MAC 110 – Introdução à Computação para o BCC
IME – Primeiro Semestre de 2011

Segundo Exercício-Programa (EP2)

Gerador de *Ringtones*

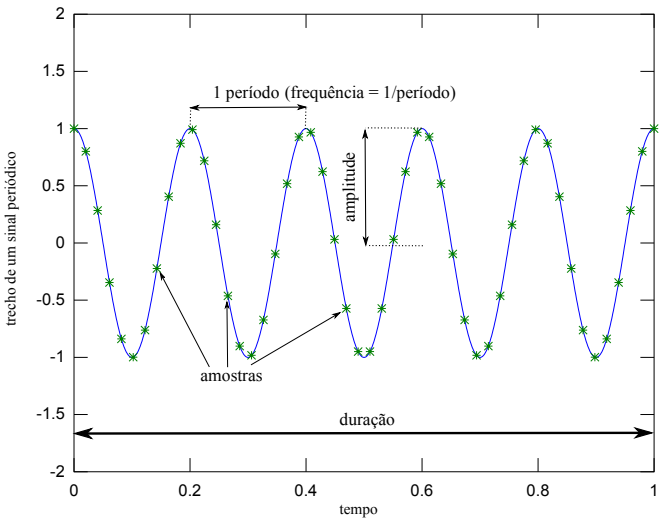
Prazo máximo para entrega: 30/5/2011

versão 1.0 – 17 de maio de 2011

1 Introdução

Neste EP, vamos desenvolver um pequeno sintetizador de áudio digital. Esse sintetizador converte uma representação simbólica de sons em uma representação dos mesmos sons na forma de um arquivo de áudio, entendido pela maioria dos programas multimídia. Assim, ele poderá ser usado, entre outras coisas, como toque de celular (*ringtone*)[1].

O som, como sabemos, é transmitido por uma onda mecânica que se propaga no ar e causa vibrações em nosso aparelho auditivo, que podem ser percebidas como música, fala, ou um barulho qualquer. O *sinal sonoro* indica a variação da pressão do ar em função do tempo. Alguns sons correspondem a oscilações periódicas ou quase-periódicas que dão origem às sensações de *altura musical* (basicamente, a diferenciação entre sons agudos e graves), e o *volume*[2] (ligado à quantidade de energia das vibrações). Os modelos matemáticos para sinais periódicos são definidos por alguns parâmetros: a *forma básica de onda* do sinal sonoro medido durante um *período completo* da oscilação, e corresponde ao formato do gráfico da onda; a *amplitude* da oscilação que está associada à sensação de volume; a *frequência* da oscilação, associada à sensação de altura musical. Estes parâmetros fornecem uma representação compacta para um sinal de duração infinita, embora também possam ser usados para representar trechos com duração finita do mesmo sinal, como na figura a seguir.



[1] Você também poderá escrever uma grande sinfonia se quiser, mas provavelmente você não terá paciência de ouví-la, de tão simples que serão os sons produzidos pelo programa ☺

[2] No dia a dia, os termos altura e volume de som são frequentemente usados como sinônimos, mas a terminologia técnica é diferente.

Um arquivo de áudio, por outro lado, é uma representação explícita do sinal sonoro que permite seu armazenamento utilizando um espaço de memória (digital) finito. A representação descrita na Seção 4 utiliza a técnica de amostragem, que consiste em medir o sinal sonoro regularmente a intervalos de tempo pequenos obtendo *amostras* do sinal (os asteriscos na figura anterior).

O programa a ser desenvolvido neste EP deverá ser capaz de ler uma representação compacta de trechos de sinais periódicos - ou seja, uma sucessão de parâmetros `(duração, altura, volume)`, que representam implicitamente uma melodia, e produzir um arquivo de áudio contendo a representação explícita do sinal sonoro correspondente.

