LUCAS FONSECA DOS SANTOS, LETICIA FERREIRA

CITY MANAGER V1.0:

SISTEMA PARA GERENCIA PÚBLICA ADMINISTRATIVA MUNICIPAL.

1^{a} edição revista e atualizada

LAVRAS – MG

2017

LUCAS FONSECA DOS SANTOS, LETICIA FERREIRA

CITY MANAGER V1.0:

SISTEMA PARA GERENCIA PÚBLICA ADMINISTRATIVA MUNICIPAL.

1^{a} edição revista e atualizada

Projeto pratico da disciplina de Paradigmas de Linguagens de Programação - GCC198 para aplicação direta dos conceitos angariados ao decorrer do semestre letivo da disciplina. O presente projeto apresentado consiste na aplicação dos conhecimentos relacionados ao paradigma de orientação à objeto.

Prof. Juliana Galvani

Orientadora

LAVRAS – MG

2017

RESUMO

O presente documento objetiva descrever detalhadamente a execução completa do projeto prático da disciplina de Paradigmas de Linguagens de Programação - GCC198 da Universidade Federal de Lavras, onde explana-se a respeito do processo de desenvolvimento do software City Manager, versão 1.0, um sistema de gerenciamento para administração publica municipal, cujo sua construção relaciona-se de modo direto e concreto com as aplicações dos conhecimentos angariados em sala de aula.

Palavras-chave: Software, Sistemas, Orientação à Objeto, Paradigmas de Linguagens de Programação.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Faz-se por objetivo do presente documento, a apresentação detalhada do processo de desenvolvimento do software City Manager, Versão 1.0 para o projeto prático da disciplina de Paradigmas de Linguagens de Programação, GCC198, Departamento de Ciência da Computação, Universidade Federal de Lavras, onde denota-se explanações e justificativas à cerca dos modelos de construção utilizados no presente projeto. Serão aprofundados os conceitos inerentes a estrutura e arquitetura do software bem como seu pleno funcionamento.

O tema escolhido para o desenvolvimento do sistema é cadastro e gerencia de bairros de uma cidade. Ampliou-se então tal temática, afim de abranger um conjunto maior de soluções, então, decidiu-se por uma implementação onde contabilizou-se também, cidades. Ao utilizar o software em qualquer maquina que possua uma Maquina Virtual Java (JVM) instalada, é possivel, ao usuario, por via de uma interface em linhas de texto de console, porém pensado de modo intuitivo, efetuar o cadastro de uma ou mais cidades. Para cada cidade cadastrada ao sistema e armazenada em um "banco de arquivos binários local", o sistema reserva e destina um módulo para a gerencia especifica, assim, o usuario final poderá acessar através de tal módulo, cada cidade e a partir de tal caminho, efetuar a gerencia pública a tal qual o referido software foi projetado.

O projeto foi arquitetado de forma altamente modularizada, tentando-se evitar um maior acoplamento de classes, bem como uma maior clareza de todos seus elementos, tornando-se de maior facilidade, manutenções prévias ou expansivas no próprio sistema. Abstraido por via dos pacotes definidos em Java, encontra-se um modelo diagramado por via de uma seção view, controller e data, o que amplia a facilidade por exemplo, na instalação de uma interface gráfica.

2 HISTÓRIA DO USUÁRIO

Subentende-se nos dias atuais, como um problema de grande escala, a centralização de dados referentes a gerencia de qualquer campo, sistema ou projeto. Trabalhando-se sobre tal ótica, desenvolveu-se o projeto do referido software City Manager, versão 1.0, visando uma centralização de dados para administração pública municipal ou estadual.

