CURSO: Ciência da Computação	**DATA: ____/____/ 2013**
PERÍODO: 4º.	**PROFESSOR: Andrey**
DISCIPLINA: Técnicas Alternativas de Programação	**AULA: 06**

APRESENTAÇÃO

Uso e criação de classes; uso de herança; uso de final, abstract e interface, acesso a arquivos em Java

DESENVOLVIMENTO

Usando a classe Arquivo e o método public void gravar(String frase, String nomeArquivo), escreva para a classe Pessoa os métodos de acesso (get e set) além de um método para gravar os dados da pessoa. Altere o programa que leia os dados de 4 pessoas e grave em um arquivo texto.

```
public class Pessoa
{
	private String nome;
	private String endereco;
	private String telefone;
	private String email;

	public void setNome(String nome) { this.nome = nome; }
	public String getNome() { return this.nome; }
	public void setEndereco(String endereco) { this.endereco = endereco; }
	public String getEndereco() { return this.endereco; }
	public void setTelefone(String telefone) { this.telefone = telefone; }
	public String getTelefone() { return this.telefone; }
	public void setEmail(String email) { this.email = email; }
	public String getEmail() { return this.email; }

	public void grava()
	{
		String frase = this.nome+"\t"+this.endereco+"\t"
			+this.email+"\t"+this.telefone;
		Arquivo.grava(frase, "arq.txt");
	}
}

public class Exemplo
{
	public static void main(String args[])
	{
		Pessoa pessoa = new Pessoa();
		for(int i=0;i<4;i++)
		{
			System.out.println("Digite um nome");
			pessoa.setNome(Leitor.leString());
			System.out.println("Digite um endereco");
			pessoa.setEndereco(Leitor.leString());
			System.out.println("Digite um email");
			pessoa.setEmail(Leitor.leString());
			System.out.println("Digite um telefone");
			pessoa.setTelefone(Leitor.leString());
			pessoa.grava();
		}
	}
}
```

Herança

A herança é um conceito muito usado em programação orientada a objetos. O conceito é baseado em generalização-especialização. Onde temos uma classe mais genérica chamada superclasse e uma classe mais especializada chamada subclasse.

A subclasse será uma classe que terá tudo que a superclasse posui (herdará) além dos seu próprios métodos. Por exemplo: Se tivermos que construir uma classe Professor que possua nome, endereco, email, telefone e disciplina. Podemos usar a classe Pessoa já construída e fazer com que Professor herde de Pessoa.

```
public  class Professor extends Pessoa
{
       private String disciplina;
       public void setDisciplina(String disciplina) { this.disciplina = disciplina;}
pubilc void getDisciplina() { return this.disciplina; }
}
```

E usando a classe Professor

```
public class Exemplo
{
       public static void main(String args[])
       {
              Professor prof = new Professor();
              for(int i=0;i<4;i++)
              {
                     System.out.println("Digite um nome");
                     prof.setNome(Leitor.leString());
                     System.out.println("Digite um endereco");
                     prof.setEndereco(Leitor.leString());
                     System.out.println("Digite um email");
                     prof.setEmail(Leitor.leString());
                     System.out.println("Digite um telefone");
                     prof.setTelefone(Leitor.leString());
                     prof.setDisciplina(“Curso de férias”);
              }
       }
}
```

Note que Professor terá os mesmos métodos de Pessoa e mais setDisciplina() e getDisciplina(), além dos atributos de Pessoa.

Para saber se conceitualmente é possível usar herança é so perguntar
Professor **é uma** Pessoa?
Se a resposta for sim, é possível
Professor é uma Disciplina? Não, então não existe uma relação de herança

Na programação é só ver se existem muitos **atributos repetidos** em duas ou mais classes.

Escrevendo Métodos e Classes Final

Você pode declarar que sua classe é final, ou seja, que sua classe não pode ser herdada por nenhuma outra classe. Existem dois motivos principais para fazê-lo: para aumentar a segurança do sistema , prevenindo subversão do sistema e para garantir boas práticas de programação.

Para especificar que a sua classe é final:

```
final class ChessAlgorithm {
    . . .
}
```

Qualquer tentativa de criar uma subclasse de ChessAlgorithm resultará em erro de compilação:

```
Chess.java:6: Can't subclass final classes: class ChessAlgorithm
class BetterChessAlgorithm extends ChessAlgorithm {
      ^
1 error
```

Métodos Final

Criar uma classe final pode parecer muito para as suas necessidades? Você pode proteger apenas um ou mais métodos de serem sobrescritos. É só declarar o método que você quer proteger como final

```
class ChessAlgorithm {
    . . .
    final void nextMove(ChessPiece pieceMoved,
                        BoardLocation newLocation)
    {
        ...
    }
    . . .
}
```

Escrevendo classes e métodos abstract

As vezes você define uma classe que representa um conceito abstrato e que não deve ser instanciado. Da mesma forma, em Java você pode querer criar uma classe sem permitir que essa classe possa ser instanciada. Esta classe servirá apenas como base para a construção de suas subclasses.

