Orientação a Objeto - Interfaces e Polimorfismo

Objetivos da seção

- Apresentar o conceito de Tipo Abstrato de Dado ("interface", em Java)
- Apresentar o polimorfismo e sua utilidade em desacoplar classes
- Apresentar a implementação de polimorfismo através de ligação dinâmica (*dynamic* ou *late binding*)

Um Sistema Simples de Manipulação de Correio Eletrônico: Um Exercíco na Modularização de um Programa

- Não há nada com respeito a interfaces e polimorfismo nesta seção
  - Ela serve de contexto para as seções posteriores que tratarão do assunto em pauta
  - O código visto aqui também mostra como usar classes, objetos e métodos para modularizar um programa um pouco maior
- Queremos implementar um programa simples de manipulação de correio eletrônico com interface a caractere (não GUI - Graphical User Interface)
  - Conceito de uma *mensagem* de correio
  - Conceito de uma *caixa postal* onde são armazenadas mensagens recebidas
  - Operações típicas de um sistema de correio
    - Ler mensagens
    - Remover mensagens
    - Enviar mensagens

Sessão típica de uso do sistema que queremos implementar

- As partes em azul foram digitadas pelo usuário

```
C:\...\src>java -classpath .;package1\p1.jar Correio1 jacques
--------------------------------------------------------------------------
Nao ha mensagem.
--------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? en
Para quem? ana
Assunto? te amo, minha linda!
Conteudo da mensagem? (. para terminar)
Ana,
Voce sabia que eh o eterno amor de minha vida?
Beijos do fofo!

fofo
.
--------------------------------------------------------------------------
Nao ha mensagem.
--------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? en
Para quem? ana
Assunto? amor, novamente
Conteudo da mensagem? (. para terminar)
Nao tendo recebido resposta a minha mensagem anterior,
eu gostaria de reafirmar tudo que ai foi dito, ta?

o fofo
.
--------------------------------------------------------------------------
Nao ha mensagem.
--------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? q
```

```
Salvar CaixaPostal? s

C:\...\src>java -classpath .;packagep1\p1.jar Correio1 ana
-------------------------------------------------------------------------
  Num Remetente    Data             Assunto
>   1 jacques      29/06/2001 15:52 te amo, minha linda!
    2 jacques      29/06/2001 15:53 amor, novamente
-------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? exi
De: jacques
Data: 29/06/2001 15:52
Assunto: te amo, minha linda!
Ana,
Voce sabia que eh o eterno amor de minha vida?
Beijos do fofo!

fofo

-------------------------------------------------------------------------
  Num Remetente    Data             Assunto
>   1 jacques      29/06/2001 15:52 te amo, minha linda!
    2 jacques      29/06/2001 15:53 amor, novamente
-------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? exc
-------------------------------------------------------------------------
  Num Remetente    Data             Assunto
>X  1 jacques      29/06/2001 15:52 te amo, minha linda!
    2 jacques      29/06/2001 15:53 amor, novamente
-------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? +
-------------------------------------------------------------------------
  Num Remetente    Data             Assunto
 X  1 jacques      29/06/2001 15:52 te amo, minha linda!
>   2 jacques      29/06/2001 15:53 amor, novamente
-------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? exi
De: jacques
Data: 29/06/2001 15:53
Assunto: amor, novamente
Nao tendo recebido resposta a minha mensagem anterior,
eu gostaria de reafirmar tudo que ai foi dito, ta?

o fofo

-------------------------------------------------------------------------
  Num Remetente    Data             Assunto
 X  1 jacques      29/06/2001 15:52 te amo, minha linda!
>   2 jacques      29/06/2001 15:53 amor, novamente
-------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? exc
-------------------------------------------------------------------------
  Num Remetente    Data             Assunto
 X  1 jacques      29/06/2001 15:52 te amo, minha linda!
>X  2 jacques      29/06/2001 15:53 amor, novamente
-------------------------------------------------------------------------
exibir, enviar, excluir, +, -, quit? q
Salvar CaixaPostal? s
```

O projeto básico do sistema

- Queremos portanto escrever o programa Correio1
- Que classes já estão disponíveis para fazer isso?
  - Examine a documentação das classes CaixaPostal e Mensagem agora
  - Deve ficar óbvio que a única coisa que falta, realmente, é escrever a interface para o usuário para o sisteminha de correio eletrônico
- Já que sabemos escrever classes, vamos escrever a interface para o usuário como uma classe CorreioIU1
  - **Examine o javadoc dessa classe** CorreioIU1 para ver o que devemos fazer. Não estamos escrevendo código até agora, só planejando as coisas.

O programa principal

- Agora, supondo que a classe CorreioIU1 existisse, como ela seria usada?
- O programa principal segue abaixo e está em Correio1.java

```
/*
 * Correio Eletrônico simples. Programa principal.
 */
import p1.aplic.correio.*;

public class Correio1 {
    public static void main(String[] args) {
        if(args.length != 1) {
            System.err.println("Sintaxe: java Correio1 nome");
            System.exit(1);
        }
        CorreioIU1 ciu = new CorreioIU1(args[0]);
        ciu.interfaceComUsuário();
    }
}
```

- Como se vê, o trabalho é realmente feito pelo método interfaceComUsuário() da classe CorreioIU1

A classe CorreioIU1

- Já que essa classe não é tão simples, vamos primeiro ver o pseudo-código do que é necessário fazer (ver CorreioIU1.pseudo)

```
// interface com o usuário para um sisteminha de correio eletrônico
public class CorreioIU1 {
  private CaixaPostal caixa;

  public CorreioIU1(String titular) {
    // construtor para manipular a caixa postal de "titular"
  }

  public void interfaceComUsuário() {
    loop de interpretação de comandos {
      mostrar um resumo da caixa postal para o usuário
      ler o comando
      se comando for "exibir"
        exibir a mensagem
      se comando for "enviar"
        obtem dados (destinatário, assunto, texto da mensagem)
        criar uma nova mensagem com esses dados
        armazenar a mensagem na caixa postal do destinatário
      se comando for "excluir"
        mandar excluir a mensagem corrente
      se comando for "+"
        avançar para a próxima mensagem
```

```
        se comando for "-"
          recuar para a mensagem anterior
        se comando for "quit"
          salvar a caixa postal se o usuário quiser e cair fora
      }
    }
  }
```

- A estrutura do código já está pronta
- Que tal testarmos essa estrutura sem adicionar detalhes complicadores agora e tentar deixar o programa funcionando (pelo menos no que diz respeito ao loop de comandos)?
  - Dessa forma, poderemos resolver separadamente uma pequena parte do problema (a estrutura global do programa) sem ter que resolver tudo de uma vez, o que seria mais complicado
  - A forma de fazer isso é de introduzir código artificial para "guardar lugar" por enquanto
  - **O resultado está em CorreioIUTemp1.java**

```
// interface com o usuário para um sisteminha de correio eletrônico
// a classe ainda não faz parte do package p1.aplic.correio
import p1.aplic.correio.*;
import p1.io.*;

public class CorreioIUTemp1 {
  private CaixaPostal caixa;

  public CorreioIUTemp1(String titular) {
    caixa = new CaixaPostal(titular);
  }

  public void interfaceComUsuário() {
    while(true) {
      System.out.println( "Resumo da caixa postal para o usuario");
      String cmd = Entrada.in.lerLinha("exibir, enviar, excluir, +, -, quit? ")
      if(cmd.startsWith("exi")) {
        System.out.println("Exibicao de mensagem");
      } else if(cmd.startsWith("en")) {
        // obtem dados (destinatário, assunto, texto da mensagem)
        // criar uma nova mensagem com esses dados
        // armazenar a mensagem na caixa postal do destinatário
        System.out.println("Envio de nova mensagem");
      } else if(cmd.startsWith("exc")) {
        System.out.println("Exclusao de mensagem");
      } else if(cmd.startsWith("+")) {
        System.out.println("Avanca para proxima mensagem");
      } else if(cmd.startsWith("-")) {
        System.out.println("Recua para mensagem anterior");
      } else if(cmd.startsWith("q")) {
        cmd = Entrada.in.lerLinha("Salvar CaixaPostal? ");
        if(cmd.startsWith("s")) {
          System.out.println("Salvamento da caixa postal");
        }
        break;
      } else {
        System.out.println("Comando <" + cmd + "> desconhecido");
      }
    }
  }

  // um mainzinho para testar
```

```
  public static void main(String[] args) {
    if(args.length != 1) {
      System.err.println("Sintaxe: java CorreioIUTemp1 nome");
        System.exit(1);
    }
    CorreioIUTemp1 ciu = new CorreioIUTemp1(args[0]);
    ciu.interfaceComUsuário();
  }
}
```

- Execute e teste o programa (não esquecendo do argumento titular)

```
C:\...\src>java -classpath .;packagep1\p1.jar CorreioIUTemp1 jacques
Resumo da caixa postal para o usuario
exibir, enviar, excluir, +, -, quit? exi
Exibicao de mensagem
Resumo da caixa postal para o usuario
exibir, enviar, excluir, +, -, quit? en
Envio de nova mensagem
Resumo da caixa postal para o usuario
exibir, enviar, excluir, +, -, quit? exc
Exclusao de mensagem
Resumo da caixa postal para o usuario
exibir, enviar, excluir, +, -, quit? +
Avanca para proxima mensagem
Resumo da caixa postal para o usuario
exibir, enviar, excluir, +, -, quit? -
Recua para mensagem anterior
Resumo da caixa postal para o usuario
exibir, enviar, excluir, +, -, quit? alo
Comando <alo> desconhecido
exibir, enviar, excluir, +, -, quit? q
Salvar CaixaPostal? s
Salvamento da caixa postal
```