A partir desse sistema, é possivel cadastrar cidades e bairros, sendo assim, gerir a partir de tais dados, sub dados estatisticos como taxas demograficas, taxas de violencia, taxas de renda per capta, sobre cada bairro, bem como a cidade, sendo a média dos valores somados (quesito solicitado ao projeto). As tabelas abaixo, representa as caracteristicas das presentes informações tratadas no software:

Tabela 2.1 – Informações Geridas de uma Cidade

Informação	***Daemons* Adendo**
`name`	nome da cidade, sempre transformado pelo software para uppercase.
`state/province/district`	estado/provincia/distrito na qual situa-se a cidade.
`country`	pais na qual situa-se a cidade.
`mayor`	atual prefeito gestor da cidade. (IDEIA DE EXPANSÃO: Armazenar prefeitos antigos.)
`demographic rate`	taxa demografica da cidade. Somatório de todos os bairros.
`per capta income rate`	taxa de renda per capta da população. Média aritimética dos bairros.
`criminal index rate`	indice de violencia. Somatório de todos os bairros.
`neighborhoods`	bairros com informações.

Fonte: fonte da tabela

Tabela 2.2 – Informações Geridas de um Bairro

Informação	***Daemons* Adendo**
`name`	nome do bairro, sempre transformado pelo software para uppercase.
`demographic rate`	taxa demografica do bairro. Somatório de todos os bairros.
`per capta income rate`	taxa de renda per capta da população.
`criminal index rate`	indice de violencia.

Fonte: fonte da tabela

Todas as operações efetuadas no software geram registros de log. Tais registros são armazenados no em um banco de registros log, por via de arquivos de texto (.txt), identificados e nomeados a partir da data atual capturada do sistema vingente. O sistema de log é capaz de registrar as escolhas do usuario, exceções tratadas e não tratadas e novos dados registrados, removidos ou procurados. Ideias de expansão são muito aplicaveis nessa áerea. O sistema ainda não possui recurso de geração de log para os eventos de tratamento de erro, porém da forma como apresenta-se implementado, torna-se facil aplicar a modificação, abaixo apresenta-se um exemplo de arquivo log gerado pelo sistema:

Figura 2.1 – Exemplo de arquivo log

```
+=========================================+
+           CITY MANAGER LOG FILE         +
+=========================================+
+ Developed by Lucas Fonseca dos Santos.   +
+ Version: 1.0 2017                       +
+=========================================+
[27-07-2017]: LOG FILE CREATED! Welcome to my software!
[27-07-2017/ACTION]: SYSTEM STARTED! ;
[27-07-2017/ACTION]: All cities loaded of binary database files. ;
[27-07-2017/ACTION]: The user chosen load all cities stored. ;
[27-07-2017/ACTION]: The new cities entered by the user has been stored. ;
[RECORD > OBJECT HASH CODE]: 1. City: 21685669 ;
[27-07-2017/ACTION]: The user chosen create a new city operation. ;
[27-07-2017/ACTION]: All cities loaded of binary database files. ;
[27-07-2017/ACTION]: The user chosen load all cities stored. ;
[27-07-2017/ACTION]: All cities loaded of binary database files. ;
[27-07-2017/ACTION]: The user chosen load all cities stored. ;
[27-07-2017/ACTION]: All cities loaded of binary database files. ;
[27-07-2017/ACTION]: All cities stored at binary database files. ;
[27-07-2017/ACTION]: The user chosen manager a city operation. ;
[27-07-2017/ACTION]: All cities loaded of binary database files. ;
[27-07-2017/ACTION]: All cities stored at binary database files. ;
[27-07-2017/ACTION]: The user chosen edit a city data operation. ;
[27-07-2017/ACTION]: All cities loaded of binary database files. ;
[27-07-2017/ACTION]: The user chosen load all cities stored. ;
```

Fonte: diretório de registros de operações

Todos os dados registrados e manipulados no uso do sistema, são armazenados e organizados no banco de arquivos binários, localizado no diretório data. Os arquivos, de extensão (.bin) binária, armazenam objetos serializados pela interface Serializable do Java. Tais arquivos ainda podem ser manipulados via sistema diretamente, como apresenta-se por função disponivel ao usuário, por exemplo, a opção de remover todos os arquivos de dados guardados no sistema.