Para declarar uma classe como abstract é só usar a palavra abastract

```
abstract class Number {
    . . .
}
```

Se você tentar criar uma instancia da classe:

```
AbstractTest.java:6: class AbstractTest is an abstract class.
It can't be instantiated.
        new AbstractTest();
        ^
1 error
```

Métodos Abstract

Métodos abstract são métodos sem implementação. Neste caso você define que a subclasse é que irá implementá-lo.

Por exemplo:

```
abstract class GraphicObject {
    int x, y;
    . . .
    void moveTo(int newX, int newY) {
        . . .
    }
    abstract void draw();
}

class Circle extends GraphicObject {
    void draw() {
        . . .
    }
}
```

Interface

Uma interface define um protocolo de comportamento que pode ser implementado por qualquer outra classe. Uma interface define um conjunto de métodos mas não implementa. A classe que implementa uma interface concorda em implementar todos os métodos definidos na interface. Uma interface pode também declarar constantes.

Declarando uma interface

```
public interface gente
{
   final String definicao=”comportamento de gente “;
   public int calculaIdade(int ano);
}
```

Cuidado! Interfaces não podem crescer!

Implementando uma interface.

```
public class Pessoa implements Gente
{
   private int anoNasc;
   public int calculaIdade(int ano)
   {
      return ano-this.anoNasc;
   }
}
```

Usando uma Interface como tipo

Podemos usar uma Interface como tipo. Para declaração de variáveis, passagem de parâmetros, etc.
Neste caso as variáveis que são atribuídas devem ser de classes que implementam esta interface.

```
public class Velho
{
   private static Pessoa mais;
   public static int diferenca(Gente novo)
   {
      return this.mais.calculaIdade(2003) – novo.calculaIdade(2003);
   }
}
```

Leitura e gravação de objetos em arquivos

Para ler e gravar um objeto de uma classe como a classe pessoa é necessário que a classe Pessoa seja declarada como implementando a interface Serializable.

Para efetuar a gravação vamos precisar de dois objetos: um objeto arq da classe FileOutputStream para estabelecer o fluxo de gravação do arquivo e um objeto out da classe ObjectOutputStream para colocar os dados do objeto no fluxo do arquivo.

Para efetuar a leitura de um objeto de um arquivo vamos precisar de um objeto arq da classe FileInputStream para estabelecer o fluxo de leitura do arquivo e um objeto in da classe ObjectInputStream para ler os dados do objeto do fluxo do arquivo.

```
import java.io.*;

public class Pessoa implements Serializable
{
```

```
    private String nome;
    private String email;

    public Pessoa() {
    }

    public Pessoa(String nome, String email)
    {
        this.nome = nome;
        this.email = email;
    }

    public String getNome()
    {
        return this.nome;
    }

    public String getEmail()
    {
     return this.email;
    }
}

import java.io.*;

public class PrincipalObj {

    public static void main(String[] args)
    {
        try
        {
            FileOutputStream arq = new FileOutputStream("arq.dat");
            ObjectOutputStream out = new ObjectOutputStream(arq);
            for(int i=0;i<5;i++)
            {
                Pessoa p = new Pessoa("pessoa"+i, "pessoa"+i+"@spet");
                out.writeObject(p);
                out.flush();
            }
            out.close();
        }
        catch(java.io.IOException exc)
        {
            System.out.println("Erro ao Gravar o arquivo");
        }

        try
        {
            FileInputStream arq = new FileInputStream("arq.dat");
            ObjectInputStream in = new ObjectInputStream(arq);
            for(int i=0;i<5;i++)
            {
                Pessoa p = (Pessoa) in.readObject();
                System.out.println(p.getNome());
                System.out.println(p.getEmail());
            }
            in.close();
        }
        catch(java.io.IOException exc2)
        {
            System.out.println("Erro ao Ler o arquivo");
        }
        catch(ClassNotFoundException cnfex)
        {
            System.out.println("Não achou a Classe");
        }
    }

}
```

ATIVIDADE

1) Construa uma classe Aluno que possua os atributos nome, endereco, email, telefone e 4 notas. Escreva os métodos get e set para os atributos da classe, um método para inserir uma nota passando o valor da nota e o índice, um método para retornar o valor de uma nota passando o índice e um método para calcular e retornar a média.
2) No exercício anterior verifique se é possível usar herança em relação à classe Pessoa. Se for possível, modifique a classe aluno da maneira adequada.
3) Digite e rode os programas para gravação de arquivo de objetos.

BIBLIOGRAFIA BÁSICA

DEITEL, H. M. e DEITEL, P. J.. *Java, como Programar.* Ed. Bookman. Porto Alegre. 2001.