- A beleza do que fizemos é que podemos essencialmente esquecer o que já está funcionando e continuar "enchendo os buracos" aos poucos até que o programa inteiro esteja pronto
  - **Isso nos permite manter o foco em uma coisa de cada vez**
  - Releia a frase anterior: é uma das lições mais importantes da programação
- Agora, vamos rechear o programa um pouco
  - Vamos acertar os comandos simples, isto é, aqueles que podemos implementar chamando métodos das classes Mensagem e CaixaPostal
  - Como esses objetos já estão prontos, temos pouco trabalho a fazer
  - Portanto, vamos agora implementar os comandos "exibir", "excluir", "+", "-" e "quit"
  - **O resultado pode ser visto em CorreioIUTemp2.java**

```
// interface com o usuário para um sisteminha de correio eletrônico
// a classe ainda não faz parte do package p1.aplic.correio
import p1.aplic.correio.*;
import p1.io.*;

public class CorreioIUTemp2 {
  private CaixaPostal caixa;

  public CorreioIUTemp2(String titular) {
    caixa = new CaixaPostal(titular);
  }
```

```
  public void interfaceComUsuário() {
    while(true) {
      System.out.println( "Resumo da caixa postal para o usuario");
      String cmd = Entrada.in.lerLinha("exibir, enviar, excluir, +, -, quit? ")
      if(cmd.startsWith("exi")) {
        Mensagem m = caixa.mensagemCorrente();
        if(m != null) {
          m.exibir();
        }
      } else if(cmd.startsWith("en")) {
        // obtem dados (destinatário, assunto, texto da mensagem)
        // criar uma nova mensagem com esses dados
        // armazenar a mensagem na caixa postal do destinatário
        System.out.println("Envio de nova mensagem");
      } else if(cmd.startsWith("exc")) {
        caixa.excluir();
      } else if(cmd.startsWith("+")) {
        caixa.avançar();
      } else if(cmd.startsWith("-")) {
        caixa.recuar();
      } else if(cmd.startsWith("q")) {
        cmd = Entrada.in.lerLinha("Salvar CaixaPostal? ");
        if(cmd.startsWith("s")) {
          caixa.salvar();
        }
        break;
      } else {
        System.out.println("Comando <" + cmd + "> desconhecido");
      }
    }
  }

  // um mainzinho para testar
  public static void main(String[] args) {
    if(args.length != 1) {
      System.err.println("Sintaxe: java CorreioIUTemp2 nome");
      System.exit(1);
    }
    CorreioIUTemp2 ciu = new CorreioIUTemp2(args[0]);
    ciu.interfaceComUsuário();
  }
}
```

- Claro que, ao testar o programa, não poderemos fazer muita coisa já que não temos mensagens nas caixas postais
  - Uma alternativa é de copiar uma caixa postal pronta (ana.correio) no diretório corrente e chamar CorreioIUTemp2
  - Observe que nosso programa está todo pronto, com exceção do envio de mensagens e da exibição do resumo da caixa postal
- Vamos agora implementar a exibicão da caixa postal
  - Já que ...
    - ... essa tarefa não é muito simples (no fim, vai precisar de 10 a 20 linhas de código)
    - ... e já que esta operação representa algo coeso (uma operação bem definida em torno de dados bem definidos)
    - ... e já que representa algo que poderia ser reproveitado em outro momento
  - ... então vamos implementá-la como método em vez de colocar todo o código dentro do loop de comandos
  - Chamemos o método de mostraResumoCaixaPostal()

- Observe que este método não é público pois não interessa aos clientes da classe
  - O método tem interesse apenas *interno*, isto o método interessa aos métodos da própria classe CorreioIUTempN
  - **Uma primeira versão da solução aparece em CorreioIUTemp3.java**

```
// interface com o usuário para um sisteminha de correio eletrônico
// a classe ainda não faz parte do package p1.aplic.correio
import p1.aplic.correio.*;
import p1.io.*;
import java.util.*;

public class CorreioIUTemp3 {
  private CaixaPostal caixa;

  public CorreioIUTemp3(String titular) {
    caixa = new CaixaPostal(titular);
  }

  public void interfaceComUsuário() {
    while(true) {
      mostraResumoCaixaPostal(caixa);
      String cmd = Entrada.in.lerLinha("exibir, enviar, excluir, +, -, quit? ")
      if(cmd.startsWith("exi")) {
        Mensagem m = caixa.mensagemCorrente();
        if(m != null) {
          m.exibir();
        }
      } else if(cmd.startsWith("en")) {
        // obtem dados (destinatário, assunto, texto da mensagem)
        // criar uma nova mensagem com esses dados
        // armazenar a mensagem na caixa postal do destinatário
        System.out.println("Envio de nova mensagem");
      } else if(cmd.startsWith("exc")) {
        caixa.excluir();
      } else if(cmd.startsWith("+")) {
        caixa.avançar();
      } else if(cmd.startsWith("-")) {
        caixa.recuar();
      } else if(cmd.startsWith("q")) {
        cmd = Entrada.in.lerLinha("Salvar CaixaPostal? ");
        if(cmd.startsWith("s")) {
          caixa.salvar();
        }
        break;
      } else {
        System.out.println("Comando <" + cmd + "> desconhecido");
      }
    }
  }

  // Observe que este método *não* é public
  void mostraResumoCaixaPostal(CaixaPostal caixa) {
   final String separador = "-------------------------------------------------
   Iterator it = caixa.iterator();
    if(!it.hasNext()) {
      System.out.println(separador);
      System.out.println("Nao ha mensagem.");
      System.out.println(separador);
      return;
    }
```

```
      int numMensagem = 1;
      System.out.println(separador);
      while(it.hasNext()) {
        Mensagem m = (Mensagem)it.next();
        String cursor = " ";
        if(m == caixa.mensagemCorrente()) {
          cursor = ">";
        }
        System.out.println(cursor + numMensagem +
                           " Rem: " + m.getRemetente() +
                           " Data: " + m.getDataEnvio().DDMMAAAAHHMM() +
                           " Ass: " + m.getAssunto());
        numMensagem++;
      }
      System.out.println(separador);
    }

    // um mainzinho para testar
    public static void main(String[] args) {
      if(args.length != 1) {
        System.err.println("Sintaxe: java CorreioIUTemp3 nome");
        System.exit(1);
      }
      CorreioIUTemp3 ciu = new CorreioIUTemp3(args[0]);
      ciu.interfaceComUsuário();
    }
  }
```

- Execute o programa para ver o resultado (use uma caixa postal já pronta)
  - A versão do package (CorreioIU1.java) tem uma versão um pouco mais bonita da saída (usando a classe p1.util.Formata) mas não vale a pena perder tempo com isso
  - Basta avisar que a saída pode ter seu visual melhorado mexendo *apenas* com o método mostraResumoCaixaPostal()
- Agora, queremos enviar uma mensagem
  - **Observe como fazer em CorreioIUTemp4.java** e execute o programa
  - Observe que usamos 3 novos métodos (não públicos). A criação de métodos úteis dessa forma chama-se "modularizar usando métodos" e é semelhante a "modularizar usando classes e objetos"

```
// interface com o usuário para um sisteminha de correio eletrônico
// a classe ainda não faz parte do package p1.aplic.correio
import p1.aplic.correio.*;
import p1.io.*;
import java.util.*;

public class CorreioIUTemp4 {
  private CaixaPostal caixa;

  public CorreioIUTemp4(String titular) {
    caixa = new CaixaPostal(titular);
  }

  public void interfaceComUsuário() {
    while(true) {
      mostraResumoCaixaPostal(caixa);
      String cmd = Entrada.in.lerLinha("exibir, enviar, excluir, +, -, quit? ")
      if(cmd.startsWith("exi")) {
        Mensagem m = caixa.mensagemCorrente();
        if(m != null) {
```

```
          m.exibir();
        }
      } else if(cmd.startsWith("en")) {
        String destinatário = obtemUmaLinha("Para quem? ");
        String assunto = obtemUmaLinha("Assunto? ");
        String conteúdo = obtemVáriasLinhas("Conteudo da mensagem? (. para term
        enviarMensagem(caixa.getTitular(), destinatário, assunto, conteúdo);
      } else if(cmd.startsWith("exc")) {
        caixa.excluir();
      } else if(cmd.startsWith("+")) {
        caixa.avançar();
      } else if(cmd.startsWith("-")) {
        caixa.recuar();
      } else if(cmd.startsWith("q")) {
        cmd = Entrada.in.lerLinha("Salvar CaixaPostal? ");
        if(cmd.startsWith("s")) {
          caixa.salvar();
        }
        break;
      } else {
        System.out.println("Comando <" + cmd + "> desconhecido");
      }
    }
  }

  // Observe que este método *não* é public
  void mostraResumoCaixaPostal(CaixaPostal caixa) {
    Iterator it = caixa.iterator();
    if(!it.hasNext()) {
      System.out.println("Nao ha mensagem.");
      return;
    }
    int numMensagem = 1;
    while(it.hasNext()) {
      Mensagem m = (Mensagem)it.next();
      String cursor = " ";
      if(m == caixa.mensagemCorrente()) {
        cursor = ">";
      }
      System.out.println(cursor + numMensagem +
                         " Rem: " + m.getRemetente() +
                         " Data: " + m.getDataEnvio().DDMMAAAAHHMM() +
                         " Ass: " + m.getAssunto());
      numMensagem++;
    }
  }

  // Observe que este método *não* é public
  String obtemUmaLinha(String prompt) {
    String linha;
    while((linha = Entrada.in.lerLinha(prompt)).equals("")) {
      System.out.println("Favor fornecer alguma informacao");
    }
    return linha;
  }

  /** Método auxiliar para obter várias linhas da entrada padrão.
   * O final da entrada é indicado digitando "." sozinho no início
   * de uma linha.
   * @param prompt O prompt a exibir ao usuário antes de ler a informação.
```

```
     * @return As linhas lida, como string único. As linhas são separadas
     * por um caractere de separação de linha apropriado.
     */
    // Observe que este método *não* é public
    String obtemVáriasLinhas(String prompt) {
      String resposta = "";
      String separador = System.getProperty("line.separator");
      String linha;
      System.out.println(prompt);
      while(!(linha = Entrada.in.lerLinha("")).equals(".")) {
        resposta += linha + separador;
      }
      return resposta;
    }

    /**
     * Envia uma mensagem de correio eletrônico para um destinatário.
     * @param remetente O remetente da mensagem.
     * @param destinatário O destinatário da mensagem.
     * @param assunto O assunto da mensagem.
     * @param conteúdo O conteúdo da mensagem, podendo conter várias linhas de te
     */
    // Observe que este método *não* é public
    void enviarMensagem(String remetente, String destinatário, String assunto, St
      CaixaPostal caixaDestino = new CaixaPostal(destinatário);
      caixaDestino.inserir(new Mensagem(remetente, assunto, conteúdo));
      caixaDestino.salvar();
    }
    // um mainzinho para testar
    public static void main(String[] args) {
      if(args.length != 1) {
        System.err.println("Sintaxe: java CorreioIUTemp4 nome");
        System.exit(1);
      }
      CorreioIUTemp4 ciu = new CorreioIUTemp4(args[0]);
      ciu.interfaceComUsuário();
    }
  }
```

