UNIVERSIDADE FEDERAL DE SANTA CATARINA - Departamento de Automação e Sistemas
PROGRAMAÇÃO DE SISTEMAS AUTOMATIZADOS - 2016/2 - P2

Nome:

---

1)(2 pontos) Em um programa orientado a objetos, a classe Aluno é subclasse da classe Cliente, que por sua vez é subclasse da classe Pessoa. O método buscaDados() da classe Cadastro espera como parâmetro um objeto do tipo Cliente.

É possível invocar este método passando para ele como parâmetro uma referência a objeto instância da classe Aluno? Justifique.

É possível invocar este método passando para ele como parâmetro uma referência a objeto instância da classe Pessoa? Justifique.

2)(2 pontos) Em um programa orientado a objetos a classe Veículo é superclasse das classes Carro e Caminhão. A classe Veículo possui o método void setOdometro(double).

É possível que as classes Carro e Caminhão possuam cada uma um método void setOdometro(double), porém com código diferente (fazendo coisas diferentes)? Justifique.

É possível que a classe Caminhão ainda adicione um método void setOdometro(int), além método void setOdometro(double), mesmo que a classe Veículo não possua tal método? Justifique.

3)(2 pontos) Crie uma classe Gerente que é subclasse de Funcionário. Ela possui um atributo departamento do tipo String e uma gratificação além do salário. O método aumento() agora aplica o fator de aumento na gratificação e no salário. Um construtor permite inicializar todos os atributos. Utilize “super” no novo construtor.

```
public class Funcionario {
String nome;
String cargo;
double salario;
public Funcionario( String n, String c, double s) {
        nome = n;
        cargo = c;
        salario = s;
}
public void print() {
        System.out.println(nome + “-“ + cargo + “-“ + salario);
}
public void aumento( double fator){
        salario = salario * fator;
}
}
```

4)(1 ponto) É possível criar uma classe que herda de duas superclasses ?

5)(1 ponto) O que acontece quando a classe não herda de nenhuma classe via “extends” ?

6)(1 ponto) Qual a diferença entre public, private e o default package ?

7)(1 ponto) Como fazer para a Classe Pessoa abaixo garantir que o atributo idade sempre assuma valores entre zero e 150 ?

```
class Pessoa {

        public int idade;
        ...
}
```

UNIVERSIDADE FEDERAL DE SANTA CATARINA - Departamento de Automação e Sistemas
PROGRAMAÇÃO DE SISTEMAS AUTOMATIZADOS - 2016/2 - P3

Nome:

QUESTÕES 1, 2 e 3

```
interface Remuneravel {
     void remunera( double taxa);
}

abstract class Conta {
     double saldo = 0;

     void retira( double valor) {
          if( saldo >= valor )
               saldo -= valor;
     }

     void deposita( double valor) {
          saldo += valor;
     }

     double getSaldo() {
          return saldo;
     }

     abstract void cobraTaxa( double taxaFixa);
}


class ContaCorrente extends Conta {
     void cobraTaxa( double taxaFixa) {
          saldo -= taxaFixa;
     }
}


class ContaCorrenteEspecial extends ContaCorrente {

     void retira( double valor) {
          saldo -= valor;
     }
}
```

QUESTÕES 1, 2 e 3

1)(2 pontos) Crie a classe ContaCorrentePoupanca que é subclasse de ContaCorrente e implementa a interface Remuneravel. A lógica da remuneração pode ser qualquer.

2)(2 pontos) Crie uma nova interface "DadosImpostoRenda" com um metodo "ganhosAuferidos" e faça ContaCorrenteEspecial implementar esta nova interface.

3)(1 ponto) Crie uma subclasse da classe Conta que também é abstrata.

QUESTÕES 4, 5 e 6

4)(1 ponto) O que acontece se existirem vários objetos interessados em eventos gerados pelo "MonitorDeChuva" ?

5)(2 pontos) Crie uma classe "ChuvaAdapter" adaptadora para a interface "ChuvaListener".