Para isso, vai ser necessário:

1. Aprender a ler e escrever arquivos – isso é só uma pequena variação em relação ao `java.util.Scanner` e `System.out.println()`, que vocês já conhecem, e está explicado na Seção 3.
2. Aprender uma forma de representação de áudio digital; em particular, vamos aprender a escrever arquivos WAVE, que são um estilo padronizado de arquivos de áudio. A teoria está na Seção 4, mas usaremos algumas funções já prontas para simplificar nosso trabalho.
3. Conhecer os detalhes do processo de tradução dos parâmetros da entrada `(duração, altura, volume)` nos sinais sonoros correspondentes, usando o conceito de *osciladores*; este processo será detalhado na Seção 5.

2 Especificação do EP

Você deve escrever um programa em Java que leia da entrada as informações específicas do início do enunciado e gere o arquivo WAVE correspondente a essa entrada.

A entrada contém as seguintes informações, a serem digitadas pelo usuário:

1. nome do arquivo de saída (com extensão `.wav`).
2. oito parâmetros em ponto flutuante $\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \alpha_4, \beta_4$ usados para controlar a síntese de quatro osciladores senoidais (a explicação está na Seção 5).
3. um valor em ponto flutuante correspondente à *duração total do arquivo de saída*, em segundos.
4. uma lista de eventos sonoros, formados por triplas de valores em ponto flutuante:
   - *duração*, em segundos;
   - *altura*, entre 0 e 120, que será traduzida em frequência; e
   - *volume*, entre 0 e 1, que será traduzido em amplitude.

   A leitura desses eventos deve parar quando o arquivo de saída chegar à duração total solicitada. Esse controle será feito através da contagem do número de amostras correspondentes a cada evento, como explicado na Seção 4.

Para o cálculo da saída você deve utilizar os métodos `Math.cos(double x)` e `Math.pow(double a, double b)`.

Você poderá utilizar a classe auxiliar `Wave` disponível junto com o enunciado do programa.

Seu programa deve

1. ler o nome do arquivo de saída (`.wav`) do teclado
2. ler os parâmetros que antecedem a lista de eventos, de acordo com o início do enunciado.

3. abrir o arquivo de saída (você encontrará algumas informações úteis sobre arquivos na Seção 3).
4. calcular o número de amostras total (a partir da duração total do arquivo) e escrever o cabeçalho do arquivo de saída, conforme explicado na Seção 4.
5. ler cada um dos eventos, descrito por três valores, e gerar as listas de amostras correspondentes ao evento, conforme explicado na Seção 5.

Além disso, sua implementação *deve* utilizar alguns métodos padronizados, que serão explicados na Seção 6.

3 Arquivos

Um *arquivo* é um agrupamento de dados, geralmente guardado em disco, identificado por um nome.

Algumas operações básicas que se faz com arquivos são:

Criar Fazer com que um novo arquivo apareça no disco.

Ler Pegar dados do arquivo; é análogo a pegar dados do teclado.

Escrever Colocar dados no arquivo; é análogo a escrever numa janela de comandos.

3.1 Abertura e criação de arquivos em Java

Neste EP, você precisará criar o arquivo de saída que utilizaremos para escrever nosso ringtone.

Para tal, podemos usar a classe File:

```
import java.io.File;
// Instanciando um objeto da classe File que abre/cria um arquivo de nome
//"nome do arquivo"no mesmo diretório do programa.
File saidaWave = new File("nome do arquivo");
```

Se você deseja criar um novo arquivo (mesmo que ele já exista no diretório atual):

```
// Caso o arquivo exista, deleta o arquivo existente.
if (saidaWave.exists()) saidaWave.delete();
```

3.2 Leitura de arquivos

Para esse EP, você deverá ler os dados da entrada através do teclado.

Como curiosidade, mostraremos que também podemos ler um arquivo usando a classe Scanner.

Para lermos do teclado fazemos:

```
Scanner sc = new Scanner(System.in);
```

Se quiséssemos ler de um arquivo, poderíamos fazer:

```
import java.io.File;
File entrada = new File("nome do arquivo");
Scanner sc = new Scanner(entrada);
```

3.3 Escrita em arquivos

Neste EP, você usará a classe Wave, conforme explica a seção 4, para escrever no seu arquivo .wav.