- Para deixar claro que modularizar com métodos é importante, examine como ficaria feio e difícil de entender o código sem modularização
  - Ver **CorreioIUTemp5.java**

```
// ***************************************
// ATENÇÃO: CÓDIGO FEIO à FRENTE!!!!!!!!!!
// ***************************************
// interface com o usuário para um sisteminha de correio eletrônico
// a classe ainda não faz parte do package p1.aplic.correio
import p1.aplic.correio.*;
import p1.io.*;
import java.util.*;

public class CorreioIUTemp5 {
  private CaixaPostal caixa;

  public CorreioIUTemp5(String titular) {
    caixa = new CaixaPostal(titular);
  }

  public void interfaceComUsuário() {
```

```
    while(true) {
      Iterator it = caixa.iterator();
      if(!it.hasNext()) {
        System.out.println("Nao ha mensagem.");
        return;
      }
      int numMensagem = 1;
      while(it.hasNext()) {
        Mensagem m = (Mensagem)it.next();
        String cursor = " ";
        if(m == caixa.mensagemCorrente()) {
          cursor = ">";
        }
        System.out.println(cursor + numMensagem +
                           " Rem: " + m.getRemetente() +
                           " Data: " + m.getDataEnvio().DDMMAAAAHHMM() +
                           " Ass: " + m.getAssunto());
        numMensagem++;
      }
      String cmd = Entrada.in.lerLinha("exibir, enviar, excluir, +, -, quit? ")
      if(cmd.startsWith("exi")) {
        Mensagem m = caixa.mensagemCorrente();
        if(m != null) {
          m.exibir();
        }
      } else if(cmd.startsWith("en")) {
        String destinatário;
        while((destinatário = Entrada.in.lerLinha("Para quem? ")).equals("")) {
          System.out.println("Favor fornecer alguma informacao");
        }
        String assunto;
        while((assunto = Entrada.in.lerLinha("Assunto? ")).equals("")) {
          System.out.println("Favor fornecer alguma informacao");
        }
        String conteúdo = "";
        String separador = System.getProperty("line.separator");
        String linha;
        System.out.println("Conteudo da mensagem? (. para terminar) ");
        while(!(linha = Entrada.in.lerLinha("")).equals(".")) {
          conteúdo += linha + separador;
        }
        CaixaPostal caixaDestino = new CaixaPostal(destinatário);
        caixaDestino.inserir(new Mensagem(caixa.getTitular(), assunto, conteúdo
        caixaDestino.salvar();
      } else if(cmd.startsWith("exc")) {
        caixa.excluir();
      } else if(cmd.startsWith("+")) {
        caixa.avançar();
      } else if(cmd.startsWith("-")) {
        caixa.recuar();
      } else if(cmd.startsWith("q")) {
        cmd = Entrada.in.lerLinha("Salvar CaixaPostal? ");
        if(cmd.startsWith("s")) {
          caixa.salvar();
        }
        break;
      } else {
        System.out.println("Comando <" + cmd + "> desconhecido");
      }
    }
```

```
    }

    // um mainzinho para testar
    public static void main(String[] args) {
      if(args.length != 1) {
        System.err.println("Sintaxe: java CorreioIUTemp5 nome");
        System.exit(1);
      }
      CorreioIUTemp5 ciu = new CorreioIUTemp5(args[0]);
      ciu.interfaceComUsuário();
    }
}
```

- O código é muito mais difícil de ler, entender e modificar
  - Você pode desconfiar disso quando vê que o método interfaceComUsuário() ficou muito grande.
- Saber quais métodos escrever, com que parâmetros e com que valor de retorno é uma *arte* que os alunos vão adquirir com experiência
- Para os interessados, aqui está a implementação completa de CorreioIU1.java
  - Mas não vamos discutir isso em aula
  - Observe que tem documentação javadoc completa

```
/*
 * Desenvolvido para a disciplina Programacao 1
 * Curso de Bacharelado em Ciência da Computação
 * Departamento de Sistemas e Computação
 * Universidade Federal da Paraíba
 *
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 * Não redistribuir sem permissão.
 */
package p1.aplic.correio;

import p1.io.*;
import p1.util.*;
import java.util.*;

/**
 * Classe que implementa uma interface simples (a caractere)
 * de manipulação de mensagens de correio eletrônico.
 * <p>
 * O programa manipula uma caixa postal de mensagens de correio eletrônico.
 * O funcionamento da interface é como segue. Um objeto dessa
 * classe deve ser criado com um argumento especificando o titular da caixa pos
 * a ser manipulada. Ao chamar o método principal (interfaceComUsuário()),
 * o conteúdo da caixa postal pode ser manipulado e novas mensagens podem ser e
 *
 * @author   Jacques Philippe Sauvé, jacques@dsc.ufpb.br
 * @version 1.0
 * <br>
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 */

public class CorreioIU1 {
    private CaixaPostal caixa;

    /**
     * Construtor de uma interface com o usuário para manipular
     * uma caixa postal de correio eletrônico.
     * @param titular O titular da caixa postal que se deseja manipular.
```

```
     */
    public CorreioIU1(String titular) {
        caixa = new CaixaPostal(titular);
    }

    /**
     * Interface com o usuário para a manipulação de caixa postal.
     * A interface consiste basicamente de um laço de interpretação de comandos
     * O conteúdo da caixa postal é mostrado na tela e um menu de comandos
     * é exibido. Os comandos disponíveis são:
     * <p><strong>exibir</strong>: exibe a mensagem corrente.
     * <p><strong>enviar</strong>: envia uma nova mensagem de correio.
     * Neste caso, deve-se especificar ainda o destinatário, o assunto e
     * o conteúdo da mensagem propriamente dita. O remetente será o titular da
     * <p><strong>excluir</strong>: marca a mensagem corrente para exclusão. Ch
     * de <strong>exclusão lógica</strong>. A exclusão em sí (física)
     * é feita ao salvar a caixa postal no final (vide o comando "quit").
     * <p><strong>+</strong>: avança para a próxima mensagem.
     * <p><strong>-</strong>: recua para a mensagem anterior.
     * <p><strong>quit</strong>: encerra a manipulação da caixa postal. Pede-se
     * se a caixa postal deve ser salva em disco ou não. Responda com 's' ou 'n
     * <p>Os comandos podem ser digitados de forma abreviada, desde que um núme
     * suficiente de letras seja informado. Por exemplo, basta digitar "en" par
     * enviar uma mensagem nova. Digitar "ex" não é suficiente, pois há dois co
     * com "ex". Deve-se digitar pelo menos "exc" (excluir) ou "exi" (exibir).
     */
    public void interfaceComUsuário() {
        while(true) {
            mostraResumoCaixaPostal(caixa);
            String cmd = Entrada.in.lerLinha("exibir, enviar, excluir, +, -, qu
            if(cmd.startsWith("exi")) {
                Mensagem m = caixa.mensagemCorrente();
                if(m != null) {
                    m.exibir();
                }
            } else if(cmd.startsWith("en")) {
                String destinatário = obtemUmaLinha("Para quem? ");
                String assunto = obtemUmaLinha("Assunto? ");
                String conteúdo = obtemVáriasLinhas("Conteudo da mensagem? (. p
                enviarMensagem(caixa.getTitular(), destinatário, assunto, conte
            } else if(cmd.startsWith("exc")) {
                caixa.excluir();
            } else if(cmd.startsWith("+")) {
                caixa.avançar();
            } else if(cmd.startsWith("-")) {
                caixa.recuar();
            } else if(cmd.startsWith("q")) {
                cmd = Entrada.in.lerLinha("Salvar CaixaPostal? ");
                if(cmd.startsWith("s")) {
                    caixa.salvar();
                }
                break;
            } else {
                System.out.println("Comando <" + cmd + "> desconhecido");
            }
        }
    }

    /**
     * Exibe na saída padrão um resumo da mensagens presentes na caixa postal.
```

```
     * @param caixa A caixa postal a ser exibida
     */
    // Observe que este método *não* é public
    private void mostraResumoCaixaPostal(CaixaPostal caixa) {
        Iterator it = caixa.iterator();
        if(!it.hasNext()) {
            System.out.println("Nao ha mensagem.");
            return;
        }
        // -16.16s significa um string (s) alinhado à esquerda (-)
        // com um mínimo de 16 caracteres e um máximo de 16 caracteres
        Formata f1 = new Formata("%-16.16s");
        Formata f2 = new Formata("%-40.40s");
        Formata f3 = new Formata("%3d");
        Formata f4 = new Formata("%-12.12s");
        System.out.println("  Num " + f4.form("Remetente") +
                           " " + f1.form("Data") +
                           " " + f2.form("Assunto"));
        int numMensagem = 1;
        while(it.hasNext()) {
            Mensagem m = (Mensagem)it.next();
            String cursor = " ";
            if(m == caixa.mensagemCorrente()) {
                cursor = ">";
            }
            String estado = m.isExcluída() ? "X" : " ";
            System.out.println(cursor + estado +
                               f3.form(numMensagem) + " " +
                               f4.form(m.getRemetente()) + " " +
                               f1.form(m.getDataEnvio().DDMMAAAAHHMM()) + " "
                               f2.form(m.getAssunto()));
            numMensagem++;
        }
    }

    /** Método auxiliar para obter uma linha da entrada padrão, tendo
     * o cuidado de não aceitar uma linha vazia.
     * @param prompt O prompt a exibir ao usuário antes de ler a informação.
     * @return A linha lida.
     */
    // Observe que este método *não* é public
    private String obtemUmaLinha(String prompt) {
        String linha;
        while((linha = Entrada.in.lerLinha(prompt)).equals("")) {
            System.out.println("Favor fornecer alguma informacao");
        }
        return linha;
    }

    /** Método auxiliar para obter várias linhas da entrada padrão.
     * O final da entrada é indicado digitando "." sozinho no início
     * de uma linha.
     * @param prompt O prompt a exibir ao usuário antes de ler a informação.
     * @return As linhas lida, como string único. As linhas são separadas
     * por um caractere de separação de linha apropriado.
     */
    // Observe que este método *não* é public
    private String obtemVáriasLinhas(String prompt) {
        String resposta = "";
        String separador = System.getProperty("line.separator");
```

```
        String linha;
        System.out.println(prompt);
        while(!(linha = Entrada.in.lerLinha("")).equals(".")) {
            resposta += linha + separador;
        }
        return resposta;
    }

    /**
     * Envia uma mensagem de correio eletrônico para um destinatário.
     * @param remetente O remetente da mensagem.
     * @param destinatário O destinatário da mensagem.
     * @param assunto O assunto da mensagem.
     * @param conteúdo O conteúdo da mensagem, podendo conter várias linhas de
     */
    // Observe que este método *não* é public
    private void enviarMensagem(String remetente, String destinatário, String a
        CaixaPostal caixaDestino = new CaixaPostal(destinatário);
        caixaDestino.inserir(new Mensagem(remetente, assunto, conteúdo));
        caixaDestino.salvar();
    }
}
```

