MC302
Primeiro semestre de 2017

Laboratório 10

Professores: Esther Colombini (esther@ic.unicamp.br) e Fábio Luiz Usberti (fusberti@ic.unicamp.br)
PEDs: (Turmas ABCD) Elisangela Santos (ra149781@students.ic.unicamp.br), Lucas Faloni (lucasfaloni@gmail.com), Lucas David (lucasolivdavid@gmail.com), Wellington Moura (wellington.tylon@hotmail.com)
PEDs (Turmas EF) Natanael Ramos (naelr8@gmail.com), Rafael Arakaki (rafaelkendyarakaki@gmail.com)
PAD: (Turmas ABCD) Igor Torrente (igortorrente@hotmail.com)
PAD: (Turmas EF) Bleno Claus (blenoclaus@gmail.com)

---

1 Objetivo

O objetivo desta atividade consiste em praticar os conceitos de escrita e leitura em arquivos utilizando a linguagem Java.

2 Atividade

Para esse laboratório, iremos precisar das classes **Carta**, **Lacaio**, **Magia**, **Dano**, **Buff** e **DanoArea** criadas nos laboratórios anteriores. Como foi visto nas aulas **19** e**20**, Java provê meios para fazer serialização de objetos, escrevendo e lendo dados de objetos de arquivos salvos em disco. Nesse laboratório iremos criar duas classes, **Escritor** e **Leitor** dentro de um novo pacote chamado `io` (acrônimo para o inglês *Input/Output*). Tais classes serão usadas para escrever e ler dados de objetos de arquivos texto, tais objetos serão instâncias das classes previamente mencionadas.

A utilidade da serialização de objetos no nosso contexto se daria se caso quiséssemos salvar um estado do jogo em disco e recuperar tal estado posteriormente. Também seria útil se tal jogo fosse feito entre dois jogadores através de uma rede de computadores, dessa maneira, através da serialização de objetos seria possível criar um protocolo de comunicação entre os diferentes clientes do jogo.

Para escrever em arquivos sugere-se o uso da classe `FileWriter`[1] e para leitura a classe `Scanner`[2], ambas vistas em sala de aula.

2.1 Formato do arquivo

Nessa seção, será descrito o formato padrão esperado para os arquivos texto que mantém as informações dos objetos. Para cada objeto escrito no artigo adicione a `obj` seguida de espaço e o nome da classe a qual o objeto foi instanciado. Essa *tag* vai ser usada no início e no final da descrição do objeto. Segue um exemplo:

```
obj <nome-classe>
...
obj <nome-classe>
```

[1] `https://docs.oracle.com/javase/7/docs/api/java/io/FileWriter.html`
[2] `https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html`

Onde `<nome-classe>` pode ser substituído por `Lacaio`, `Buff`, etc. Note que em um mesmo arquivo podem existir vários objetos. Entre as *tags* `obj` será feita a descrição dos atributos do respectivo objeto. Para cada atributo adiciona-se uma linha com o nome do atributo seguido de um espaço em branco e seu valor. Segue um exemplo para um objeto da classe `Lacaio`:

```
obj Lacaio
id 57ab6ed5-e255-4d27-a27d-1c4b840eb1bf
nome zr1kz
custoMana 2
ataque 3
vidaAtual 8
vidaMaxima 10
habilidade EXAUSTAO
obj Lacaio
```

Note que os atributos herdados da classe `Carta` (`id`, `nome` e `custoMana`) também estão presentes.

2.2 Classe `Escritor`

A classe `Escritor` é responsável por prover métodos auxiliares para a escrita dos atributos de cada objeto no arquivo. Os métodos que devem ser implementados são listados a seguir:

- `void escreveAtributo(String nomeAtributo, String valor)`: Escreve um atributo no arquivo seguindo as especificações do formato padrão.
- `void escreveDelimObj(String nomeObj)`: Adiciona uma linha ao arquivo indicando o início/término da especificação de um objeto.
- `void finalizar()`: Fecha o arquivo onde os dados foram escritos.