Para escrever uma sequência de bytes no arquivo de saída, os métodos da classe Wave recebem um objeto da classe File com o arquivo de saída e utilizam a classe FileOutputStream do Java:

```
import java.io.FileOutputStream;
// Cria um objeto da classe FileOutputStream para escrever no
// arquivo saidaWave (objeto da classe File).
FileOutputStream escritor = new FileOutputStream (saidaWave);
// Escreve sequenciaDeBytes (vetor de byte) no arquivo.
escritor.write(sequenciaDeBytes);
```

Lembre-se de que você não terá que se preocupar com a escrita dos bytes, pois a classe Wave se encarregará de tudo isso.

3.4 Estrutura dos arquivos

Embora um arquivo seja só uma sucessão de bytes, os dados nele costumam ser organizados de acordo com algum *formato*, que varia de acordo com a natureza da informação contida nele (por exemplo, texto, vídeo, programas, música, etc.). Frequentemente encontramos formatos com a estrutura a seguir:

Cabeçalho contém informações descritivas sobre os dados, ou *meta-dados* (por exemplo, o número de pixels de uma imagem, o número de canais de um arquivo de áudio, ou o nome da linguagem de programação do programa contido no arquivo);

Dados esta é a parte que contém os dados "prá valer" (embora em vários casos dificilmente teríamos condições de interpretar os dados sem o cabeçalho).

Em particular, os arquivos WAVE, explicados na Seção 4, são assim. Normalmente, quando se trabalha com arquivos padronizados, costuma-se usar bibliotecas de classes que cuidam de escrever o cabeçalho e os dados no formato desejado, simplificando enormemente o acesso e manipulação destes arquivos.

4 Representações de áudio

Uma das representações de áudio digital mais comuns é o *Pulse Code Modulation* (PCM), que utiliza a técnica de amostragem e converte cada amostra do sinal em um código binário (inteiro). O número de amostras obtidas para cada segundo do som é chamado de *taxa de amostragem* do sinal e está associado à precisão da representação do sinal sonoro no eixo horizontal (do tempo); quanto maior a taxa de amostragem, melhor a representação do som original e maior a fidelidade obtida na hora de reproduzir o sinal armazenado[3]. O *número de bits* utilizados para armazenar cada amostra está associado à precisão da representação do gráfico no eixo vertical (da variação de pressão do ar). Um arquivo de áudio pode armazenar vários *canais* diferentes contendo sinais sonoros que serão reproduzidos simultaneamente em um sistema com várias caixas acústicas.

Como exemplo, se armazenamos 44100 amostras por segundo, com 16 bits por amostra, em um arquivo de áudio estéreo, então cada segundo de som ocupa $44100 * 16 * 2$ bits $= 176400$ bytes deste arquivo. Estes valores correspondem à qualidade de CD[4] e são adotados como padrão neste EP.

Para a programação do EP, estarão disponíveis dois métodos explicadas a seguir que tornarão seu trabalho muito mais fácil. O parâmetro da representação de áudio que terá mais importância

[3] A taxa de amostragem R define não apenas a qualidade da representação da onda sonora como também a máxima frequência sonora representável, que é conhecida como frequência de Nyquist e vale $R/2$ Hz.

[4] Outro padrão comum é 8000 amostras de 8 bits por segundo em um único canal, que é aproximadamente a qualidade sonora do sinal transmitido por telefone.

no seu código é a taxa de amostragem $R = 44100Hz$, usada para converter todas as durações (expressas em segundos) para a correspondente quantidade de amostras. O arquivo de áudio terá uma quantidade total de amostras igual a $R * \textit{(duração total do arquivo de saída)}$, enquanto cada evento terá $44100 * \textit{(duração do evento)}$ amostras. Na prática, você deve calcular o número de amostras correspondentes usando o *arredondamento* destas expressões. Observe que a terminação do laço principal depende da comparação do número de amostras geradas com o número de amostras total do arquivo, valor este que é necessário para a geração do cabeçalho WAVE.

Arquivos wave

Aqui veremos uma descrição simplificada dos arquivos WAVE. Para informações mais detalhadas, consulte este site, ou a Wikipedia. O importante é que estamos fornecendo uma pequena biblioteca que cuida dos vários detalhes (alguns desses detalhes estão explicados nos comentários, no código dos métodos), assim você não precisa se preocupar em aprender esses pormenores para fazer o EP.