- Observações finais com respeito a todo este exercício:
  - O programa implementa apenas a interface com o usuário e apoia-se em duas classes adicionais para tratar da "lógica do negócio": CaixaPostal e Mensagem
    - É uma excelente prática isolar a interface com o usuário da lógica de negócio
    - CaixaPostal e Mensagem poderiam ser usadas com interface gráfica, por exemplo

A classe Mensagem

- Lembre a documentação da classe Mensagem
- Vamos agora ver a implementação da classe Mensagem (em Mensagem.java)
- Não faremos desenvolvimento TDD aqui por falta de tempo em aula
  - Como prática, crie os testes para criar as classes Mensagem e CaixaPostal

```
/*
 * Desenvolvido para a disciplina Programacao 1
 * Curso de Bacharelado em Ciência da Computação
 * Departamento de Sistemas e Computação
 * Universidade Federal da Paraíba
 *
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 * Não redistribuir sem permissão.
 */

import p1.aplic.geral.*;
import java.io.*;

/**
 * Classe que representa uma mensagem normal de correio eletronico.
 *
 * Uma mensagem contém um remetente, um assunto uma data de envio e algum conte
 * O conteúdo depende do tipo exato de mensagem (textual, áudio).
 * Uma mensagem pode ser exibida (lida) e marcada para exclusão.
 *
 * @author   Jacques Philippe Sauvé, jacques@dsc.ufpb.br
 * @version 1.0
```

```
 * <br>
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 */

public class Mensagem {
    private static final int LIDA = 0x1;
    private static final int EXCLUÍDA = 0x2;
    private static final int NOVA = ~(LIDA | EXCLUÍDA);

    private String  remetente;
    private String  assunto;
    private String  conteúdo;
    private Data    dataEnvio;
    private int     estado;

    public Mensagem(String remetente, String assunto, String conteúdo) {
        this.remetente = remetente;
        this.assunto = assunto;
        this.conteúdo = conteúdo;
        dataEnvio = new Data();
        estado = NOVA;
    }

    /**
     * Recupera o remetente da mensagem
     * @return O remetente da mensagem
     */
    public String getRemetente() {
        return remetente;
    }

    /**
     * Recupera o assunto da mensagem
     * @return O assunto da mensagem
     */
    public String getAssunto() {
        return assunto;
    }

    /**
     * Recupera o conteúdo da mensagem
     * @return O conteúdo da mensagem
     */
    public String getConteúdo() {
        return conteúdo;
    }

    /**
     * Recupera a data de envio da mensagem
     * @return A data de envio da mensagem
     */
    public Data getDataEnvio() {
        return dataEnvio;
    }

    /**
     * Informa se a mensagem foi lida ou não
     * @return true se a mensagem foi lida, false caso contrário
     */
    public boolean isLida() {
```

```
        return (estado & LIDA) == LIDA;
    }

    /**
     * Informa se a mensagem foi excluída ou não
     * @return true se a mensagem foi excluída, false caso contrário
     */
    public boolean isExcluída() {
        return (estado & EXCLUÍDA) == EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como excluída.
     * A exclusão deve ser feita pela coleção que armazena as mensagens.
     * Um exemplo de tal coleção é CaixaPostal.
     */
    public void excluir() {
        estado |= EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como não excluída.
     */
    public void marcarNãoExcluída() {
        estado &= ~EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como não lida.
     */
    public void marcarNãoLida() {
        estado &= ~LIDA;
    }

    /**
     * Testa a igualdade de um objeto com esta mensagem.
     * @param objeto O objeto a comparar com esta mensagem.
     * @return true se o objeto for igual a esta mensagem, false caso contrário
     */
    public boolean equals(Object objeto) {
        if(! (objeto instanceof Mensagem)) {
            return false;
        }
        Mensagem outra = (Mensagem)objeto;
        return getRemetente().equals(outra.getRemetente())
                && getAssunto().equals(outra.getAssunto())
                && getConteúdo().equals(outra.getConteúdo());
    }

    /**
     * Exibir a mensagem. Os dados da mensagem são apresentados na saída padrão
     * Após este método, a mensagem é considerada "lida".
     */
    public void exibir() {
        System.out.println("De: " + remetente);
        System.out.println("Data: " + dataEnvio.DDMMAAAAHHMM());
        System.out.println("Assunto: " + assunto);
        System.out.println(conteúdo);
        estado |= LIDA;
    }
```

```
    /**
     * Forneça uma representação da mensagem como String
     * @return A representação da mensagem como String.
     */
    public String toString() {
        return "Remetente: " + remetente +
               ", Data: " + dataEnvio.DDMMAAAAHHMM() +
               ", Assunto: " + assunto +
               ", Conteúdo: " + conteúdo;
    }
}
```

- Neste código:
  - Veja como o estado da mensagem é representado por dois bits (LIDA e EXCLUIDA)
    - Há muito uso de operadores de bits:
    - & é AND bit-a-bit
    - | é OR bit-a-bit
    - ~ é NOT bit-a-bit
  - Veja que a mensagem é considerada lida quando é exibida para o usuário

A classe CaixaPostal

- Lembre a documentação da classe CaixaPostal
- Agora, podemos ver a implementação de CaixaPostal (em CaixaPostal.java)

```
/*
 * Desenvolvido para a disciplina Programacao 1
 * Curso de Bacharelado em Ciência da Computação
 * Departamento de Sistemas e Computação
 * Universidade Federal da Paraíba
 *
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 * Não redistribuir sem permissão.
 */
package p1.aplic.correio;

import java.io.*;
import java.util.*;

/**
 * Classe que representa uma caixa de mensagens de correio eletronico.
 * Uma caixa pode conter várias mensagens.
 * Uma caixa pertence a um "titular".
 * <p>A caixa inclui um "cursor" de mensagem. Isto é, existe
 * o conceito de "mensagem corrente" e pode-se avançar e recuar
 * na lista de mensagens (mudando assim a mensagem corrente).
 * <p>Algumas operações podem ser aplicadas à mensagem corrente: excluir, por e
 * Outras operações se aplicam à caixa como um todo (salvar, removeCaixaPostal)
 * <p>A caixa postal é salva em disco com a operação salvar.
 * Ao criar uma caixa postal, caso exista uma cópia em disco, a caixa
 * é inicializada com as mensagens que estão no disco.
 *
 * @author   Jacques Philippe Sauvé, jacques@dsc.ufpb.br
 * @version 1.1
 * <br>
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 */
```

```
public class CaixaPostal implements Serializable {
  static final long serialVersionUID = 7121517428757951951L;
  private List mensagens;
  private int índiceMensagemCorrente;
  private String titular;

  /**
   * Construção de uma CaixaPostal anônima (sem titular).
   */
  public CaixaPostal() {
    this("");
  }

  /**
   * Construção de uma Caixa Postal
   * Caso haja uma cópia da caixa postal deste titular em disco,
   * as mensagens em disco são carregados na caixa postal sendo criada.
   * No início, a mensagem corrente será a primeira mensagem da caixa.
   * @param titular O titular da caixa postal.
   */
  public CaixaPostal(String titular) {
    this.titular = titular;
    ObjectInputStream in = null;
    try {
      try {
        in = new ObjectInputStream(new FileInputStream(getNomeArquivo()));
        mensagens = (List)in.readObject();
        in.close();
      } catch( FileNotFoundException e ) {
        // nao achar o arquivo significa que estamos começando do zero
        mensagens = new ArrayList();
      }
    } catch(Exception e) {
      System.err.println(e);
      System.exit(1);
    }
    índiceMensagemCorrente = Math.min(0, mensagens.size()-1);
  }

  protected String getNomeArquivo() {
    return titular + ".correio";
  }
  /**
   * Recupera o titular da caixa postal
   * @return O titular da caixa postal.
   */
  public String getTitular() {
    return titular;
  }

  /**
   * Recupera o número de mensagens na caixa postal
   * @return O número de mensagens na caixa postal
   */
  public int númeroDeMensagens() {
    return mensagens.size();
  }

  /**
   * Insira uma nova mensagem no final da caixa postal
```

```
     * @param m A Mensagem sendo inserida.
     */
    public void inserir(Mensagem m) {
      mensagens.add(m);
      índiceMensagemCorrente = Math.max(índiceMensagemCorrente, 0);
    }

    /**
     * Recupera a mensagem corrente.
     * <p>A caixa inclui um "cursor" de mensagem. Isto é, existe
     * o conceito de "mensagem corrente" e pode-se avançar e recuar
     * na lista de mensagens (mudando assim a mensagem corrente).
     * @return A mensagem corrente.
     */
    public Mensagem mensagemCorrente() {
      return índiceMensagemCorrente >= 0 ? (Mensagem)mensagens.get(índiceMensagem
    }

    /**
     * Avançar o cursor da caixa postal. A mensagem corrente passa a ser a próxim
     * caso haja. Se a mensagem corrente for a última da caixa postal, não há mud

     */
    public void avançar() {
      índiceMensagemCorrente++;
      índiceMensagemCorrente = Math.min(índiceMensagemCorrente, mensagens.size()-
    }

    /**
     * Recuar o cursor da caixa postal. A mensagem corrente passa a ser a anterio
     * caso haja. Se a mensagem corrente for a primeira da caixa postal, não há m
     */
    public void recuar() {
      índiceMensagemCorrente--;
      índiceMensagemCorrente = Math.max(índiceMensagemCorrente, 0);
    }

    /**
     * Forneça um iterador para as mensagens da caixa postal.
     * @return O iterador de mensagens.
     */
    public Iterator iterator() {
      return mensagens.iterator();
    }

    /**
     * Excluir a mensagem Corrente da caixa postal.
     * A exclusão é apenas lógica. A mensagem está marcada para ser excluída
     * mas só é, de fato, excluída ao salvar a caixa postal.
     * @return true, se houve mensagem excluída, false caso contrário (caixa vazi
     */
    public boolean excluir() {
      if(índiceMensagemCorrente >= 0 && índiceMensagemCorrente < mensagens.size()
        mensagemCorrente().excluir();
        índiceMensagemCorrente = Math.min(índiceMensagemCorrente, mensagens.size(
        return true;
      } else {
        return false;
      }
    }
```

```
  /**
   * Salvar a caixa postal em disco.
   * Neste momento, as mensagens marcadas para exclusão são removidas
   * (isto é, não são gravadas em disco)
   */
  public void salvar() {
    // primeiro, remover as mensagens excluídas
    Iterator it = iterator();
    while(it.hasNext()) {
      Mensagem m = (Mensagem)it.next();
      if(m.isExcluída()) {
        it.remove();
      }
    }
    ObjectOutputStream out = null;
    try {
      try {
        out = new ObjectOutputStream(new FileOutputStream(getNomeArquivo()));
      } catch(FileNotFoundException e) {
        System.err.println("Nao pode criar " + getNomeArquivo());
        System.exit(1);
      }
      out.writeObject(mensagens);
      out.close();
    } catch(IOException e) {
      System.err.println(e);
      System.exit(1);
    }
    índiceMensagemCorrente = Math.min(índiceMensagemCorrente, mensagens.size()-
  }

  /**
   * Remove uma caixa postal armazenada em disco.
   * @param titular O titular da caixa postal a ser removida.
   */
  public static void removeCaixaPostal(String titular) {
    File f = new File(titular + ".correio");
    try {
      f.delete();
    } catch(Exception e) {}
  }
}
```