Fica a critério do aluno se ao escrever em um arquivo qual a decisão que deve ser tomada caso o arquivo no qual se fará a escrita já exista. Ou seja, se o novo conteúdo sera anexado ao arquivo ou se ele será sobrescrito.

2.3 Classe `Leitor`

A classe `Leitor` basicamente lê os objetos de um arquivo texto e retorna-os em uma lista. Tal método tem a seguinte assinatura:

```
1 public List<ILaMaSerializable> leObjetos()
```

Listing 1: Interface de objetos serializáveis.

Pode-se fazer a suposição de que a ordem para ler os atributos é a mesma da ordem em que foram escritas.

2.4 Interface

O objetivo aqui é que cada objeto seja "auto-serializável", i.e., cada objeto recebe como parâmetro um objeto da classe `Escritor` e escreve seus atributos no arquivo. De maneira a deixar tal comportamento genérico, crie uma interface chamada `ILaMaSerializable`:

```java
package base;

import io.Escritor;

import java.io.IOException;

/**
 * Created by nael on 5/22/17.
 */
public interface ILaMaSerializable {
    void escreveAtributos(Escritor fw) throws IOException;
}
```

Listing 2: Interface de objetos serializáveis.

Todas as classes que serão serializadas **devem** implementar essa interface.

3 Sugestões

Ao escrever e ler em arquivos, os dados são lidos e escritos em formato de *String*. Alguns atributos das classes são do tipo inteiro ou algum tipo complexo como o enumerador `TipoLacaio` ou o id do tipo `UUID`. Para fazer a conversão entre os respectivos tipos, faça uso dos métodos `toString`, `valueOf` e `fromString`. Segue um exemplo para deixar mais claro:

```java
// String para Inteiro
String s = "10";
int i = Integer.valueOf(s);

// Inteiro para String
i = 20;
s = String.valueOf(i);

// UUID para String
UUID id = UUID.randomUUID();
s = id.toString();

// String para UUID
id = UUID.fromString(s);
```

Listing 3: Exemplo de conversão entre tipos.

Classes do Java que lidam com arquivos podem lançar exceções de variados tipos. Alguns deles são:

- `FileNotFoundException`: Sinaliza quando o arquivo solicitado não existe.
- `UnsupportedEncodingException`: A codificação do arquivo não é suportada.

Uma lista de exceções que podem ser lançadas pelas classes pertencentes ao pacote `java.io` podem ser encontrada na documentação[3] na seção `Exception Summary`. Certifique-se então de cercar as chamadas dos métodos que utilizam Entrada/Saída com blocos `try..catch`.

A classe `Scanner` possui um método chamado `hasNext` que identifica se o arquivo terminou ou não. Também possui um método chamado `next` que lê os caracteres do arquivo até encontrar um delimitador. Java tem o espaço em branco como um dos delimitadores padrões.

[3] https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html

4 Validação

De maneira a validar a implementação, crie um objeto de cada uma das classes: **Lacaio**, **Dano**, **Buff** e **DanoArea**. Serialize cada um deles utilizando a classe `Escritor` e em seguida leia os objetos do mesmo arquivo gerado usando a classe `Leitor`. Imprima os campos dos objetos antes e depois da serialização.

5 Questões

Responda as seguintes questões em um **arquivo texto** e submeta junto ao código no Moodle:

- A classe `Baralho` possui vários objetos do tipo `Carta`, como poderia ser feita uma serialização dessa classe?

6 Submissão

Para submeter a atividade, utilize o Moodle (`https://www.ggte.unicamp.br/ea`). Crie um arquivo texto com as respostas para cada item da seção tarefas e as saídas geradas pelo código. Compacte o código-fonte contido no diretório **src** juntamente com arquivo de respostas no formato .zip ou similar e nomeie-o **Lab10-000000.zip**, trocando '000000' pelo seu número de RA. Submeta o arquivo na seção correspondente para esse laboratório no moodle da disciplina MC302.

Certifique-se de entregar um código compilável, incluindo todas as subpastas dos pacotes em prol de facilitar a correção.

Datas de entrega

- Dia **06/06** Turma **ABCD** até às 23:55h
- Dia **03/06** Turma **EF** até às 23:55h