoStrategy {
    @Override
    public List<Endereco> calcularPercurso(final Coordenada origem, final Coordenada destino, final
Mapa mapa) {
        // TODO: Implementar busca no grafo considerando somente segurança
        System.out.println(">> Calculando a rota mais segura, mas com maior distância");
        return new ArrayList<>();
    }
}
```

Código 18: PercursoSeguro

3.7 DAO

```java
public class MapaDAO {

    public Mapa carregarMapa() {
        // TODO: conectar no banco de dados
        // TODO: carregar todos endereços e constuir o grafo
        System.out.println(">> Construindo mapa a partir do banco de dados...");
        return new Mapa();
    }
}
```

Código 19: MapaDAO