- Algumas observações sobre a classe CaixaPostal
  - Por enquanto, esqueça de "implements Serializable"
  - Por enquanto, esqueça da parte de manipulação de arquivos
  - Math.min(...) e Math.max(...) são usados para achar o mínimo e máximo entre dois valores
  - O atributo índiceMensagemCorrente é usado para manter controle da mensagem corrente da CaixaPostal
    - Dizemos que a CaixaPostal tem um "cursor"

Manipulando outros tipos de mensagens: Interfaces e Polimorfismo

Manutenção de programas

- Vamos fazer "manutenção" ao nosso programa
  - Manutenção é uma atividade extremamente comum feita por programadores
  - Software não é uma coisa estática que não muda depois de feita
  - Há *sempre* mudanças a fazer em programas que são utilizados

    - Programas que não precisam mudar mais morreram: ninguém os está utilizando
- Nosso problema de manutenção: nosso usuário deseja manipular mensagem de áudio
    - Para simplificar nosso trabalho, vamos manipular clipes de som já gravados e não gerenciar a gravação com microfone, etc. (embora isso não fosse muito difícil fazer)

O programa principal

- O programa principal segue abaixo e está em Correio2.java

```
/*
 * Polimorfismo. Programa principal.
 */
import p1.aplic.correio.*;

public class Correio2 {
    public static void main(String[] args) {
        if(args.length != 1) {
            System.err.println("Sintaxe: java Correio2 nome");
            System.exit(1);
        }
        CorreioIU2 ciu = new CorreioIU2(args[0]);
        ciu.interfaceComUsuário();
    }
}
```

- Parece semelhante a Correio1, mas usa-se a classe CorreioIU2 para a interface com o usuário

A classe CorreioIU2

- Vamos ver uma sessão de uso deste programa:

```
C:\...\src>java -classpath .;packagep1\p1.jar Correio2 jacques
Nao ha mensagem.
exibir, texto, voz, excluir, +, -, quit? t
Para quem? ana
Assunto? vamos forrozar?
Conteudo da mensagem? (. para terminar)
Hoje aa noite, na Parque do Povo, ta bem?
tchau
o fofo
.
Nao ha mensagem.
exibir, texto, voz, excluir, +, -, quit? v
Para quem? ana
Assunto? uma musica pra voce ...
Arquivo de clip de audio? clip1.mid
Nao ha mensagem.
exibir, texto, voz, excluir, +, -, quit? q
Salvar CaixaPostal? s

C:\...\src>java -classpath .;packagep1\p1.jar Correio2 ana

  Num Remetente    Data             Assunto
>   1 jacques      29/06/2001 10:24 vamos forrozar?
    2 jacques      29/06/2001 10:24 uma musica pra voce ...
exibir, texto, voz, excluir, +, -, quit? exi
De: jacques
Data: 29/06/2001 10:24
Assunto: vamos forrozar?
```

```
Hoje aa noite, na Parque do Povo, ta bem?
tchau
o fofo

  Num Remetente    Data              Assunto
>   1 jacques      29/06/2001 10:24 vamos forrozar?
    2 jacques      29/06/2001 10:24 uma musica pra voce ...
exibir, texto, voz, excluir, +, -, quit? exc
  Num Remetente    Data              Assunto
>X  1 jacques      29/06/2001 10:24 vamos forrozar?
    2 jacques      29/06/2001 10:24 uma musica pra voce ...
exibir, texto, voz, excluir, +, -, quit? +
  Num Remetente    Data              Assunto
 X  1 jacques      29/06/2001 10:24 vamos forrozar?
>   2 jacques      29/06/2001 10:24 uma musica pra voce ...
exibir, texto, voz, excluir, +, -, quit? exi
Se tiver multimidia no computador, o clip deve estar tocando
  Num Remetente    Data              Assunto
 X  1 jacques      29/06/2001 10:24 vamos forrozar?
>   2 jacques      29/06/2001 10:24 uma musica pra voce ...
exibir, texto, voz, excluir, +, -, quit? exc
  Num Remetente    Data              Assunto
 X  1 jacques      29/06/2001 10:24 vamos forrozar?
>X  2 jacques      29/06/2001 10:24 uma musica pra voce ...
exibir, texto, voz, excluir, +, -, quit? q
Salvar CaixaPostal? s
```

- Agora, podemos ver a implementação de CorreioIU2 (em CorreioIU2.java)

```
/*
 * Desenvolvido para a disciplina Programacao 1
 * Curso de Bacharelado em Ciência da Computação
 * Departamento de Sistemas e Computação
 * Universidade Federal da Paraíba
 *
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 * Não redistribuir sem permissão.
 */
package p1.aplic.correio;

import p1.io.*;
import p1.util.*;
import java.util.*;

/**
 * Classe que implementa uma interface simples (a caractere)
 * de manipulação de mensagens de correio eletrônico.
 * <p>
 * O programa manipula uma caixa postal de mensagens de correio eletrônico.
 * O funcionamento da interface é como segue. Um objeto dessa
 * classe deve ser criado com um argumento especificando o titular da caixa pos
 * a ser manipulada. Ao chamar o método principal (interfaceComUsuário()),
 * o conteúdo da caixa postal pode ser manipulado e novas mensagens podem ser e
 * Pode-se enviar correio textual ou de áudio.
 *
 * @author   Jacques Philippe Sauvé, jacques@dsc.ufpb.br
 * @version 1.0
 * <br>
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 */
```

```
public class CorreioIU2 {
    private CaixaPostal caixa;

    /**
     * Construtor de uma interface com o usuário para manipular
     * uma caixa postal de correio eletrônico.
     * @param titular O titular da caixa postal que se deseja manipular.
     */
    public CorreioIU2(String titular) {
        caixa = new CaixaPostal(titular);
    }

    /**
     * Interface com o usuário para a manipulação de caixa postal.
     * A interface consiste basicamente de um laço de interpretação de comandos
     * O conteúdo da caixa postal é mostrado na tela e um menu de comandos
     * é exibido. Os comandos disponíveis são:
     * <p><strong>exibir</strong>: exibe a mensagem corrente.
     * <p><strong>texto</strong>: envia uma nova mensagem textual de correio.
     * Neste caso, deve-se especificar ainda o destinatário, o assunto e
     * o conteúdo da mensagem propriamente dita. O remetente será o titular da
     * <p><strong>voz</strong>: envia uma nova mensagem de correio usando áudio
     * Neste caso, deve-se especificar ainda o destinatário, o assunto e
     * o arquivo contendo o áudio da mensagem. Essa interface estranha foi usad
     * não queremos obrigar o uso de um microfone para gravar a mensagem.
     * O remetente será o titular da caixa postal.
     * <p><strong>excluir</strong>: marca a mensagem corrente para exclusão. Ch
     * de <strong>exclusão lógica</strong>. A exclusão em sí (física)
     * é feita ao salvar a caixa postal no final (vide o comando "quit").
     * <p><strong>+</strong>: avança para a próxima mensagem.
     * <p><strong>-</strong>: recua para a mensagem anterior.
     * <p><strong>quit</strong>: encerra a manipulação da caixa postal. Pede-se
     * se a caixa postal deve ser salva em disco ou não. Responda com 's' ou 'n
     * <p>Os comandos podem ser digitados de forma abreviada, desde que um núme
     * suficiente de letras seja informado. Por exemplo, basta digitar "v" para
     * enviar uma mensagem nova de áudio.
     * Digitar "ex" não é suficiente, pois há dois comandos começando
     * com "ex". Deve-se digitar pelo menos "exc" (excluir) ou "exi" (exibir).
     */
    public void interfaceComUsuário() {
        while(true) {
            mostraResumoCaixaPostal(caixa);
            String cmd = Entrada.in.lerLinha("exibir, texto, voz, excluir, +, -
            if(cmd.startsWith("exi")) {
                Mensagem m = caixa.mensagemCorrente();
                if(m != null) {
                    m.exibir();
                }
            } else if(cmd.startsWith("t")) {
                String destinatário = obtemUmaLinha("Para quem? ");
                String assunto = obtemUmaLinha("Assunto? ");
                String conteúdo = obtemVáriasLinhas("Conteudo da mensagem? (. p
                enviarMensagemTexto(caixa.getTitular(), destinatário, assunto,
            } else if(cmd.startsWith("v")) {
                String destinatário = obtemUmaLinha("Para quem? ");
                String assunto = obtemUmaLinha("Assunto? ");
                String clip = obtemUmaLinha("Arquivo de clip de audio? ");
                enviarMensagemÁudio(caixa.getTitular(), destinatário, assunto,
            } else if(cmd.startsWith("exc")) {
```

```
                caixa.excluir();
            } else if(cmd.startsWith("+")) {
                caixa.avançar();
            } else if(cmd.startsWith("-")) {
                caixa.recuar();
            } else if(cmd.startsWith("q")) {
                cmd = Entrada.in.lerLinha("Salvar CaixaPostal? ");
                if(cmd.startsWith("s")) {
                    caixa.salvar();
                }
                break;
            } else {
                System.out.println("Comando <" + cmd + "> desconhecido");
            }
        }
    }

    /**
     * Exibe na saída padrão um resumo da mensagens presentes na caixa postal.
     * @param caixa A caixa postal a ser exibida
     */
    // Observe que este método *não* é public
    private void mostraResumoCaixaPostal(CaixaPostal caixa) {
        Iterator it = caixa.iterator();
        if(!it.hasNext()) {
            System.out.println("Nao ha mensagem.");
            return;
        }
        // -16.16s significa um string (s) alinhado à esquerda (-)
        // com um mínimo de 16 caracteres e um máximo de 16 caracteres
        Formata f1 = new Formata("%-16.16s");
        Formata f2 = new Formata("%-40.40s");
        Formata f3 = new Formata("%3d");
        Formata f4 = new Formata("%-12.12s");
        System.out.println("  Num " + f4.form("Remetente") +
                           " " + f1.form("Data") +
                           " " + f2.form("Assunto"));
        int numMensagem = 1;
        while(it.hasNext()) {
            Mensagem m = (Mensagem)it.next();
            String cursor = " ";
            if(m == caixa.mensagemCorrente()) {
                cursor = ">";
            }
            String estado = m.isExcluída() ? "X" : " ";
            System.out.println(cursor + estado +
                               f3.form(numMensagem) + " " +
                               f4.form(m.getRemetente()) + " " +
                               f1.form(m.getDataEnvio().DDMMAAAAHHMM()) + " "
                               f2.form(m.getAssunto()));
            numMensagem++;
        }
    }

    /** Método auxiliar para obter uma linha da entrada padrão, tendo
     * o cuidado de não aceitar uma linha vazia.
     * @param prompt O prompt a exibir ao usuário antes de ler a informação.
     * @return A linha lida.
     */
    // Observe que este método *não* é public
```

```
    private String obtemUmaLinha(String prompt) {
        String linha;
        while((linha = Entrada.in.lerLinha(prompt)).equals("")) {
            System.out.println("Favor fornecer alguma informacao");
        }
        return linha;
    }

    /** Método auxiliar para obter várias linhas da entrada padrão.
     * O final da entrada é indicado digitando "." sozinho no início
     * de uma linha.
     * @param prompt O prompt a exibir ao usuário antes de ler a informação.
     * @return As linhas lida, como string único. As linhas são separadas
     * por um caractere de separação de linha apropriado.
     */
    // Observe que este método *não* é public
    private String obtemVáriasLinhas(String prompt) {
        String resposta = "";
        String separador = System.getProperty("line.separator");
        String linha;
        System.out.println(prompt);
        while(!(linha = Entrada.in.lerLinha("")).equals(".")) {
            resposta += linha + separador;
        }
        return resposta;
    }

    /**
     * Envia uma mensagem textual de correio eletrônico para um destinatário.
     * @param remetente O remetente da mensagem.
     * @param destinatário O destinatário da mensagem.
     * @param assunto O assunto da mensagem.
     * @param conteúdo O conteúdo da mensagem, podendo conter várias linhas de
     */
    // Observe que este método *não* é public
    private void enviarMensagemTexto(String remetente, String destinatário, Str
        CaixaPostal caixaDestino = new CaixaPostal(destinatário);
        caixaDestino.inserir(new MensagemTexto(remetente, assunto, conteúdo));
        caixaDestino.salvar();
    }

    /**
     * Envia uma mensagem de áudio de correio eletrônico para um destinatário.
     * @param remetente O remetente da mensagem.
     * @param destinatário O destinatário da mensagem.
     * @param assunto O assunto da mensagem.
     * @param clip O arquivo de áudio contendo a mensagem.
     */
    // Observe que este método *não* é public
    private void enviarMensagemÁudio(String remetente, String destinatário, Str
        CaixaPostal caixaDestino = new CaixaPostal(destinatário);
        caixaDestino.inserir(new MensagemAudio(remetente, assunto, clip));
        caixaDestino.salvar();
    }
}
```