Como já mencionamos antes, o arquivo WAVE possui um cabeçalho, informando o formato das amostras, número de canais, taxa de amostragem e o número total de amostras. Em seguida, aparecem as várias amostras. Nossos arquivos serão estéreo, assim as amostras aparecem na ordem

$$\text{amostra}_1\text{-canal}_1,\ \text{amostra}_1\text{-canal}_2,\ \text{amostra}_2\text{-canal}_1,\ \text{amostra}_2\text{-canal}_2,\ \ldots$$

A convenção para arquivos estéreo é canal_1=esquerdo e canal_2=direito. Nossos arquivos estéreo poderão possuir duas cópias distintas do sinal sonoro, de acordo com o uso dos osciladores descrito na seção seguinte.

Os arquivos WAVE tem, por convenção, nome terminado com a extensão `.wav`. Após abrir o arquivo de saída, é preciso escrever o cabeçalho antes de começar a escrever as amostras. Para isso, use o método

```
escreveCabecalhoWave (File saidaWave, int numeroDeAmostras);
```

onde o tamanho do vetor passado como parâmetro é calculado a partir da duração total da entrada. Os vários outros ingredientes do cabeçalho serão fixos neste EP, e estão embutidos no código do método.

Em seguida, são escritas as amostras, que no arquivo WAVE são inteiros de 16 bits. O modelo que explicamos abaixo produz amostras em ponto flutuante no intervalo $[-1, 1]$; para simplificar esta conversão, providenciamos o método abaixo que recebe uma amostra em ponto flutuante e faz todo o tratamento necessário para escrever os valores no formato exigido pelo arquivo WAVE:

```
escreveAmostra(File saidaWave, float amostra);
```

Quem tiver curiosidade a respeito desta conversão poderá encontrar comentários no código do método `escreveAmostra()`.

5 Síntese sonora

A descrição dos eventos que usamos na entrada é bem simples, pois corresponde a uma representação abstrata (do modelo matemático do som), e não do sinal sonoro propriamente dito. Para produzir os sinais sonoros, usaremos osciladores senoidais, explicados a seguir. Além disso, usaremos também um mecanismo de suavização, a fim de evitar "barulhos" devido à mudança abrupta nos parâmetros dos osciladores na transição de um evento para o outro; isso será abordado na Seção 5.2.

5.1 Osciladores senoidais

Um oscilador senoidal (discreto) é a função descrita pelos parâmetros *amplitude (A)* e *frequência (F)* que corresponde à expressão $f(n) = A\cos(2\pi F\frac{n}{R})$, onde R é a taxa de amostragem e $n = 0, 1, \ldots$ são os índices das amostras. Utilizaremos um banco de osciladores desse tipo para sintetizar os eventos

sonoros descritos na entrada do programa. Por conveniência, usaremos osciladores que produzem valores em ponto flutuante no intervalo $[-1, \ldots, +1]$.

Lembre-se que cada evento sonoro é descrito por uma *duração (D)* (em segundos), uma *altura (B)* e um *volume (V)*. A duração determinará o número de amostras utilizadas pelo oscilador neste evento, enquanto os parâmetros B e V determinarão a frequência e a amplitude de referência do evento através das fórmulas[5]

$$F(B) = 27.5 * 2^{\frac{B}{12}} \qquad \text{e} \qquad A(V) = 2^{(10V-10)}.$$

Nosso EP utilizará um banco formado por quatro osciladores similares, porém com ajustes diferentes de amplitude e de altura. Especificamente, a *amplitude* do i-ésimo oscilador deverá ser multiplicada por α_i, enquanto a *altura* deverá ser incrementada em β_i, ou seja, a saída do i-ésimo oscilador corresponderá à expressão

$$f_i(n) = \alpha_i A \cos\left(2\pi F(B+\beta_i)\frac{n}{R}\right).$$

Como exemplo, se os parâmetros forem $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0,25$ e $\beta_1 = 0$, $\beta_2 = 12$, $\beta_3 = 19$ e $\beta_4 = 24$, e estivermos produzindo um evento $(D; B; V) = (1; 48; 0,9)$, então os quatro osciladores corresponderão respectivamente às expressões

$$f_1(n) = 0.125\cos\left(2\pi 440\frac{n}{44100}\right), \quad f_2(n) = 0.125\cos\left(2\pi 880\frac{n}{44100}\right), \quad f_3(n) = 0.125\cos\left(2\pi 1318.5\frac{n}{44100}\right), \quad f_4(n) = 0.125\cos\left(2\pi 1760\frac{n}{44100}\right),$$

calculadas para $n = 0, 1, \ldots, 44099$.