6)(2 pontos) Crie uma classe "ControladorJanelas" que usa a classe "ChuvaAdapter" como classe interna, e no caso de iniciar chuva forte ele coloca mensagem na tela.

QUESTÕES 4, 5 e 6 – Primeira Parte

```
import java.util.EventObject;

class EventoChuva extends EventObject{
     private boolean chuvaForte;

     EventoChuva(Object source, boolean cf) {
          super(source);
          chuvaForte = cf;
     }

     boolean getChuvaForte() {
          return chuvaForte;
     }
}

interface ChuvaListener {
     void chuvaIniciou(EventoChuva e);
     void chuvaParou(EventoChuva e);
}

class ControladorToldo implements ChuvaListener{
     MonitorDeChuva meuMonitor;

     ControladorToldo( MonitorDeChuva monitor) {
          meuMonitor = monitor;
          meuMonitor.addChuvaListener( this );
     }

     public void chuvaIniciou(EventoChuva e) {
          System.out.println("Chuva " +
           (e.getChuvaForte()?"forte":"") + "iniciou");
     }

     public void chuvaParou(EventoChuva e) {
          System.out.println("Chuva parou");
     }
}
```

QUESTÕES 4, 5 e 6 – Segunda Parte

```
class MonitorDeChuva {
     private int ultimaMedidaChuva = 0;
     private ChuvaListener interessado = null;

     boolean addChuvaListener(ChuvaListener cl) {
          if( interessado == null ) {
               interessado = cl;
               return true;
          }
          else
               return false;
     }

     void geraEventoChuvaIniciou(EventoChuva e) {
          if (interessado != null)
               interessado.chuvaIniciou( e );
     }

     void geraEventoChuvaParou(EventoChuva e) {
          if (interessado != null)
               interessado.chuvaParou( e );
     }

     void medidaChuva(int volume){
          if( volume > 10  &&  ultimaMedidaChuva < 10 ) {
               boolean forte = volume > 50;
               EventoChuva ec = new EventoChuva( this, forte);
               geraEventoChuvaIniciou( ec );
          }
          else if( volume < 10  &&  ultimaMedidaChuva > 10 ) {
               EventoChuva ec = new EventoChuva( this, false);
               geraEventoChuvaParou( ec );
          }
     }

}
```

UNIVERSIDADE FEDERAL DE SANTA CATARINA - Departamento de Automação e Sistemas
PROGRAMAÇÃO DE SISTEMAS AUTOMATIZADOS - 2017/2 - P2

Nome:

```
interface ValeCredito {
    int getCreditos();
}

class Disciplina {
    private String nome;
    private String horario;
    private int codigo;

    Disciplina( ) {
        nome = "xxx";
        horario = "xxx";
        codigo = 0;
    }

    Disciplina( String n, String h, int c) {
        nome = n;
        horario = h;
        codigo = c;
    }

    String getString() {
        return  nome + " , " + horario + " , " + codigo;
    }

    void setCodigo( int c) {
        codigo = c;
    }
}


class DisciplinaPosGraduacao extends Disciplina {
    private char nivel = 'M';   // 'M' ou 'D'
}

class Matricula {
    private Disciplina[] lista;

    Matricula( ) {
        lista = new Disciplina[10];
    }

    void setDisciplina( int i, Disciplina di) {
        lista[i] = di;
    }
}
```

1)(2 pontos) Inclua na classe DisciplinaPosGraduacao um construtor que permite definir o nível da mesma, juntamente com todos os atributos da classe Disciplina. Não altere a classe Disciplina.

2)(2 pontos) Crie uma classe DisciplinaGraduacao que é subclasse de Disciplina. Ela possui um atributo "fase" do tipo int e inclui métodos get/set para acessar este atributo. Valores válidos para "fase" estão entre 1 e 10.