- O que há de novo aqui em comparação a CorreioIU1?
  - Os métodos enviarMensagemTexto() e enviarMensagemÁudio() são usados para os diferentes tipos de mensagens de correio
  - Há uma classe para cada uma das mensagens: MensagemTexto e

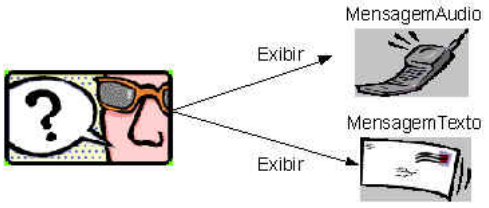
MensagemAudio
- O construtor de cada classe é ligeiramente diferente

Polimorfismo

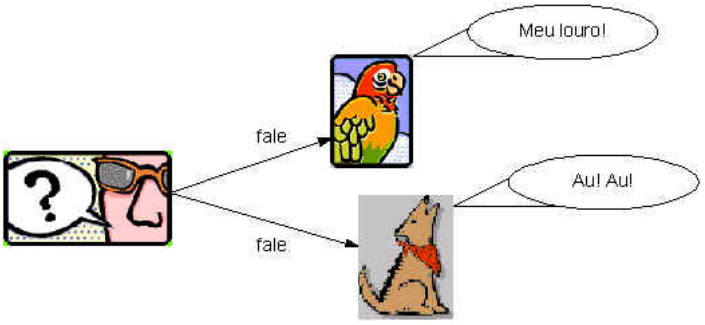
- Acabamos de tocar o clipe de áudio. Onde isso está sendo feito no código de CorreioIU2????
  - Só pode ser no seguinte trecho:

```
Mensagem m = caixa.mensagemCorrente();
if(m != null) {
    m.exibir();
}
```

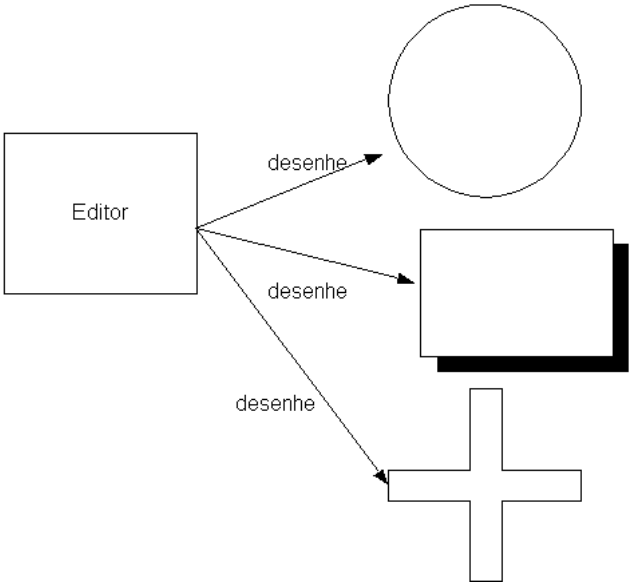
- Mas então onde estamos escrevendo a mensagem na tela quando a mensagem é de texto de não de voz?
  - No mesmo trecho de cima!
- Estamos vendo polimorfismo em ação: a mesma chamada m.exibir() faz coisas diferentes dependendo do objeto que recebe a mensagem
- A palavra "polimorfismo" significa "Que apresenta várias formas"
  - Você concorda que uma mensagem de correio apresenta duas formas para nós?
    - Forma Texto e forma Áudio
- Numa linguagem de programação, o polimorfismo permite tratar objetos de classes diferentes do mesmo jeito (com as mesmas chamadas a métodos), porque elas têm o mesmo comportamento
  - As classes fazem a mesma operação (método), mas de forma diferente
  - "O quê" é igual
  - "Como" é diferente
- Exemplos de objetos de classes diferentes que têm o mesmo comportamento:



Mensagens de correio são "exibidas"



Animais "falam"



Figuras geométricas são "desenhadas"

- O importante é que o objeto da esquerda pode tratar os objetos da direita uniformemente porque têm o mesmo comportamento
- Qual comportamento comum existe entre MensagemTexto e MensagemAudio?
  - Cada uma tem um remetente, um assunto, uma data de envio
  - Cada uma pode estar lida ou não, excluída ou não
  - Posso operar com cada objeto para: excluir, marcar não excluída, marcar não lida, exibir, etc.
- Queremos representar esse comportamento comum em Java
  - Para tanto, criamos um Tipo Abstrato de Dado (usamos mais o termo interface como se chama no mundo Java)
  - Uma interface define um novo tipo, isto é, um novo comportamento
  - Uma interface *não* define como esse comportamento é implementado
- No nosso caso, o comportamento comum entre MensagemTexto e MensagemAudio será chamado simplesmente de Mensagem

A interface Mensagem

- A implementação segue (ver Mensagem.java)

```
/*
 * Desenvolvido para a disciplina Programacao 1
 * Curso de Bacharelado em Ciência da Computação
 * Departamento de Sistemas e Computação
 * Universidade Federal da Paraíba
 *
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 * Não redistribuir sem permissão.
 */
package p1.aplic.correio;

import p1.aplic.geral.*;
import java.io.*;

/**
 * Interface para manipular uma mensagem de correio eletronico.
 * Uma mensagem contém um remetente, um assunto uma data de envio e algum conte
 * O conteúdo depende do tipo exato de mensagem (textual, áudio).
 * Uma mensagem pode ser exibida (lida) e marcada para exclusão.
 *
 * @author   Jacques Philippe Sauvé, jacques@dsc.ufpb.br
 * @version 1.0
 * <br>
```

```
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 */

public interface Mensagem  {
    /**
     * Recupera o remetente da mensagem
     * @return O remetente da mensagem
     */
    public String getRemetente();

    /**
     * Recupera o assunto da mensagem
     * @return O assunto da mensagem
     */
    public String getAssunto();

    /**
     * Recupera a data de envio da mensagem
     * @return A data de envio da mensagem
     */
    public Data getDataEnvio();

    /**
     * Informa se a mensagem foi lida ou não
     * @return true se a mensagem foi lida, false caso contrário
     */
    public boolean isLida();

    /**
     * Informa se a mensagem foi excluída ou não
     * @return true se a mensagem foi excluída, false caso contrário
     */
    public boolean isExcluída();

    /**
     * Marcar a mensagem como excluída.
     * A exclusão deve ser feita pela coleção que armazena as mensagens.
     * Um exemplo de tal coleção é CaixaPostal.
     */
    public void excluir();

    /**
     * Marcar a mensagem como não excluída.
     */
    public void marcarNãoExcluída();

    /**
     * Marcar a mensagem como não lida.
     */
    public void marcarNãoLida();

    /**
     * Testa a igualdade de um objeto com esta mensagem.
     * @param objeto O objeto a comparar com esta mensagem.
     * @return true se o objeto for igual a esta mensagem, false caso contrário
     */
    public boolean equals(Object objeto);

    /**
     * Exibir a mensagem. Isso poderá imprimir algo na saída
```

```
     * ou provocar outras saídas relacionadas com a leitura da mensagem.
     * Após este método, a mensagem é considerada "lida".
     */
    public void exibir();

    /**
     * Forneça uma representação da mensagem como String
     * @return A representação da mensagem como String.
     */
    public String toString();
}
```

- Observações
  - Numa interface, por definição, tudo é public
    - Poderíamos retirar a palavra "public", acima
  - Observe a palavra "interface" em vez de "classe"
    - Significa que não teremos implementação, apenas assinaturas de métodos
- O que fizemos até agora foi apenas definir um tipo
  - Isso só é útil se fizermos mais duas coisas
    - Fornecer uma ou mais implementações desta interface
    - Usar o tipo

