UNIVERSIDADE DO OESTE DE SANTA CATARINA
ÁREA DAS CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE ENGENHARIA DE COMPUTAÇÃO **FASE – 3ª**
Professor: Karl Phillip Buhr **23-05-2016**
Componente Curricular: Programação Orientada a Objetos
Avaliação A1/02

Honestidade Acadêmica:

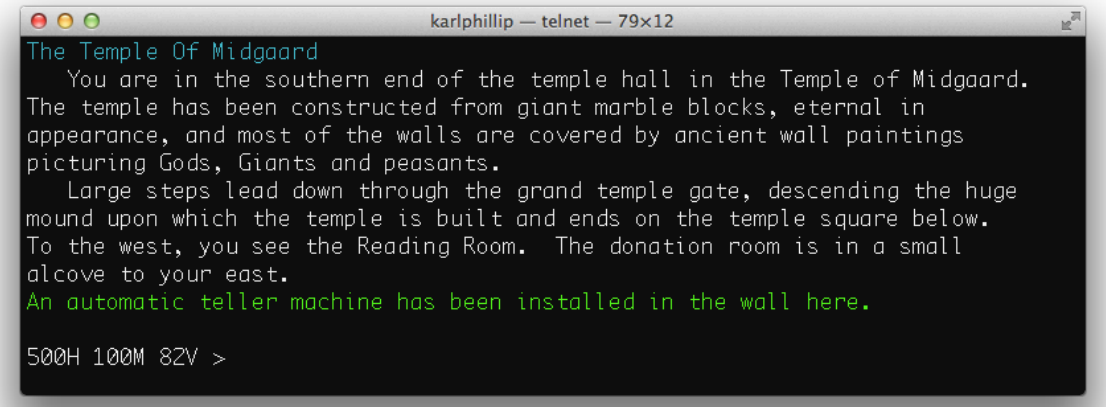
A universidade está comprometida em preservar a reputação do seu diploma. Para garantir que cada diploma outorgado possua o significado que ele realmente representa, nesta disciplina exige-se que o acadêmico:

- Entregue um trabalho original, isto é, desenvolvido de forma integral e individual pelo próprio acadêmico.
- Não utilize uma solução parcial ou completa obtida de qualquer outra fonte (outra pessoa, Internet, etc).
- Não discuta seu trabalho com outros colegas.
- Proteja seu trabalho e evite que ele seja roubado. Não compartilhe seu trabalho com ninguém.
- Não entregue um trabalho que não é seu.

O não comprimento destas regras caracteriza desonestidade acadêmica e a sua nota pode ser zerada. Além disso, penalidades como suspensão ou expulsão podem ser impostas pelo departamento de Engenharia de Computação.

Aventura Baseada em Texto

Neste trabalho você demonstrará domínio básico sobre Programação Orientada a Objetos ao implementar uma versão mínima de um jogo de aventura baseado em texto.



Jogos baseados em texto são mais fáceis de escrever e requerem menos poder de processamento que jogos que contém gráficos, sendo muito populares entre as décadas de 70 e 90. Jogos neste estilo utilizam apenas caracteres de texto ao invés de gráficos e bitmaps. Este tipo de jogo ainda é bastante explorado por desenvolvedores iniciantes para se familiarizarem de forma divertida com uma linguagem de programação.

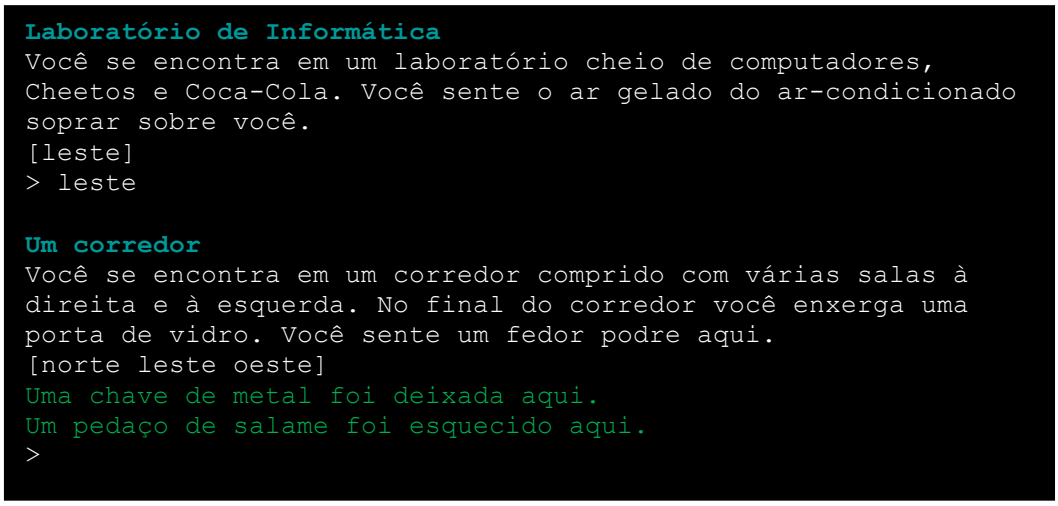
O jogo deve ser implementado em Java e deve usar os conceitos de POO que foram estudados durante a disciplina. Para facilitar o entendimento das tarefas, este documento está dividido nas seguintes seções:

1. **Visão geral do jogo de aventura**
2. **Visão geral das classes do jogo**
3. **Visão geral dos arquivos de dados**
4. **A mecânica da construção do mundo**
5. **Observações**

Seção 1: Visão geral do jogo de aventura

O jogo de aventura proposto neste trabalho é baseado em um mundo virtual, no qual o jogador pode locomover-se de um ambiente para outro, colecionar tesouros e outros artefatos.

Os ambientes virtuais, conhecidos como **salas**, são mostrados para o jogador através de descrições textuais narrativas que oferecem um senso de geografia. O jogador interage com o jogo através de **comandos**. Por exemplo, para se locomover o usuário digita comandos específicos que atuam como uma indicação da direção do movimento:



De acordo com a figura acima, o jogador executou o comando ***leste*** para se locomover até a sala que está conectada à direita, chamada *Um corredor*.

Ao entrar em uma sala, o jogo imprime na tela para o jogador:

- O nome da sala;
- A descrição completa do ambiente;
- A lista das saídas (norte, sul, leste, oeste);
- A lista de itens que estão disponíveis para o jogador interagir;
- O *prompt de comando*, simbolizado pelo caractere **>**.

É importante perceber que a descrição da sala aparece uma vez, no exato momento em que o jogador entra na sala. Entretanto, a qualquer momento o jogador pode obter esta descrição executando o comando **olhar** que imprime essas informações novamente.

Seção 2: Visão geral das classes do jogo

O jogo deve implementar todas as classes descritas nesta seção, com exceção da classe **DBUtils**, que é fornecida pelo professor. O acadêmico tem total liberdade para projetar os métodos que achar necessário para resolver as tarefas descritas em cada classe:

- Classe **Aventura**: esta é a classe principal do programa. Ela deve:
  - Inicializar o mundo virtual (carregar os dados necessários - salas e itens - do disco rígido para montar o cenário do jogo);
  - Exibir a mensagem de boas-vindas ao jogador;
  - Apresentar o menu do jogo para o usuário:
    1) **Entrar no jogo**
    2) **Sair**
  - Ao entrar no jogo, o programa deve permitir o usuário:
    - Escolher o nome de seu personagem e definir uma senha;
    - Escolher o sexo do personagem (M/F);
    - Escolher uma entre 5 profissões disponíveis para o personagem;
  - Posicionar o Jogador na sala inicial e executar o *loop* do jogo. O *loop* deve:
    - Imprimir o *prompt* de comando para o jogador;
    - Esperar que o usuário digite um comando para interagir com o jogo.

- Classe **Jogador**: Apenas um objeto deste tipo deve existir durante toda a execução do programa. Ele deve ser instanciado no momento em que o usuário entra no jogo. A classe Jogador deve armazenar:
  - O nome do personagem, a senha, o sexo e a profissão do personagem;
  - A lista de itens que o jogador carrega;
  - A quantidade máxima de peso que o personagem pode carregar;
  - A sala em que ele está no momento.

- Classe **Sala**: Esta classe representa um ambiente dentro do mundo virtual. A classe deve armazenar no mínimo:
  - O nome da sala;
  - A descrição breve do cenário que ela representa;
  - A lista de possíveis saídas (norte/sul/leste/oeste);
  - A lista de itens que estão na sala.

- Classe **Item**: Esta classe representa um item (ou artefato) no mundo virtual, e é algo que possui apenas um dono: ou o jogador, ou uma sala, mas nunca os dois ao mesmo tempo. A classe deve armazenar no mínimo:
  - O nome do item;
  - A descrição breve do que ele representa;
  - O peso do item;

- Classe **Comando**: Esta classe representa uma instrução que o usuário pode executar para interagir com o jogo. A classe deve armazenar no mínimo:
  - O nome do comando;
  - Uma referência ao objeto Jogador.