3)(2 pontos) O código abaixo está correto ou errado ? Comente.

```
Matricula m = new Matricula();
DisciplinaPosGraduacao dpg = new DisciplinaPosGraducao();
m.setDisciplina(0,dpg);
```

4)(2 pontos) As disciplinas de pós-graduação possuem código maior que 100. Altere a classe DisciplinaPosGraduacao para que isto seja verificado quando, por exemplo, as linhas abaixo são executadas:

```
DisciplinaPosGraduacao dpg = new DisciplinaPosGraducao();
dpg.setCodigo(50);     // Deve manter código antigo pois 50 é inválido
```

5)(2 pontos) Altere a classe DisciplinaPosGraduacao para que a mesma implemente a interface ValeCredito. Por default cada disciplina vale 2 créditos.

UNIVERSIDADE FEDERAL DE SANTA CATARINA - Departamento de Automação e Sistemas
PROGRAMAÇÃO DE SISTEMAS AUTOMATIZADOS - 2017/2 - P3

Nome:

QUESTÕES 1 e 2

```
class Conta {
     static int novoNumero = 1000;
     int numero;
     double saldo = 0;

     void retira( double valor) {
          if( saldo >= valor )
               saldo -= valor;
     }

     void deposita( double valor) {
          saldo += valor;
     }

     double getSaldo() {
          return saldo;
     }
}


class AcessoGerente {

     Conta criaConta( ) {
          System.out.println("num contas " + Conta.getQuantas());
          Conta nova = new Conta(12.34);
          return nova;
     }
}
```

1) (2 pontos) Crie um construtor para **Conta** que recebe o saldo inicial como parâmetro e define um novo número único para a conta criada.

2) (2 pontos) Crie um método **getQuantas( )** para **Conta** que informa quantas contas foram criadas até o momento, e funcione com o código da classe **AcessoGerente** como este está escrito.

QUESTÕES 3, 4 e 5 – Primeira Parte

```
import java.util.EventObject;

class EventoOnibus extends EventObject{
     private String placa;

     EventoOnibus(Object source, String p) {
          super(source);
          placa = p;
     }

     String getPlaca() {
          return placa;
     }
}

interface OnibusListener {
     void onibusChegou(EventoOnibus e);
     void onibusPartiu(EventoOnibus e);
     void onibusPifou(EventoOnibus e);
}

class PainelAvisos implements OnibusListener{
     EstacaoRodoviaria minhaEstacao;

     PainelAvisos(EstacaoRodoviaria estacao) {
          minhaEstacao = estacao;
          minhaEstacao.addOnibusListener( this );
     }

     public void onibusChegou(EventoOnibus e) {
          System.out.println("Onibus chegou");
     }

     public void onibusPartiu(EventoOnibus e) {
          System.out.println("Onibus partiu");
     }

     public void onibusPifou(EventoOnibus e) {
          System.out.println("Onibus pifou");
     }
}
```

QUESTÕES 3, 4 e 5 – Segunda Parte

```
class EstacaoRodoviaria {
     private String placa;
     private OnibusListener interessado = null;

     boolean addOnibusListener(OnibusListener ol) {
          if( interessado == null ) {
               interessado = ol;
               return true;
          }
          else
               return false;
     }

     void geraEventoOnibusChegou(EventoOnibus e) {
          if (interessado != null)
               interessado.onibusChegou( e );
     }

     void geraEventoOnibusPartiu(EventoOnibus e) {
          if (interessado != null)
               interessado.onibusPartiu( e );
     }

     void informa( String placa, int situacao){
          if( situacao == 1 ) {
               EventoOnibus eo = new EventoOnibus( this, placa);
               geraEventoOnibusChegou( eo );
          }
          else {
               EventoOnibus eo = new EventoOnibus( this, placa);
               geraEventoOnibusPartiu( eo );
          }
     }

}
```

QUESTÕES 4 e 5

3)(2 pontos) Crie uma classe **OnibusAdapter** adaptadora para a interface **OnibusListener**.

4)(3 pontos) Crie uma classe **ListaPifados** que usa uma subclasse de **OnibusAdapter** como classe interna e apenas no caso do onibus pifar ela coloca uma mensagem na tela. **ListaPifados** se adiciona como escutadora de **EstacaoRodoviaria** no código do construtor de **ListaPifados**.

5)(1 ponto) É possível fazer new em **OnibusListener** ? Explique.