Cada cidade registrada no sistema, gera um arquivo binário de dados, já os bairros registrados, são armazenados nos arquivos de dados referentes as cidades.

A interface, como supracitado no presente capitulo, faz-se por via do console em linhas de texto, por comando unitários seguidos de argumentos simples. A interface, mesmo assim, demonstra-se bastante intuitiva, com toda comunicação em ingles e informações coloridas, facilitando um entendimento do usuário. À todo momento, o console passa por limpeza de tela, efetuado pelo objeto GUI, presente no software afim de se evitar confusões desnecessárias por parte do usuário. A seguir apresenta-se algumas opções presentes nas interfaces:

Figura 2.2 – Menu principal de navegação do sistema

```
+================================================+
+                                                +
+            CITY MANAGER SYSTEM v1.0            +
+                                                +
+================================================+
+                                                +
+ What do you want here?                         +
+ [1] Add a new city;                            +
+ [2] Remove a city;                             +
+ [3] Load all registereds citys;                +
+ [4] Edit a city;                               +
+ [5] Enter a city to manager it;                +
+ [6] Search a City;                             +
+ [7] Search a Neighborhood;                     +
+ [8] Generate statistics;                       +
+ [9] Delete Data Files;                         +
+                                                +
+ [-99] EXIT                                     +
+                                                +
+================================================+
[#]  ENTER YOUR OPTION:
```

Fonte: Software em execução

Figura 2.3 – Menu principal - impressão de dados

```
+------------------------------------------------+
 REGISTERED CITIES INFORMATIONS
+------------------------------------------------+


[!] CITY NAME: TESTE
[!] STATE/DISTRICT: 34
[!] COUNTRY: 23
[!] CURRENT MAYOR: 3
[!] DEMOGRAPHIC RATE: 4
[!] PER CAPTA INCOME RATE: 435.0
[!] CRIMINAL INDEX RATE: 345

 NEIGHBORHOODS:

[Neighborhood Name]: teste.
[Demographic Rate]: 4 peoples.
[Per Capta Income Rate]: 435.0
[Crime Rate]: 345 crime cases.


++++++++++++++++++++++++++++++++++++++++++++++++++

[#] Press any key to continue...
```

Fonte: Software em execução

Figura 2.4 – Menu principal - busca de bairro

```
+------------------------------------------------+
 SEARCH A NEIGHBORHOOD
+------------------------------------------------+

[#] WHICH NEIGHBORHOOD DO YOU WANT TO SEARCH? [TYPE NEIGHBORHOOD NAME]:
teste
[Neighborhood Name]: teste.
[Demographic Rate]: 4 peoples.
[Per Capta Income Rate]: 435.0
[Crime Rate]: 345 crime cases.

[#] Press any key to continue...
```

Fonte: Software em execução

Figura 2.5 – Menu do módulo de gerencia de cidade

```
+-------------------------------------------------+
TESTE
+-------------------------------------------------+
+                                                 +
+ What do you want with this city?                +
+ [1] Show city informations;                     +
+ [2] Add a new neighborhood;                     +
+ [3] Remove a neighborhood;                      +
+ [4] Edit a neighborhood;                        +
+ [5] Generate statistcs report file              +
+                                                 +
+ [-99] Back to main menu                         +
+                                                 +
+-------------------------------------------------+
[#]  ENTER YOUR OPTION:
```

Fonte: Software em execução

3 CARACTERISTICAS GERAIS DO SISTEMA

O referente sistema foi desenvolvido em Java, compartilhado em repositório público, sobre licença GPL-3 (General Public License), onde denota-se uma perspectiva, por parte de seus desenvolvedores, muito ampla com relação aos anseios de expansão posteriores. Em outras palavras, o software foi projeto à modo em que pudesse ser incrementado em versões posteriores, novas funcionalidades, interfaces e afins. Para tal, faz-se de suma importancia, a aplicação concreta da orientação à objeto, em todas as suas vertentes, com encapsulamento de informações, interfaces de comunicação, hierarquia de classes, herança de classes, polimorfismo aplicado, sobrecarca, sobrescrita e documentação propriamente dita (realizada nesse projeto por uso do Javadoc). O uso de todos e outros elmentos inerentes à orientação à objeto, aumentam a confiabilidade do sistema, legibilidade e manutentiabilidade, tornando atualizações futuras, algo a ser consideravel perante ao baixo nivel de dificuldade.