- Classe **DButils**: fornecida pelo professor. Esta classe provê métodos que você deve utilizar para carregar dados do disco rígido para popular o mundo virtual com salas e itens. Ela serve para ler 2 tipos diferentes de arquivo: o arquivo que contém as descrições das salas do jogo, e o arquivo que contém a descrição dos itens. A tabela a seguir apresenta os métodos públicos que são oferecidos por esta classe e uma breve descrição do que eles fazem:

Métodos da classe DBUtils

Boolean	**carregar**(String nome_arq) Este método lê dados de um arquivo no formato *.wld* ou *.obj* do disco rígido. Ele deve ser chamado antes que qualquer outro método para popular o objeto DButils com informações. **nome_arq**: caminho do arquivo .wld ou .obj no disco. **retorno**: *true* ou *false* para indicar o sucesso da operação.
int[]	**getObjVnums**() Retorna um array de inteiros com o **vnum** de todos os artefatos que estavam declarados no arquivo.
String[]	**getItemNomes**() Retorna um array de strings com o **nome** de todos os artefatos que estavam declarados no arquivo.
String[]	**getItemDesc**() Retorna um array de strings com a **descrição** de todos os artefatos que estavam declarados no arquivo.
int[][]	**getSalaSaidas**() Retorna um array de inteiros que possui as **saídas** (4 números inteiros) de cada sala declarada no arquivo.
int[][]	**getSalaObjs**() Retorna um array de inteiros com o **vnum dos itens** de cada sala declarada no arquivo.
int[]	**getObjPeso**() Retorna um array de inteiros com o **peso dos itens** listados no arquivo.

Seção 3: Visão geral dos arquivos de dados

As informações do cenário virtual (salas e itens) não fazem parte do programa, isto é, não são definidas no código-fonte. Elas são especificadas em arquivos no disco rígido. Esta abordagem permite trabalharmos na arquitetura do mundo virtual, adicionando ou removendo salas e itens, sem precisar alterar o código-fonte do programa.

Um único arquivo de texto no disco deverá conter informações das salas e outro deverá conter informações dos itens. O padrão apresentado aqui serve apenas como um exemplo para demonstrar e orientar como você pode construir seus ambientes e incrementar o mundo virtual. Jogos deste tipo são avaliados de diversas maneiras, mas uma das mais importantes é o número e a qualidade das áreas disponíveis no cenário. São esses atributos que que fazem o jogo ser original.

Cada sala e item dentro de uma determinada área recebe um número virtual (**vnum**). Este número é o que identifica uma sala ou um item dentro do jogo. Estes números são independentes. Isto quer dizer que é possível existir uma sala com o **vnum** 3001 e ao mesmo tempo existir um item com o **vnum** 3001.

Quando definimos e referenciamos as partes de uma área, o arquiteto dela sempre se refere a seus componentes pelo **vnum** e nunca pelo nome da sala/item. **vnums** nunca são vistos pelo jogador.

Na nossa implementação minimalista do jogo, o mundo virtual é definido por:

- Arquivos de mundo: extensão **.wld**. Contém as definições das salas e suas ligações umas com as outras.
- Arquivos de itens: extensão **.obj**. Define as armas, armaduras, tesouros e outros itens que são manipuláveis pelo jogador.

Algumas convenções são essenciais para definir o formato-padrão dos arquivos e possibilitar que outros arquitetos possam contribuir com a criação de novas áreas para o jogo. Vejamos agora o formato de um arquivo **.obj**:

- Cada item definido dentro do arquivo deve começar o caractere (**#**) seguido do **vnum** que vai ser utilizado para identificar aquele item;
- A segunda linha deve apresentar o nome do item e terminar com o caractere (**~**);
- A terceira deve apresentar uma breve descrição do item e terminar com o caractere (**~**);
- A quarta linha informa o peso (em gramas) do item;
- O arquivo deve terminar com o sinal de dólar (**$**) para informar o fim da definição de dados.

