Centro de Ciências Exatas, Ambientais e de Tecnologias
Faculdade de Análise de Sistemas
Curso Sistemas de Informação

Projeto Integrado C
2º Trabalho do 1º Semestre de 2019

Este projeto utiliza os conceitos estudados nas disciplinas de Programação Orientada a Objetos e Estrutura e Recuperação de Informação.

O objetivo deste projeto é implementar uma aplicação Cliente-Servidor para consulta de músicas. Segue abaixo as principais informações e regras sobre este projeto:

Objetivo do Projeto: Desenvolver uma aplicação Cliente-Servidor onde vários usuários (*Client*) poderão simultaneamente acessar o servidor (*Server*) para fazer: consulta de músicas, seleção de músicas e cálculo de preço das músicas, caso o mesmo deseje adquiri-las.

Requisitos do Projeto:

1) **O que estará armazenado no servidor:** Este servidor deverá ter um banco de dados (*SQL Server*) com uma única tabela que contém os dados de músicas. Cada música deverá conter os seguintes dados: título, cantor (podendo ser um grupo), estilo, preço e duração. Os dados deverão ser preenchidos nesta tabela através de *script*, ou seja, não há a necessidade de se implementar no servidor uma lógica para manutenção dos dados no servidor via interface de usuário.

2) **Quais são as funcionalidades que deverão estar disponíveis no *Client*:** Ao acionar o programa *Client*, o usuário terá acesso às seguintes funcionalidades:

    a. **Consulta de músicas:** o programa *Client* deverá apresentar uma interface onde o usuário poderá fazer uma busca de músicas por: título, cantor ou estilo. Uma vez escolhida uma destas opções de busca e digitada a palavra que se busca, o Client submeterá esta solicitação ao *Server* que irá executar a busca na tabela, retornando a informação ao *Client* que irá apresentar em tela a lista das músicas encontradas no servidor.
    b. **Seleção de músicas:** o programa *Client* deverá permitir que o usuário, a partir da lista das músicas que foi consultada no passo (a), selecione as músicas que deseja adquirir, formando assim a sua lista de desejos.
    c. **Compra de músicas:** o programa *Client* deverá habilitar ao usuário a funcionalidade de compra de músicas, somente quando o usuário terminar de definir a sua lista de desejos. A funcionalidade de compra de músicas permitirá que o usuário calcule o preço das músicas que deseja adquirir. O cálculo do valor das músicas será realizado com base na lógica explicada no passo 3.

3) **Como calcular o preço das músicas que deseja adquirir:** o preço da lista de desejos será calculado com base em uma lógica relacionada à duração das músicas. Se a duração total das músicas for maior que 30 min e menor que 60 min, o usuário terá 10% de desconto sobre o valor total. Se a duração total das músicas for maior que 60 min e menor que 90 min, o usuário terá 20% de desconto sobre o valor total. Se a duração total das músicas for maior que 90 min, o usuário terá 30% de desconto sobre o valor total.

Dicas para Desenvolvimento do Projeto:

1) **Utilizar banco de dados no Servidor:** conforme descrito acima, o programa do servidor deverá ter condições de consultar o banco de dados de músicas. Para manipular os dados no banco de dados, você deverá utilizar os conceitos de DAOs e DBOs aprendidos na disciplina de Programação Orientada a Objetos.

2) **Estabelecer a comunicação Cliente-Servidor**: na disciplina de Programação Orientada a Objetos, vocês aprenderam que para desenvolvermos aplicações em Java que se comunicam via rede local ou via internet, devemos utilizar *sockets*. Vocês também viram que uma aplicação que utiliza sockets normalmente é composta por uma parte servidora e diversos clientes. Para que o servidor aceite vários clientes, simultâneos, deveremos utilizar o conceito de *Threads*. Cada cliente tem sua própria *Thread*, para que cada cliente estabeleça independentemente a sua conexão com o servidor.