3.1 Polimorfismo

Um conceito extremamente importante dentro da orientação à objetos. Tal conceito, quando usado e aplicado de modo inteligente dentro do software, permite ao usuario uma enorme flexibilização do algoritimo, onde as especificidades podem ser tratadas de formas mais genéricas, permitindo alterações mais faceis do código. No presente projeto, todas as estrutura de dados listas, nativas do java, foram aplicadas de modo polimorfico, declarando-se variveis polimorficas (onde uma super classe instancia uma sub classe), permitindo uma flexibilização maior. Abaixo encontra-se um exemplo supracitado:

Figura 3.1 – Exemplo de aplicação do polimorfismo

```
List<City> cities = data.loadCities(); //Polimorfic variable List<> receives a Ar

List<City> newCities = new ArrayList<City>(); //List object instance a ArrayList<

public List<City> createCityInformations(List<City> oldCities);
```

Fonte: Class GUI

3.2 Herança

O conceito da herança dentro da orientação à objetos, permite uma maior coesão de classes dentro do sistema, com isso inclusive, aplicação de padrões de projeto. Porém, inversamente proporcional a tal, aumenta-se o acoplamento do sistema, por isso, usa-se somente quando necessário. No presente projeto, gozou-se do usufruto de uma interface, na qual duas classes concretas à implementava. Tal interface, rege os objetos que serão dados de gravações no sistema, nesse caso, cidade e bairro, que implementam a interface Record. Desse modo, como a tipificação por interface é uma opção valida, pode-se ter uma variavel do tipo Record apontando para um objeto que o implementa, garantindo assim a esperada flexibilidade do sistema para futuras implementações de atualização.

Figura 3.2 – Exemplo de aplicação de herança com class abstrata e interface

```
//Public concret class extends a record abstract class.
public class City extends Record

//Public abstract class record, that implements java Serializable interface.
public abstract class Record implements Serializable

//Public concret class extends a record abstract class.
public class Neighborhood extends Record
```

Fonte: app package

3.3 Sobrecarga

A sobrecarga de métodos é uma pratica altamente produtiva, ao momento em que faz-se necessário o uso de uma mesma função ou lógica de método porém para casos diferntes. É vedado a sobrecarga no algoritimo quando perante ao compilador, denota-se ambiguidade, seja por via de parametros posicionais o quantidade dos mesmos. Abaixo, podemos ver alguns trechos retirados do código do projeto, em que enquandram-se situações de sobrecarga de método:

Figura 3.3 – Exemplo de sobrecarga de método

```
public void showCitiesData(List<City> cities) {
...
}

public void showCitiesData(City c) {
...
}

\\Overwrite methods, but the both aren't equals, because
\\your parameters aren't equals.
```

Fonte: GUI class

3.4 Tratamento de Exceções

O uso de tratamento de exceções é fundamental afim de se garantir a confiabilidade e operacionalidade do sistema, bem como prover boas informações ao desenvolvedor sobre falhas que ocorrem em momento de execução. Na linguagem java, as exceções são tratadas por classes. Existem classes de exceções tratadas e não tratadas, quando há uma dificuldade em se identificar a falha que gerou a exeção. Abaixo encontra-se um exemplo de trecho de código em que tal tratamento ocorre.