Em princípio, a saída dos quatro osciladores será somada para produzir os valores que correspondem às amostras dos canais esquerdo e direito. No exemplo acima, o valor $\sum_{i=1}^{4} f_i(n)$ corresponderá à amostra de índice n, que neste caso será igual para o canal esquerdo e para o canal direito.

Para aumentar a diversidade de resultados, permitiremos aos dois últimos osciladores comportarem-se de maneira diferente quando $\beta_i < 0$. Especificamente, se $\beta_3 < 0$, então o terceiro oscilador produzirá os valores

$$\Delta(n) = \frac{(2^{\frac{\alpha_3}{12}} - 1)}{F(\beta_3)} \cos\left(2\pi F(\beta_3)\frac{n}{R}\right); \tag{1}$$

Esse sinal será usado para modificar o comportamento dos outros osciladores, que passarão a computar expressões do tipo

$$f_i(n) = \alpha_i A \cos\left(2\pi F(B+\beta_i)\frac{n}{R} + F(B+\beta_i)\Delta(n)\right). \tag{2}$$

Já no caso em que $\beta_4 < 0$, então o quarto oscilador produzirá os valores

$$\Gamma(n) = \frac{\alpha_4}{2} * \cos\left(2\pi F(\beta_4)\frac{n}{R}\right), \tag{3}$$

valores estes que serão usados para modificar a saída dos canais esquerdo e direito, que deverão ser multiplicadas respectivamente pelos valores

$$K_e(n) = 1 - \frac{\alpha_4}{2} + \Gamma(n) \qquad\qquad K_d(n) = 1 - \frac{\alpha_4}{2} - \Gamma(n)$$

Observe que estes comportamentos modificados dos dois últimos osciladores se sobrepõem aos comportamentos originais. Ou seja, sempre que $\beta_3 < 0$, o terceiro oscilador será usado *exclusivamente* para modificar o comportamento dos demais osciladores, que incorporarão $\Delta(n)$ às suas expressões. Analogamente, sempre que $\beta_4 < 0$, o quarto oscilador será usado *exclusivamente* para modificar os valores enviados para os canais esquerdo e direito de acordo com as expressões $K_e(n)$ e $K_e(n)$. Se valerem simultaneamente $\beta_3 < 0$ e $\beta_4 < 0$, então na prática apenas as amostras dos dois primeiros osciladores (modificados) serão somadas para produzir as saída deste banco de osciladores.

[5]Note que altura e volume, percebidos pelo ouvido, são aproximadamente os logaritmos da frequência e amplitude.

5.2 Envoltórias dinâmicas

Talvez você já tenha tido o desprazer de ouvir uma pessoa conectando (ou desconectando) um cabo de som em um alto-falante com o ajuste de volume no máximo. O ruído produzido se deve a descontinuidade do sinal da entrada da caixa acústica, que passa subitamente de zero ao valor do sinal transmitido pelo cabo. Algo muito parecido com isso aconteceria na passagem de um evento para o outro se emendássemos de qualquer jeito os valores produzidos pelo banco de osciladores. A solução, como no caso dos alto-falantes, é iniciar o evento com amplitude zero, gradualmente atingir o valor máximo de amplitude pretendida, e finalizar o evento levando a amplitude gradualmente até zero. Estas rampas de controle de amplitude são conhecidas como *ataque* e *decaimento* do evento sonoro.

