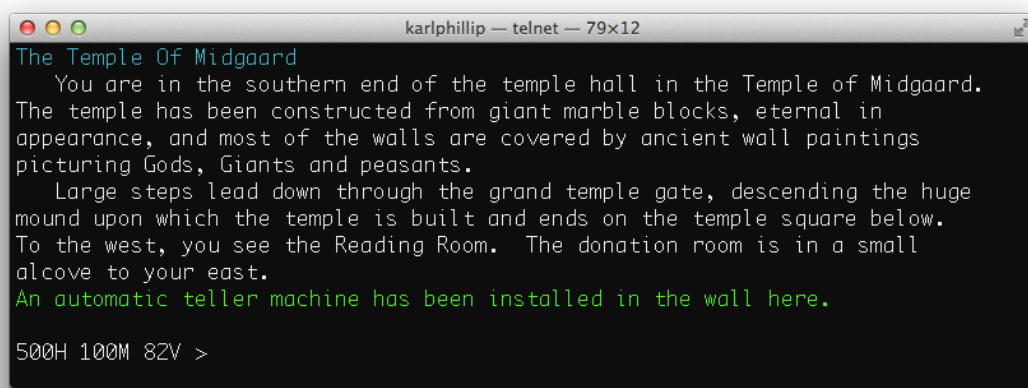


1. O trabalho é individual e sem consulta. A interpretação das informações abaixo fazem parte da avaliação.
2. É permitido tirar dúvidas com o professor, entretanto, se um estudante obter ou conseguir ajuda inapropriada de terceiros para realizar esta atividade ele estará sujeito a ter sua nota zerada.
3. Desonestidade acadêmica: plágio e outras formas de trapaça são ofensas sérias, e o acadêmico estará sujeito as penalidades estabelecidas pela universidade.

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](#).



```
karlphillip — telnet — 79x12
The Temple Of Midgaard
  You are in the southern end of the temple hall in the Temple of Midgaard.
  The temple has been constructed from giant marble blocks, eternal in
  appearance, and most of the walls are covered by ancient wall paintings
  picturing Gods, Giants and peasants.
  Large steps lead down through the grand temple gate, descending the huge
  mound upon which the temple is built and ends on the temple square below.
  To the west, you see the Reading Room. The donation room is in a small
  alcove to your east.
  An automatic teller machine has been installed in the wall here.

500H 100M 82V >
```

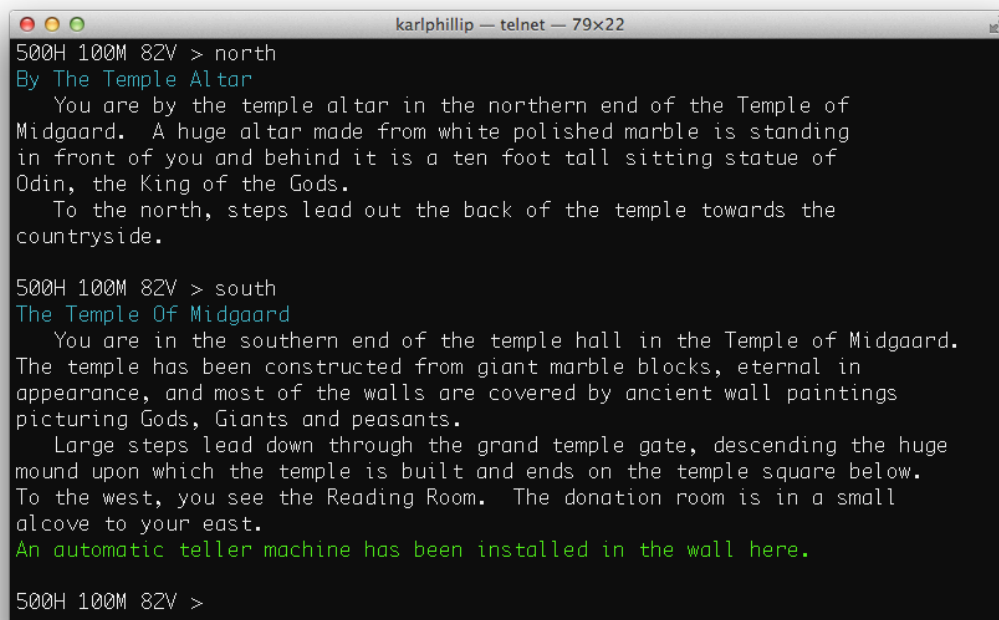
Jogos baseados em texto são mais fáceis de escrever e requerem menos poder de processamento que jogos que contém gráficos, e eram bastante populares entre as décadas de 70 e 90. Jogos assim utilizam apenas caracteres de texto ao invés de gráficos e *bitmaps*. Este tipo de jogo ainda é bastante explorado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

O jogo deve ser desenvolvido em Java e implementar os conceitos de POO que foram apresentados 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 que será desenvolvido é baseado em um mundo virtual, no qual você, como jogador, pode se locomover de um local para outro e colecionar tesouros e outros artefatos. Os ambientes virtuais, mais conhecidos como **salas**, são mostrados para você através de descrições textuais narrativas que fornecem 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:



```
karlphillip — telnet — 79x22
500H 100M 82V > north
By The Temple Altar
  You are by the temple altar in the northern end of the Temple of
  Midgaard. A huge altar made from white polished marble is standing
  in front of you and behind it is a ten foot tall sitting statue of
  Odin, the King of the Gods.
  To the north, steps lead out the back of the temple towards the
  countryside.

500H 100M 82V > south
The Temple Of Midgaard
  You are in the southern end of the temple hall in the Temple of Midgaard.
  The temple has been constructed from giant marble blocks, eternal in
  appearance, and most of the walls are covered by ancient wall paintings
  picturing Gods, Giants and peasants.
  Large steps lead down through the grand temple gate, descending the huge
  mound upon which the temple is built and ends on the temple square below.
  To the west, you see the Reading Room. The donation room is in a small
  alcove to your east.
  An automatic teller machine has been installed in the wall here.

500H 100M 82V >
```

No exemplo da figura acima, o jogador inicia sua jornada na sala *The Temple of Midgaard* e decide seguir em direção ao *norte*, e para isso executa o comando **north**. Esse comando conduz o jogador até a próxima sala, *By The Temple Altar*, onde ele recebe uma descrição completa da sala que ele acabou de entrar. A descrição da sala aparece apenas uma vez, no momento em que o jogador entra em um novo ambiente. Nesta versão do jogo, é possível que o jogador obtenha a descrição da sala a qualquer momento através do comando **look** (olhar).