Figura 3.4 – Tratamento de Exceção

```
try {
   return Integer.parseInt(scanner.nextLine());
}catch (NumberFormatException nfe) {
   nfe.printStackTrace();
}catch (Exception e) {
   e.printStackTrace();
}
```

Fonte: GUI class

4 CLASSES E ESTRUTURA DO SISTEMA

Todo o código fonte do sistema está organizado de forma a facilitar futuras atualizações de funcionalidades e operações. Os pacotes javas separam classes de acesso à dados, interface com usuário e controller. Abaixo, encontra-se os pacotes presentes no projeto:

Tabela 4.1 – Pacotes do Sistema

Informação	***Daemons* Adendo**
`app`	equivalente ao controller, contem o core do aplicativo.
`utils`	contem classes de ferramentas para o sistema, como logger, acesso a dados, dentre outros.
`data`	diretório com arquivos binários de dados.
`logs`	diretório com arquivos de texto de logs.

Fonte: Código fonte

Tabela 4.2 – Classes do Sistema

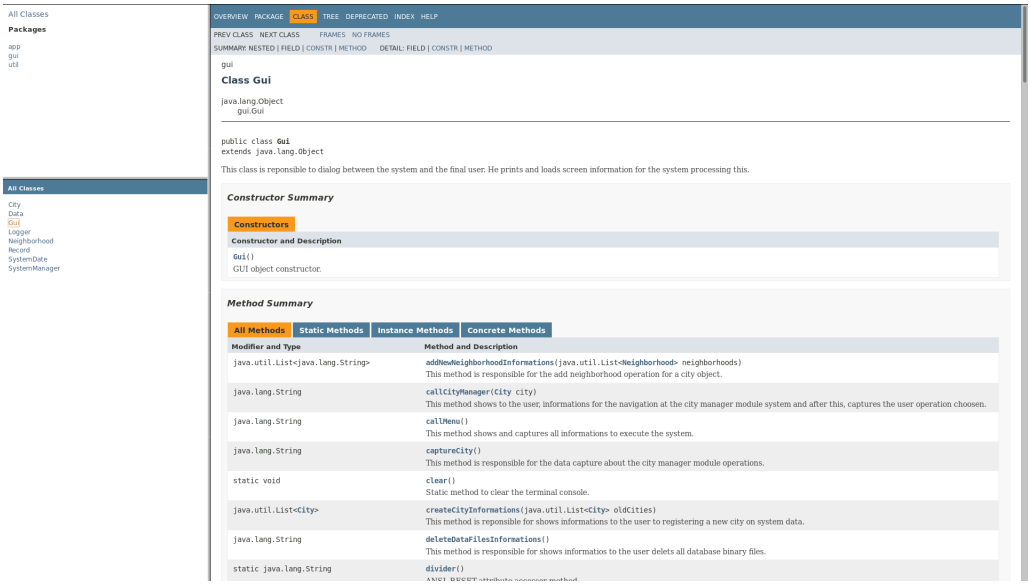
Informação	***Daemons* Adendo**
`Main`	contém o método principal que será chamado pela Java Virtual Machine.
`app/Records`	classe abstrata que descreve dados que serão armazenados no sistema.
`app/City`	classe concreta que implementa Record. Essa classe descreve os objetos Cidades que serão armazenados no sistema.
`app/Neighborhood`	classe concreta que implementa Record. Essa classe descreve os objetos Bairros que serão armazenados no sistema.
`SystemManager`	o core do sistema, que controla todas as operações do sistema.
`gui/Gui`	classe que implementa a interface visual para com o usuario.
`util/Data`	Classe que efetua acesso a dados.
`util/SystemDate`	Classe que captura e formata data capturada pelo sistema.
`util/Logger`	Classe que manipula e gera arquivos de registro log.

Fonte: Código fonte

5 DOCUMENTAÇÃO

Usufruindo-se do mecanismo Javadoc de documentação padronizada, todo o software está documentado de forma completa, com atributos, parametros e retornos explicados sobre cada método, bem como as classes, todos dispostos sobre arquivo de web (.html) no diretório docs do projeto. Todo o detalhamento da informação está transcorrido em Ingles.

Figura 5.1 – Exemplo da Documentação Javadoc - Classe GUI



Fonte: docs/