Neste EP, vamos usar ataques e decaimentos com duração fixada em $0,1$ segundos, ou 4410 amostras. Em princípio, devemos multiplicar a saída do banco de osciladores por uma função linear por trechos chamada envoltória dinâmica, que cresce linearmente de 0 a 1 em 4410 amostras, permanece fixa em 1 no segmento intermediário, com duração (em amostras) igual a *(número de amostras do evento)-8820*, e decresce linearmente de 1 a 0 durante as 4410 amostras finais.

Aplicar este princípio só não será possível quando a duração total do evento for menor do que 8820 amostras. Neste caso, ajustaremos as rampas para que a envoltória vá de 0 a 1 em *(duração do evento)/2* amostras (use truncamento para obter um valor inteiro) e volte a 0 linearmente durante as amostras restantes (em relação à duração do evento).

6 Métodos auxiliares *obrigatórios*

Para auxiliar a confecção do EP e também para auxiliar a correção, será exigida a implementação de dois métodos com assinaturas padronizadas, que corresponderão aos osciladores da Seção 5.1 e à envoltória dinâmica da Seção 5.2:

Osciladores serão implementados pelo método

```
DeltaGamma oscilador(double altura, double volume,
                     double alfa, double beta, int indiceAmostra);
```

Essa função deverá computar a expressão do oscilador modificado (equação 2), caso `beta`≥ 0. Quando `beta`< 0, essa função deve computar as expressões correspondentes a `Delta` (equação 1) e `Gama` (equação 3), devolvendo esses dois valores através de um objeto do tipo `DeltaGamma` a ser definido por você[6].

Envoltória será calculada pelo método

```
double envoltoria(int indiceAmostra, int numeroDeAmostrasDoEvento);
```

Esse método devolverá o valor correspondente à função envoltória descrita na Seção 5.2 para a amostra de índice `indiceAmostra` de um evento com duração dada por `numeroDeAmostrasDoEvento`, fazendo os ajustes necessários caso a duração do evento seja menor do que $0,2$ segundos (8820 amostras).

7 Organização do código

Pense bem em como organizar o seu código. Agrupe métodos em várias classes de acordo com a natureza da computação que eles realizem. Você deve escolher bons nomes para essas classes e para os métodos que criar (além, é lógico, de usar os nomes já definidos neste enunciado como obrigatórios).

Você deve evitar linhas longas demais (por exemplo, maior do que 100 caracteres), métodos longos demais (por exemplo, com mais de 15 ou 20 linhas) e classes grandes demais (por exemplo, com mais de 8 ou 10 métodos).

[6] Pense em como evitar a criação de um novo objeto do tipo `DeltaGamma` a cada chamada ao método `oscilador()`, isso não seria muito eficiente.

8 Executável e conversor de wave para texto

Durante a depuração, você naturalmente vai querer olhar o conteúdo do arquivo WAVE gerado pelo seu programa, e compará-lo com outros arquivos WAVE. Como esses não são arquivos de texto, mas sim arquivos binários, eles não podem ser visualizados de maneira adequada por processadores de texto usuais. Para ajudar, disponibilizamos um programa, que chamamos de `wav2txt`, para exibir o conteúdo de arquivos WAVE em formato de texto; ele exibe as principais informações do cabeçalho e também os valores das amostras usando a notação textual. Além deste, o programa `wavdiff` permite comparar dois arquivos WAVE, exibindo o maior valor de diferença entre amostras dos dois arquivos caso não sejam idênticos.

Os programas `wav2txt` e `wavdiff` estão disponíveis em http://www.ime.usp.br/~am/110/ep2/wav2txt. Uma versão já finalizada desse mesmo EP implementado na linguagem C encontra-se disponível em formato executável em http://www.ime.usp.br/~am/110/ep2/executaveis. Esses programas estão disponíveis nos sabores Windows e Linux. Como tarefa opcional, se alguém quiser reimplementar os programas `wav2txt` e `wavdiff` na linguagem Java, por favor, envie-o para a monitora que nós iremos disponibilizá-lo na página do EP.