3) **Definir um padrão de comunicação Cliente-Servidor:** na disciplina de Programação Orientada a Objetos, vocês também aprenderam que um cliente solicita determinado serviço ao servidor. O servidor processa a solicitação e devolve a informação ao cliente. Portanto, um dos passos fundamentais do seu projeto será você estabelecer um padrão de comunicação entre o cliente e o servidor, que chamaremos de Protocolo de Comunicação. Abaixo estão definidas as principais regras do protocolo de comunicação deste projeto:

Transação	comando	comp1	comp2	comp3	comp4	comp5
Consultar músicas de acordo com um critério de busca – na descrição dos requisitos do projeto vimos que o cliente terá que mandar uma solicitação ao servidor para que seja feita a consulta de músicas, de acordo com um critério de busca definido. Para isto, vamos definir que o cliente enviará o comando "COM" (consulta musica) informando o critério de busca ("titulo", "cantor" ou "estilo"), e a informação a ser buscada. Se o critério escolhido não for válido, o comando de resposta será "ERR". Se o critério escolhido for válido, e o servidor verificou que não há músicas cadastradas no banco de dados, o servidor envia o comando "FIC" (fim da consulta). Se o servidor enviará ao cliente um comando "MUS" quando 1 ou mais músicas forem encontrada. Para cada comando "MUS" o servidor informará os dados de uma música: título, cantor, estilo, preço e duração. Caso haja mais de uma música, para cada música haverá a execução de um comando "MUS". Quando concluído o envio dos dados de todas as músicas encontradas no servidor, é enviado pelo servidor o comando "FIC" (fim da consulta).	CON	Critério de Busca	Info Buscada			
	ERR					
	MUS	titulo	cantor	estilo	preco	duracao
	... (MUS poderá repetir 0 ou mais vezes e termina com FIC)					
	FIC					

Obs: Caso o grupo identifique a necessidade de outras transações, fique à vontade para complementar.