Implementação de uma interface: MensagemTexto

- Vamos primeiro ver duas implementações da interface Mensagem
- Primeiro: MensagemTexto.java

```
/*
 * Desenvolvido para a disciplina Programacao 1
 * Curso de Bacharelado em Ciência da Computação
 * Departamento de Sistemas e Computação
 * Universidade Federal da Paraíba
 *
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 * Não redistribuir sem permissão.
 */

import p1.aplic.correio.*;
import p1.aplic.geral.*;
import java.io.*;

/**
 * Classe que representa uma mensagem normal de correio eletronico.
 *
 * Uma mensagem contém um remetente, um assunto uma data de envio e algum conte
 * O conteúdo depende do tipo exato de mensagem (textual, áudio).
 * Uma mensagem pode ser exibida (lida) e marcada para exclusão.
 *
 * @author   Jacques Philippe Sauvé, jacques@dsc.ufpb.br
 * @version 1.0
 * <br>
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 */

public class MensagemTexto implements Mensagem {
    private static final int LIDA = 0x1;
    private static final int EXCLUÍDA = 0x2;
    private static final int NOVA = ~(LIDA | EXCLUÍDA);
```

```
    private String  remetente;
    private String  assunto;
    private String  conteúdo;
    private Data    dataEnvio;
    private int     estado;

    public MensagemTexto(String remetente, String assunto, String conteúdo) {
        this.remetente = remetente;
        this.assunto = assunto;
        this.conteúdo = conteúdo;
        dataEnvio = new Data();
        estado = NOVA;
    }

    /**
     * Recupera o remetente da mensagem
     * @return O remetente da mensagem
     */
    public String getRemetente() {
        return remetente;
    }

    /**
     * Recupera o assunto da mensagem
     * @return O assunto da mensagem
     */
    public String getAssunto() {
        return assunto;
    }

    /**
     * Recupera o conteúdo da mensagem
     * @return O conteúdo da mensagem
     */
    public String getConteúdo() {
        return conteúdo;
    }

    /**
     * Recupera a data de envio da mensagem
     * @return A data de envio da mensagem
     */
    public Data getDataEnvio() {
        return dataEnvio;
    }

    /**
     * Informa se a mensagem foi lida ou não
     * @return true se a mensagem foi lida, false caso contrário
     */
    public boolean isLida() {
        return (estado & LIDA) == LIDA;
    }

    /**
     * Informa se a mensagem foi excluída ou não
     * @return true se a mensagem foi excluída, false caso contrário
     */
    public boolean isExcluída() {
        return (estado & EXCLUÍDA) == EXCLUÍDA;
```

```
    }

    /**
     * Marcar a mensagem como excluída.
     * A exclusão deve ser feita pela coleção que armazena as mensagens.
     * Um exemplo de tal coleção é CaixaPostal.
     */
    public void excluir() {
        estado |= EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como não excluída.
     */
    public void marcarNãoExcluída() {
        estado &= ~EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como não lida.
     */
    public void marcarNãoLida() {
        estado &= ~LIDA;
    }

    /**
     * Testa a igualdade de um objeto com esta mensagem.
     * @param objeto O objeto a comparar com esta mensagem.
     * @return true se o objeto for igual a esta mensagem, false caso contrário
     */
    public boolean equals(Object objeto) {
        if(! (objeto instanceof Mensagem)) {
            return false;
        }
        Mensagem outra = (Mensagem)objeto;
        return getRemetente().equals(outra.getRemetente())
                && getAssunto().equals(outra.getAssunto())
                && getConteúdo().equals(outra.getConteúdo());
    }

    /**
     * Exibir a mensagem. Os dados da mensagem são apresentados na saída padrão
     * Após este método, a mensagem é considerada "lida".
     */
    public void exibir() {
        System.out.println("De: " + remetente);
        System.out.println("Data: " + dataEnvio.DDMMAAAAHHMM());
        System.out.println("Assunto: " + assunto);
        System.out.println(conteúdo);
        estado |= LIDA;
    }

    /**
     * Forneça uma representação da mensagem como String
     * @return A representação da mensagem como String.
     */
    public String toString() {
        return "Remetente: " + remetente +
               ", Data: " + dataEnvio.DDMMAAAAHHMM() +
               ", Assunto: " + assunto +
```

```
                ", Conteúdo: " + conteúdo;
    }
}
```

- Essa é a mesma classe Mensagem que vimos antes (isto é, quando Mensagem era uma classe e não uma interface como agora)
  - Só mudou a cláusula "implements Mensagem" que significa que esta classe implementa a interface Mensagem
  - Isso nos obrigado a implementar cada método que pertence à interface
  - Portanto, qualquer objeto da classe MensagemTexto pode ser tratado como se fosse o tipo Mensagem
  - Observe também que há métodos implementados pela classe que não fazem parte da interface (quais, por exemplo?)

Implementação de uma interface: MensagemAudio

- Agora, vamos ver a implementação da classe MensagemAudio (em MensagemAudio.java)

```
/*
 * Desenvolvido para a disciplina Programacao 1
 * Curso de Bacharelado em Ciência da Computação
 * Departamento de Sistemas e Computação
 * Universidade Federal da Paraíba
 *
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 * Não redistribuir sem permissão.
 */

import p1.aplic.correio.*;
import p1.aplic.geral.*;
import java.io.*;
import java.net.*;
import java.applet.*;

/**
 * Classe que representa uma mensagem de áudio de correio eletronico.
 *
 * @author   Jacques Philippe Sauvé, jacques@dsc.ufpb.br
 * @version 1.0
 * <br>
 * Copyright (C) 1999 Universidade Federal da Paraíba.
 */

public class MensagemAudio implements Mensagem {
    private static final int LIDA = 0x1;
    private static final int EXCLUÍDA = 0x2;
    private static final int NOVA = ~(LIDA | EXCLUÍDA);

    private String  remetente;
    private String  assunto;
    private String  arquivoÁudio;
    private Data    dataEnvio;
    private int     estado;

    /**
     * Cria uma mensagem de áudio de correio eletrônico
     * @param remetente O remetente da mensagem
     * @param assunto O assunto da mensagem
     * @param arquivoÁudio O arquivo contendo o áudio da mensagem
     */
```

```
    public MensagemAudio(String remetente, String assunto, String arquivoÁudio)
        this.remetente = remetente;
        this.assunto = assunto;
        this.arquivoÁudio = arquivoÁudio;
        dataEnvio = new Data();
        estado = NOVA;
    }

    /**
     * Recupera o remetente da mensagem
     * @return O remetente da mensagem
     */
    public String getRemetente() {
        return remetente;
    }

    /**
     * Recupera o assunto da mensagem
     * @return O assunto da mensagem
     */
    public String getAssunto() {
        return assunto;
    }

    /**
     * Recupera o arquivo de áudio da mensagem.
     * @return O arquivo de áudio da mensagem.
     */
    public String getarquivoÁudio() {
        return arquivoÁudio;
    }

    /**
     * Recupera a data de envio da mensagem
     * @return A data de envio da mensagem
     */
    public Data getDataEnvio() {
        return dataEnvio;
    }

    /**
     * Informa se a mensagem foi lida ou não
     * @return true se a mensagem foi lida, false caso contrário
     */
    public boolean isLida() {
        return (estado & LIDA) == LIDA;
    }

    /**
     * Informa se a mensagem foi excluída ou não
     * @return true se a mensagem foi excluída, false caso contrário
     */
    public boolean isExcluída() {
        return (estado & EXCLUÍDA) == EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como excluída.
     * A exclusão deve ser feita pela coleção que armazena as mensagens.
     * Um exemplo de tal coleção é CaixaPostal.
```

```
     */
    public void excluir() {
        estado |= EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como não excluída.
     */
    public void marcarNãoExcluída() {
        estado &= ~EXCLUÍDA;
    }

    /**
     * Marcar a mensagem como não lida.
     */
    public void marcarNãoLida() {
        estado &= ~LIDA;
    }

    /**
     * Exibir a mensagem. O arquivo de áudio é tocado.
     * Após este método, a mensagem é considerada "lida".
     */
    public void exibir() {
        try {
            URL u = new URL("file", "localhost", arquivoÁudio);
            AudioClip clip = Applet.newAudioClip(u);
            System.out.println("Se tiver multimidia no computador, o clip deve
            clip.play();
        } catch(Exception e) {
            System.out.println("Nao pode abrir Audio Clip: " + arquivoÁudio);
        }
        estado |= LIDA;
    }

    /**
     * Testa a igualdade de um objeto com esta mensagem.
     * @param objeto O objeto a comparar com esta mensagem.
     * @return true se o objeto for igual a esta mensagem, false caso contrário
     */
    public boolean equals(Object objeto) {
        if(! (objeto instanceof MensagemAudio)) {
            return false;
        }
        MensagemAudio outra = (MensagemAudio)objeto;
        return super.equals(objeto) &&
               getarquivoÁudio().equals(outra.getarquivoÁudio());
    }

    /**
     * Forneça uma representação da mensagem como String
     * @return A representação da mensagem como String.
     */
    public String toString() {
        return "Remetente: " + remetente +
               ", Data: " + dataEnvio.DDMMAAAAHHMM() +
               ", Assunto: " + assunto +
               ", Arquivo de áudio: " + arquivoÁudio;
    }
}
```

- Observações sobre a implementação
  - É bem semelhante a MensagemTexto, mas:
    - Não tem atributo conteúdo nem o método getConteúdo()
    - Tem o atributo arquivoAudio e o método getarquivoAudio()
    - Alguns métodos são diferentes, principalmente exibir() porque exibir um clip de áudio é completamente diferente de imprimir uma mensagem de texto na saída padrão
  - A classe também implementa a interface Mensagem

Uso de interfaces: chamadas polimórficas

- Agora, vamos *usar* tudo isso que fizemos!
  - Na realidade, já fizemos isso: são as duas classes CorreioIU2 e CaixaPostal
- Examinemos CorreioIU2 novamente
  - Observe o uso de uma variável do tipo Mensagem nas linhas abaixo
  - Observe também a chamada m.exibir() que é uma chamada polimórfica
    - Qual dos dois métodos exibir() será executado depende da classe do objeto que está na mão num certo momento

```
Mensagem m = caixa.mensagemCorrente();
if(m != null) {
    m.exibir();
}
```