Em http://www.ime.usp.br/~am/110/ep2/entradas estão disponíveis vários arquivos de entrada para teste, que simulam a entrada pelo teclado. Se o seu programa se chama `Sintetizador`, basta executar o comando a seguir numa janela de comandos, como o **Terminal** no Linux ou o **cmd.exe** (também conhecido como Prompt de comando) no Windows:

```
java Sintetizador < entrada1.txt
```

e o seu programa lerá o arquivo `entrada1.txt` como se fosse uma entrada do teclado. Note que tanto o arquivo `Sintetizador.class` quanto o arquivo de entrada precisam estar no diretório corrente para a linha acima funcionar. Algumas dessas entradas são pequenas e, por isso, convenientes para a fase de depuração do seu programa (quando você poderá inspecionar o conteúdo usando os programas `wav2txt` ou comparar com a saída do executável usando `wavdiff`). Outras são mais longas e, por isso, interessantes de ouvir com algum software de áudio. Um software livre para manipulação de áudio muito bom que eu uso com frequência e recomendo é o Audacity. Ele inclusive converte para MP3 se vocês quiserem. Você pode criar outros arquivos de entrada e disponibilizá-los no fórum de discussão da disciplina.

É muito importante se certificar que a saída produzida pelo seu programa está correta antes de ouvir os arquivos produzidos, especialmente se você estiver usando fones-de-ouvido. Uma saída incorreta, além de desagradável de ouvir, poderia mesmo machucar seus ouvidos ou danificar seu equipamento. Seja prudente!

8.1 Sobre a avaliação:

- É sua responsabilidade manter o código do seu EP em sigilo, ou seja, apenas você e seu par devem ter acesso ao código.
- No caso de exercícios feitos em dupla, a mesma nota da correção será atribuída aos dois alunos do grupo.
- **Não serão toleradas cópias!** Exercícios copiados (com ou sem eventuais disfarces) levarão à reprovação da disciplina e o encaminhamento do caso (tanto dos alunos copiados quanto os alunos que executaram a fraude) para a Comissão de Graduação do aluno.
- Exercícios atrasados não serão aceitos;
- Exercícios com erros de sintaxe (ou seja, erros de compilação) receberão nota zero.

- É muito importante que seu programa tenha bons comentários sempre que necessário e esteja bem indentado, ou seja, digitado de maneira a ressaltar a estrutura de subordinação dos comandos do programa (conforme visto em aula). A qualidade do seu trabalho sob esse ponto de vista influenciará sua nota!

- Uma componente importante da avaliação será a clareza do seu programa. Em particular, é fundamental que você escolha bons nomes para suas variáveis, parâmetros, atributos, métodos e classes e que esses nomes descrevam claramente a sua intenção.

- As informações impressas pelo seu programa na tela devem aparecer da forma mais clara possível.

- Uma regra básica é a seguinte: do ponto de vista do monitor responsável pela correção dos trabalhos, quanto mais convenientemente apresentado estiver o seu programa, melhor avaliado será seu trabalho.

9 Informações sobre entrega do EP

- O prazo de entrega é o dia 30/5/2011;

- Entregar apenas os arquivos .java comprimidos, todos dentro de um arquivo comprimido cujo nome deve ser SeusNomesReais.zip.

- No início do arquivo, acrescente um cabeçalho bem informativo, como o seguinte:

```
/*****************************************************************/
/**   MAC 110 - Introdução à Computação                         **/
/**   IME-USP - Primeiro Semestre de 2009                       **/
/**   <turma> - <nome do professor>                             **/
/**                                                             **/
/**   Segundo Exercício-Programa  --  Sintetizador Musical      **/
/**   Arquivo:Algo.java                                         **/
/**                                                             **/
/**   <nome do(a) aluno(a)>              <número USP>           **/
/**   <nome do(a) aluno(a)>              <número USP>           **/
/**                                                             **/
/*****************************************************************/
```

  Não é obrigatório que o cabeçalho seja idêntico a esse, apenas que contenha pelo menos as mesmas informações.

- Para a entrega, utilize o Paca. Você pode entregar várias versões de um mesmo EP até o prazo, mas somente a última será armazenada pelo sistema.

- Não serão aceitas submissões por email ou atrasadas. Não deixe para a última hora, pois o sistema pode ficar congestionado e você poderá não conseguir enviar.

- Guarde uma cópia do seu EP pelo menos até o fim do semestre!