4) **Como implementar o padrão de comunicação definido acima:** será necessário que você implemente uma classe que definirá o objeto de "Comunicado" que será trocado entre o cliente e o servidor, seja na comunicação do cliente para o servidor, como do servidor para o cliente. Veja abaixo um exemplo de implementação. Basta você adequá-lo para o seu projeto (obs: não esqueça de incluir os métodos obrigatórios *hashCode*(), *equals*() *toString*(), caso necessário):

```
import java.io.Serializable;

public class Comunicado implements Serializable {
    private String comando,
                   complemento1,
                   complemento2,
                   complemento3,
                   complemento4,
                   complemento5;

    // Construtor de comunicado com 5 complementos
    public Comunicado (String comando,
                       String complemento1,
                       String complemento2,
                       String complemento3,
                       String complemento4,
                       String complemento5)
                       throws Exception {
        if (this.comando==null || comando.equals(""))
            throw new Exception ("Comando ausente");

        if (this.complemento1==null || complemento1.equals(""))
            throw new Exception ("Complemento1 ausente");

        if (this.complemento2==null || complemento2.equals(""))
            throw new Exception ("Complemento2 ausente");

        if (this.complemento3==null || complemento3.equals(""))
            throw new Exception ("Complemento3 ausente");

        if (this.complemento4==null || complemento4.equals(""))
            throw new Exception ("Complemento4 ausente");

        if (this.complemento5==null || complemento5.equals(""))
            throw new Exception ("Complemento5 ausente");

        this.comando      = comando;
        this.complemento1 = complemento1;
        this.complemento2 = complemento2;
        this.complemento3 = complemento3;
        this.complemento4 = complemento4;
        this.complemento5 = complemento5;
    }

    // Construtor de comunicado com 4 complementos
    public Comunicado (String comando,
                       String complemento1,
                       String complemento2,
                       String complemento3,
                       String complemento4)
                       throws Exception {
        if (this.comando==null || comando.equals(""))
            throw new Exception ("Comando ausente");

        if (this.complemento1==null || complemento1.equals(""))
```

```
            throw new Exception ("Complemento1 ausente");

        if (this.complemento2==null || complemento2.equals(""))
            throw new Exception ("Complemento2 ausente");

        if (this.complemento3==null || complemento3.equals(""))
            throw new Exception ("Complemento3 ausente");

        if (this.complemento4==null || complemento4.equals(""))
            throw new Exception ("Complemento4 ausente");

        this.comando      = comando;
        this.complemento1 = complemento1;
        this.complemento2 = complemento2;
        this.complemento3 = complemento3;
        this.complemento4 = complemento4;
    }

    // Construtor de comunicado com 3 complementos
    public Comunicado (String comando,
                       String complemento1,
                       String complemento2,
                       String complemento3)
                       throws Exception {
        if (this.comando==null || comando.equals(""))
            throw new Exception ("Comando ausente");

        if (this.complemento1==null || complemento1.equals(""))
            throw new Exception ("Complemento1 ausente");

        if (this.complemento2==null || complemento2.equals(""))
            throw new Exception ("Complemento2 ausente");

        if (this.complemento3==null || complemento3.equals(""))
            throw new Exception ("Complemento3 ausente");

        this.comando      = comando;
        this.complemento1 = complemento1;
        this.complemento2 = complemento2;
        this.complemento3 = complemento3;
    }

    // Construtor de comunicado com 2 complementos
    public Comunicado (String comando,
                       String complemento1,
                       String complemento2)
                       throws Exception {
        if (this.comando==null || comando.equals(""))
            throw new Exception ("Comando ausente");

        if (this.complemento1==null || complemento1.equals(""))
            throw new Exception ("Complemento1 ausente");

        if (this.complemento2==null || complemento2.equals(""))
            throw new Exception ("Complemento2 ausente");

        this.comando      = comando;
        this.complemento1 = complemento1;
        this.complemento2 = complemento2;
    }

    // Construtor de comunicado com 1 complementos
    public Comunicado (String comando,
                       String complemento1)
                       throws Exception {
```

```
        if (this.comando==null || comando.equals(""))
            throw new Exception ("Comando ausente");

        if (this.complemento1==null || complemento1.equals(""))
            throw new Exception ("Complemento1 ausente");

        this.comando      = comando;
        this.complemento1 = complemento1;
    }

    // Construtor de comunicado com sem complementos
    public Comunicado (String comando)
                       throws Exception {
        if (this.comando==null || comando.equals(""))
            throw new Exception ("Comando ausente");

        this.comando = comando;
    }

    public String getComando() {
        return comando;
    }

    public String getComplemento1() throws Exception {
        if (this.complemento1==null)
            throw new Exception ("Complemento1 indisponivel");

        return this.complemento1;
    }

    public String getComplemento2() throws Exception {
        if (this.complemento2==null)
            throw new Exception ("Complemento2 indisponivel");

        return this.complemento2;
    }

    public String getComplemento3() throws Exception {
        if (this.complemento3==null)
            throw new Exception ("Complemento3 indisponivel");

        return this.complemento3;
    }

    public String getComplemento4() throws Exception {
        if (this.complemento4==null)
            throw new Exception ("Complemento4 indisponivel");

        return this.complemento4;
    }

    public String getComplemento5() throws Exception {
        if (this.complemento5==null)
            throw new Exception ("Complemento5 indisponivel");

        return this.complemento5;
    }

    public String toString ()
    {
        String ret=this.comando;

        if (this.complemento1!=null)
            ret = ret+" "+complemento1;
```

```
        if (this.complemento2!=null)
            ret = ret+" "+complemento2;

        if (this.complemento3!=null)
            ret = ret+" "+complemento3;

        if (this.complemento4!=null)
            ret = ret+" "+complemento4;

        if (this.complemento5!=null)
            ret = ret+" "+complemento5;

        return ret;
    }

    public boolean equals (Object obj)
    {
        if (this==obj)
            return true;

        if (obj==null)
            return false;

        if (this.getClass()!=obj.getClass())
            return false;

        Comunicado comunicado = (Comunicado)obj;

        if (!this.comando.equals(comunicado.comando))
            return false;

        if ((this.complemento1==null && comunicado.complemento1!=null) ||
            (this.complemento1!=null && comunicado.complemento1==null))
            return false;

        if (this.complemento1!=null && comunicado.complemento1!=null &&
           !this.complemento1.equals  (comunicado.complemento1))
            return false;

        if ((this.complemento2==null && comunicado.complemento2!=null) ||
            (this.complemento2!=null && comunicado.complemento2==null))
            return false;

        if (this.complemento2!=null && comunicado.complemento2!=null &&
           !this.complemento2.equals  (comunicado.complemento2))
            return false;

        if ((this.complemento3==null && comunicado.complemento3!=null) ||
            (this.complemento3!=null && comunicado.complemento3==null))
            return false;

        if (this.complemento3!=null && comunicado.complemento3!=null &&
           !this.complemento3.equals  (comunicado.complemento3))
            return false;

        if ((this.complemento4==null && comunicado.complemento4!=null) ||
            (this.complemento4!=null && comunicado.complemento4==null))
            return false;

        if (this.complemento4!=null && comunicado.complemento4!=null &&
           !this.complemento4.equals  (comunicado.complemento4))
            return false;

        if ((this.complemento5==null && comunicado.complemento5!=null) ||
            (this.complemento5!=null && comunicado.complemento5==null))
```

```
            return false;

        if (this.complemento5!=null && comunicado.complemento5!=null &&
           !this.complemento5.equals  (comunicado.complemento5))
            return false;

        return true;
    }

    public int hashCode ()
    {
        int ret=1;

        ret = 2*ret + this.comando.hashCode();

        if (this.complemento1!=null)
            ret = 2*ret + complemento1.hashCode();

        if (this.complemento2!=null)
            ret = 2*ret + complemento2.hashCode();

        if (this.complemento3!=null)
            ret = 2*ret + complemento3.hashCode();

        if (this.complemento4!=null)
            ret = 2*ret + complemento4.hashCode();

        if (this.complemento5!=null)
            ret = 2*ret + complemento5.hashCode();

        return ret;
    }

    // como nao ha metodos, alem dos construtores que
    // alterem atributos do this, nao teremos clone e
    // nem construtor de copia
}
```