- Isso é muito poderoso, não??!!
- Agora, vamos examinar CaixaPostal novamente
  - Destacamos apenas alguns trechos interessantes aqui

```
public class CaixaPostal implements Serializable {
    private List mensagens;
    ...

    /**
     * Insira uma nova mensagem no final da caixa postal
     * @param m A Mensagem sendo inserida.
     */
    public void inserir(Mensagem m) {
        mensagens.add(m);
        índiceMensagemCorrente = Math.max(índiceMensagemCorrente, 0);
    }

    /**
     * Recupera a mensagem corrente.
     * <p>A caixa inclui um "cursor" de mensagem. Isto é, existe
     * o conceito de "mensagem corrente" e pode-se avançar e recuar
     * na lista de mensagens (mudando assim a mensagem corrente).
     * @return A mensagem corrente.
     */
    public Mensagem mensagemCorrente() {
        return índiceMensagemCorrente >= 0 ? (Mensagem)mensagens.get(índiceMens
    }

    /**
     * Excluir a mensagem Corrente da caixa postal.
     * A exclusão é apenas lógica. A mensagem está marcada para ser excluída
     * mas só é, de fato, excluída ao salvar a caixa postal.
     * @return true, se houve mensagem excluída, false caso contrário (caixa va
     */
    public boolean excluir() {
```

```
            if(índiceMensagemCorrente >= 0 && índiceMensagemCorrente < mensagens.si
                mensagemCorrente().excluir();
                índiceMensagemCorrente = Math.min(índiceMensagemCorrente, mensagens
                return true;
            } else {
                return false;
            }
        }

        /**
         * Salvar a caixa postal em disco.
         * Neste momento, as mensagens marcadas para exclusão são removidas
         * (isto é, não são gravadas em disco)
         */
        public void salvar() {
            // primeiro, remover as mensagens excluídas
            Iterator it = iterator();
            while(it.hasNext()) {
                Mensagem m = (Mensagem)it.next();
                if(m.isExcluída()) {
                    it.remove();
                }
            }
            ...
        }
        ...
}
```

- CaixaPostal usa um ArrayList como coleção
- O que essa coleção contém?
    - Objetos de classes diferentes!
- Porém, eles são todos tratados como se fossem do tipo Mensagem, através de chamadas polimórficas
    - Identifique tais chamadas no código acima

Mais manutenção de código

- Conclusão até agora: o uso de polimorfismo, através de uma interface Mensagem implementada por várias classes permite ter um código genérico em CorreioIU2 e CaixaPostal para tratar qualquer tipo de mensagem
- A vantagem disso é que alterar o programa fica mais simples
    - Considerando que um programa se comporta como gelatina, dizemos que dar um chute na gelatina (a mudança) criará poucas ondas (poucas mudanças em outra classes)
- Vamos tentar ver se isso é verdade?
- OK. Vamos introduzir um terceiro tipo de mensagem de correio eletrônico
    - Vamos criar uma classe MensagemMissaoImpossivel em que uma mensagem se auto-destroi depois de lida
- Segue uma sessão de uso (Correio3.java)

```
C:\..\src>java -classpath .;packagep1\p1.jar Correio3 jacques
Nao ha mensagem.
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? t
Para quem? ana
Assunto? msg1
Conteudo da mensagem? (. para terminar)
msg1
.
Nao ha mensagem.
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? m
```

```
Para quem? ana
Assunto? msg2
Conteudo da mensagem? (. para terminar)
msg2
.
Nao ha mensagem.
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? t
Para quem? ana
Assunto? msg3
Conteudo da mensagem? (. para terminar)
msg3
.
Nao ha mensagem.
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? q
Salvar CaixaPostal? s

C:\...\src>java -classpath .;packagep1\p1.jar Correio3 ana

  Num Remetente    Data             Assunto
>   1 jacques      29/06/2001 13:06 msg1
    2 jacques      29/06/2001 13:06 msg2
    3 jacques      29/06/2001 13:06 msg3
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? exi
De: jacques
Data: 29/06/2001 13:06
Assunto: msg1
msg1

  Num Remetente    Data             Assunto
>   1 jacques      29/06/2001 13:06 msg1
    2 jacques      29/06/2001 13:06 msg2
    3 jacques      29/06/2001 13:06 msg3
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? +
  Num Remetente    Data             Assunto
    1 jacques      29/06/2001 13:06 msg1
>   2 jacques      29/06/2001 13:06 msg2
    3 jacques      29/06/2001 13:06 msg3
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? exi
De: jacques
Data: 29/06/2001 13:06
Assunto: msg2
msg2

  Num Remetente    Data             Assunto
    1 jacques      29/06/2001 13:06 msg1
>X  2 jacques      29/06/2001 13:06 msg2
    3 jacques      29/06/2001 13:06 msg3
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? q
Salvar CaixaPostal? s


C:\...\src>java -classpath .;packagep1\p1.jar Correio3 ana

  Num Remetente    Data             Assunto
>   1 jacques      29/06/2001 13:06 msg1
    2 jacques      29/06/2001 13:06 msg3
exibir, texto, missaoimpossivel, voz, excluir, +, -, quit? q
Salvar CaixaPostal? s
```

- Como implementar?

    - Como fazer MensagemMissaoImpossivel?
    - Como alterar CorreioIU2?
    - Como alterar CaixaPostal?
- Primeiro a classe MensagemMissaoImpossivel (em MensagemMissaoImpossivel.java)
    - Muda pouco com relação a MensagemTexto
    - Ver as linhas abaixo: basicamente, uma única linha mudou

```
/**
 * Exibir a mensagem. Os dados da mensagem são apresentados na saída padrão
 * Após este método, a mensagem se auto-destroi.
 */
public void exibir() {
    System.out.println("De: " + remetente);
    System.out.println("Data: " + dataEnvio.DDMMAAAAHHMM());
    System.out.println("Assunto: " + assunto);
    System.out.println(conteúdo);
    estado |= LIDA;
    excluir();
}
```

- A interface com o usuário tem que mudar um pouco, mas de forma trivial, para poder mandar mensagens do novo tipo
    - Ver CorreioIU3.java
- A classe CaixaPostal não é alterada em nada!
- Resultado
    - Foi extremamente simples de alterar o programa para incluir um novo tipo de mensagem
    - Isolamos boa parte do código de outras partes através do uso de um interface
        - Isolamento significa que mexo num lugar mas não preciso mexer no outro
        - Interfaces podem ser vistas como barreiras que impedem a propagação de mudanças
    - Essa facilidade de manutenção (flexibilidade, facilidade de extensão) é devida ao *polimorfismo*, uma das pedras angulares da Orientação a Objeto

Mais discussão de polimorfismo

- Veja aqui

Implementação do polimorfismo: Ligação dinâmica

- A chamada polimórfica m.exibir() deve ser implementada com cuidado em tempo de execução
    - É impossível fazer o binding (a ligação entre a chamada e o código sendo chamado) em tempo de compilação ou link-edição
    - O código gerado contém algum tipo de "switch" para decidir qual versão de exibir() deve ser chamada
    - Isso é chamado de:
        - Ligação dinâmica (ou *dynamic binding* ou *late binding*)

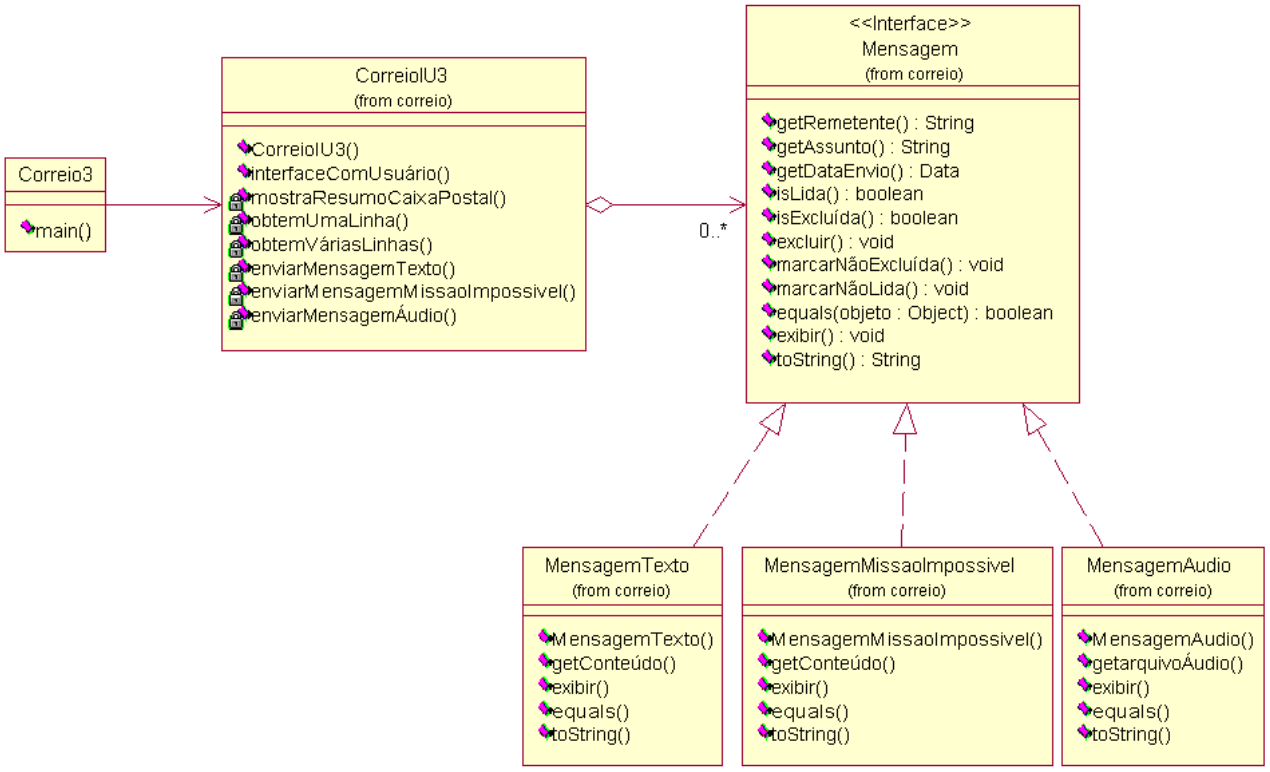
Exemplos de interfaces da API Java

- API significa "Application Programming Interface", ou as coisas que oferecemos ao programador para brincar
- Primeiro exemplo: a interface Iterator
    - Métodos: hasNext(), next(), remove()
    - Classes que implementam a interface: classes associadas às coleções ArrayList, LinkedList, ...
- Segundo exemplo: a interface Collection
    - Alguns métodos: add(...), clear(), contains(...), equals(...), iterator(), remove(...), size(), toArray()
    - Classes que implementam a interface: ArrayList, LinkedList, e muitas outras

- Terceiro exemplo: a interface Comparable
  - Método: compareTo(...)
  - Classe que implementam: um monte de classes que têm o método acima

UML

- Em UML, interfaces são representadas de duas formas
  - Com definição completa, como "classe" usando o estereótipo <<interface>>
  - De forma resumida, usando uma bola
- Exemplo: nosso último programa



- Exemplo: nosso último programa mas usando a forma abreviada de interface