Vejamos um exemplo de um arquivo que define itens (***itens.obj***):

```
#3000
barril~
Um barril de cerveja foi largado aqui.~
20000
#3001
chave~
Uma chave de metal foi deixada aqui.~
150
#3002
salame~
Um pedaço de salame foi esquecido aqui.~
450
$
```

Arquivos que definem salas são bastante similares. Entretanto, eles apresentam as ligações com as outras salas e a lista de itens que se encontra dentro dela. Vejamos agora o formato de um arquivo **.wld**:

- Depois do **vnum**, do **nome** da sala e sua **breve descrição**, vêm a definição das ligações que ela possui com outras salas. Nesta linha consta os *vnums* das salas com as quais esta se conecta. É importante respeitar a seguinte ordem de ligação das salas: norte, sul, leste e oeste.
- Neste campo, o número zero é utilizado como **vnum** para representar que a sala não está conectada com nenhuma outra naquela direção.
- Na linha seguinte, é informado o **vnum** de todos os itens que estão presentes na sala. Este número deve estar definido no arquivo **.obj**. Para indicar que a sala não possui nenhum item, utilizamos o número 0.

Vejamos um exemplo real de um arquivo que define algumas salas (***salas.wld***):

```
#5001
Laboratório de Informática 3~
Você se encontra em um laboratório cheio de computadores, Cheetos e Coca-
Cola. Você sente o ar gelado do ar-condicionado soprar sobre você.~
0 0 5002 0
0
#5002
Um corredor~
Este é um corredor comprido com várias salas a sua direita e a sua
esquerda. No final do corredor você enxerga uma porta de vidro. Você
sente um cheiro podre aqui.~
0 0 0 5001
3001 3002
$
```

Seção 4: A mecânica da construção do mundo

Para facilitar a carga de itens e salas que representam o mundo virtual, você deve utilizar a classe **DButils** disponibilizada pelo professor para carregar dados de arquivos **.wld** e **.obj** do disco rígido. Você tem total liberdade para modificar a classe e seus métodos se achar necessário.

Lembre-se que depois que estes dados forem lidos do disco, você deve criar objetos -- conforme as classes descritas na seção 2 -- para encapsular estes dados.

Abaixo pode-se observar dois exemplos de utilização da classe DButils:

```
/* Imprimir todos os dados armazenados em um arquivo .wld */

DButils sala_u = new DButils();
sala_u.carregar("test.wld");

for (int x = 0; x < sala_u.getObjVnums().length; x++)
{
    System.out.println("SALA Vnum: " + sala_u.getObjVnums()[x]);
    System.out.println("SALA Nome: " + sala_u.getItemNomes()[x]);
    System.out.println("SALA Desc: " + sala_u.getItemDesc()[x]);

    System.out.print("SALA Saidas: ");
    for (int j = 0; j < sala_u.getSalaSaidas()[x].length; j++)
        System.out.print(sala_u.getSalaSaidas()[x][j] + " ");
    System.out.println();

    System.out.print("SALA Lista de itens: ");
    for (int j = 0; j < sala_u.getSalaObjs()[x].length; j++)
        System.out.print(sala_u.getSalaObjs()[x][j] + " ");
    System.out.println();
}

/* Imprimir todos os dados armazenados em um arquivo .obj */

DButils objs_u = new DButils();
objs_u.carregar("test.obj");

for (int x = 0; x < objs_u.getObjVnums().length; x++)
{
    System.out.println("OBJ Vnum: " + objs_u.getObjVnums()[x]);
    System.out.println("OBJ Nome: " + objs_u.getItemNomes()[x]);
    System.out.println("OBJ Desc: " + objs_u.getItemDesc()[x]);
    System.out.println("OBJ Peso: " + objs_u.getObjPeso()[x]);
}
```

Seção 5: Observações

Os seguintes comandos devem ser suportados pelo jogo:

Lista de Comandos

norte	Movimenta o jogador para a sala à frente.
sul	Movimenta o jogador para a sala atrás.
leste	Movimenta o jogador para a sala à direita.
oeste	Movimenta o jogador para a sala à esquerda.
olhar	Sem parâmetros, exibe a descrição da sala. Com 1 parâmetro, exibe a descrição do item.
inv	Exibe a lista (nomes) de itens que o jogador carrega.
pegar	Transfere um item que está no chão da sala para o inventário do personagem. Esta operação deve verificar se o usuário atingiu o limite máximo de peso antes de adicionar mais um item ao seu inventário.
largar	Remove um item do inventário e coloca-o no chão da sala. Esta operação deve diminuir o peso que o personagem está carregando.
sair	Sair do jogo.

Para desenvolver este trabalho, sua implementação deve:

- Criar uma *thread* secundária para carregar os dados das salas e dos itens dos arquivos utilizando a classe **DButils** para construir um mundo completamente novo com 15 salas e 10 itens;
- Implementar as classes descritas na *seção 2*;
- Garantir que os atributos das classes sejam acessíveis apenas através de métodos *set/get*;
- Implementar 1 classe abstrata;
- Implementar 1 *interface*;