Em seguida, o jogador decide retornar a sala anterior executando o comando **south**, que faz ele voltar para a sala de onde veio, e novamente receber a descrição da sala *The Temple of Midgaard*. É importante observar que **descrição completa** da sala inclui:

- o nome da sala;
- uma breve descrição daquele ambiente;
- a lista de possíveis saídas da sala (norte, sul, leste, oeste);
- a lista de itens que se encontram nela (na imagem acima, identificados pela cor verde).

Seção 2: visão geral das classes do jogo

- classe **Aventura**: esta é a classe principal do programa e é sua responsabilidade implementá-la. A classe deve:
 - Inicializar o mundo virtual, ou seja, 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 fazer *login*:
 - 1) Entrar no jogo
 - 2) Sair
 - Permitir o usuário escolher uma de 4 profissões existentes para seu personagem;
 - Posicionar o Jogador na sala inicial e executar o *loop* do jogo. Este *loop* deve:
 - Imprimir o *prompt* de comando para o jogador;
 - Esperar que o usuário digite um comando para interagir com o jogo.
- classe **Jogador**: é sua responsabilidade implementá-la. Esta classe representa a entidade jogador dentro do programa. Apenas um objeto deste tipo existe durante toda a execução do programa, e ele deve ser instanciado no momento em que o usuário decide entrar no jogo. A classe Jogador deve armazenar no mínimo:
 - o nome do personagem;
 - a senha dele;
 - sua profissão;
 - a lista de itens que o usuário carrega;
 - a quantidade de peso máximo que o personagem pode carregar;
 - a sala em que ele está.
- classe **Sala**: é sua responsabilidade implementá-la. Esta classe representa uma sala dentro do ambiente 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**: é sua responsabilidade implementá-la. 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**: é sua responsabilidade implementá-la. 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 cópia do objeto Jogador.
- classe **DButils**: fornecida pelo professor. Esta classe provê métodos que você deve utilizar para carregar dados do mundo virtual a partir do disco rígido. Ela suporta 2 formatos de arquivo: um utilizado para descrever as salas, e o outro usado para descrever os itens. A tabela a seguir apresenta a assinatura dos métodos públicos disponibilizados na classe e o que eles fazem:

Métodos

boolean	carregar (String nome_arq) Este método lê dados de um arquivo no formato <i>.wld</i> ou <i>.obj</i> do disco rígido. Ele deve ser chamado antes que qualquer outro método para popular o objeto DButils com informações. nome_arq : caminho do arquivo <i>.wld</i> ou <i>.obj</i> no disco. retorno : <i>true</i> ou <i>false</i> 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

Como mencionado durante a aula, as informações do cenário virtual (salas e itens) não fazem parte do programa e por isso ficam armazenadas em arquivos no disco rígido. Esta abordagem permite mudarmos a arquitetura do mundo, adicionar mais salas ou itens ou alterá-los, sem a necessidade de modificar o código-fonte, o que nos obrigaria a recompilar o projeto sempre que uma pequena inclusão tivesse que ser feita.

Um único arquivo de texto no disco pode conter informações de muitas salas ou muitos itens. O padrão apresentado aqui serve apenas como um exemplo para demonstrar e orientar como você pode construir diversos ambientes e incrementar seu 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 estas áreas que fazem o jogo ser original.

Cada sala e item dentro de uma determinada área recebe um número virtual, ou **Vnum**. Estes números são independentes, e isso quer dizer que podemos ter uma sala com o Vnum 3001 e ao mesmo tempo possuir um item também 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 *minimalística*, cada área do mundo virtual pode ser definida por 2 tipos de arquivos:

- Arquivos de mundo: contém as definições das salas e suas ligações umas com as outras. Possui extensão **.wld**;
- Arquivos de itens: definem as armas, armaduras, tesouros e outros itens que são manipuláveis. Possui extensão **.obj**.

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 real de um arquivo que define alguns itens (**ex1.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 também 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**, **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 é conectada. É importante respeitar a seguinte ordem de ligação: sala **norte**, sala **sul**, sala **leste**, sala **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. Para indicar que a sala não possui nenhum item, utilizamos o número 0.

Vejamos um exemplo real de um arquivo que define algumas salas (*ex1.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: 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**:

```
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();
}

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

O jogo deve suportar a seguinte lista de comandos:

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.

A implementação do jogo deve possibilitar que um usuário crie seu personagem e realize as interações com o jogo conforme as seções descritas neste documento.

Para realizar este trabalho, sua implementação deve (**no mínimo**):

- Carregar dados utilizando a classe **DButils** para construir um mundo com 10 salas e 8 itens (todos diferentes);
- Implementar as classes descritas na *seção 2*;
- Garantir que os atributos das classes sejam acessíveis apenas via métodos Gets/Sets;
- Implementar 1 classe abstrata;
- Implementar 1 interface;
- Implementar 1 *thread* adicional para executar o *loop* do jogo;