5) **Como o cliente deve manipular as músicas recebidas do servidor:** o cliente ao receber do servidor informações de uma ou mais músicas, deverá armazenar estas informações em uma lista encadeada, chamada **lista_de_musicas**. Ao selecionar nesta **lista_de_musicas** as músicas que deseja comprar, você deverá mover estes itens da **lista_de_musicas** para uma **lista_de_desejos**, que também deverá ser implementada utilizando o conceito de lista encadeada visto na disciplina de Estrutura e Recuperação de Informação (não podem ser utilizadas classes prontas da linguagem). Na sequência, você pode usar a **lista_de_desejos** para calcular o preço das músicas que deseja comprar, utilizando a lógica de descontos baseado na duração total de músicas compradas (explicada acima).

Exigências para a Implementação deste Projeto:

Segue abaixo, uma lista das exigências que serão feitas pelos professores:

1. O trabalho deverá ser executado em grupos de até 3 alunos (os mesmos alunos do Projeto 1);
2. Uma boa estrutura de orientação a objetos deverá ser criada pelos grupos e será alvo de avaliação;

3. Todas as classes programadas deverão ter os métodos obrigatórios equals, toString e hashCode, além de, quando for o caso, também terem os métodos clone, construtor de cópia e compareTo (lembrem-se de fazer com que a classes implementem Cloneable e Comparable neste caso);
4. Todas as classes deverão ser documentadas com JavaDoc;
5. Para implementar a conectividade cliente e servidor, o grupo deverá obrigatoriamente utilizar o protocolo de comunicação definido pelos docentes;
6. O grupo deverá planejar o seu **cronograma de desenvolvimento** e enviar a professora 1 dia após a explicação deste trabalho à turma. Este cronograma deve ser definido adotando o escopo de desenvolvimento em 3 semanas, conforme períodos definidos abaixo:

TURMA – Profª Daniele Frosoni:

- 1ª Semana (26/Abr a 02/Mai/2019):
- 2ª Semana (03/Mai a 09/Mai/2019):
- 3ª Semana (10/Mai a 16/Mai/2019):
- 4ª Semana (17/Mai a 23/Mai/2019):
- 23/Mai/2019: Entrega Final e apresentações do Projeto 2 para Avaliação Final da Professora
- 30/Mai/2019: Continuação das apresentações do Projeto 2 para Avaliação Final da Professora

TURMA – Profª Eliane Azevedo:

- 1ª Semana (27/Abr a 03/Mai/2019):
- 2ª Semana (04/Mai a 10/Mai/2019):
- 3ª Semana (11/Mai a 17/Mai/2019):
- 4ª Semana (18/Mai a 24/Mai/2019):
- 24/Mai/2019: Entrega Final e apresentações do Projeto 2 para Avaliação Final da Professora
- 31/Mai/2019: Continuação das apresentações do Projeto 2 para Avaliação Final da Professora

Obs: As professoras de Projeto Integrado irão utilizar este cronograma como referência para avaliação das entregas parciais.

Bom trabalho!	**Profs André Carvalho, Daniele Frosoni, Eliane Azevedo e Patrícia Nogueira**	**Campinas, 25/Abr/2019**