UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
SISTEMAS DE INFORMAÇÃO - CÂMPUS DE COXIM
FUNDAMENTOS EM ORIENTAÇÃO A OBJETOS

Data final de entrega 16/09/2014, até às 23h59min.
Enviar o arquivo de respostas em formato PDF e o arquivo.zip com códigos fontes para o e-mail mota.fernandomaia@gmail.com, insira no assunto do e-mail "[Lista 2 – Fundamentos em Orientação a Objetos]".

Lista de Exercícios – 02

1. Descreva com suas palavras a principal característica de um objeto, método ou argumento de função polimórfico.

Um objeto polimórfico, tal como uma variável ou argumento de função, que pode armazenar valores de tipos diferentes durante o curso de execução de um programa.

2. Em relação ao código a seguir:

```
public class Janela {
    private Double largura;
    private Double altura;
    private String cor;
        .
        .
        .

}

public class JanelaQuarto extends Janela{
    private boolean persiana;
        private int numero;
        .
        .
        .

        public Double getArea(){
        return super.getAltura()*super.getLargura();
    }

    public Double getArea(Double areaMoldura){
        return super.getAltura()*super.getLargura()+areaMoldura;
```

```
    }
}


public class Main {
    public static void main(String args[]){
        JanelaQuarto janela = new JanelaQuarto();
        janela.getArea();
        janela.getArea(1.45);
    }
}
```

a. Descreva qual é o tipo de polimorfismo implementado.

Polimorfismo de carga, no método getArea.

b. É possível aplicar haver coerção neste exemplo? Se sim, por quê?

Sim, no mesmo método getArea na sua segunda instância poderia ser passado um valor inteiro como argumento, este valor inteiro seria coergido para Double.

3. Em relação ao código a seguir:

```
public class Janela {
    private Double largura;
    private Double altura;
    private String cor;


        public Janela(){
                System.out.println("Olá, eu sou o construtor da classe Janela!");
                saida();
        }


    public void saida(){
        System.out.println("Para sair do quarto, utilize a porta frontal à janela!");
    }
}
```

```
public class JanelaQuarto extends Janela{

    private boolean persiana;

        private int numero;


        public JanelaQuarto(){

                System.out.println("Olá, eu sou o construtor da classe
JanelaQuarto!");

        }


        public void saida(){

        System.out.println("Para sair do quarto, utilize a porta lateral à janela!");

    }

}


public class Main {

    public static void main(String args[]){

                Janela janela = new Janela();

        System.out.println("");

        Janela janela2 = new JanelaQuarto();

    }

}
```

Sua Saída é:

```
Olá, eu sou o construtor da classe Janela!
Para sair do quarto, utilize a porta frontal à janela!

Olá, eu sou o construtor da classe Janela!
Para sair do quarto, utilize a porta lateral à janela!
Olá, eu sou o construtor da classe JanelaQuarto!
```

Descreva qual é a relação da ordem das mensagens de saída, em relação a polimorfismo e construtores.

Na primeira parte do código da Main, que é a instanciação do objeto *janela* tanto o construtor da classe *Janela* é executado como o método *saída* de Janela.
Na segunda parte do código da Main, que é a instanciação do objeto *janela2* primeiro se executa o método da classe *Janela* (classe-pai), depois dentro do construtor de Janela é executado o método saída de *JanelaQuarto*, por fim é executado o construtor de *JanelaQuarto*, este workflow de execução caracteriza a relação entre polimorfismo e construtores.

4. Dentro da herança de classes é onde o polimorfismo de objetos é mais claro, desta forma baseado no código a seguir, responda:

   a. Em qual parte do código o polimorfismo fica explícito?

```
Janela janela2 = new JanelaQuarto();
```

   b. Em qual momento o polimorfismo ocorre?

```
Janela janela2 = new JanelaQuarto();
```

O polimorfismo ocorre na instanciação da subclasse JanelaQuarto que extende Janela, pois o objeto janela2 mesmo declarado do tipo Janela quando executado na verdade o objeto criado será do tipo JanelaQuarto.

```
public class Janela {
    private Double largura;
    private Double altura;
    private String cor;
  .
  .
  .

  private void fechar(){
          //codigo omitido
  }
}

public class JanelaQuarto extends Janela{
    private boolean persiana;
  private int numero;
  .
  .
  .
  private void fechar(boolean persiana){
          if(persiana)
                  fecharPersiana();
```

```
            //codigo omitido
    }

    private void fecharPersiana(){
            //codigo omitido
    }
}

public class Main {
    public static void main(String args[]){
        Janela janela1 = new Janela();
        Janela janela2 = new JanelaQuarto();

            janela1.fechar();
            janela2.fechar();
    }
}
```

5. Defina encapsulamento com suas palavras.

   O encapsulamento é o processo de proteger os membros de uma classe (atributos e métodos), permitindo que somente os membros necessários (públicos) sejam acessados pelos usuários da classe. Desta forma, em manutenções futuras este atributos que se encontram "protegidos" poderão sofrer alteração sem que os outros objetos ou partes de códigos que os utilizem sofram alterações, pois seu acesso esta encapsulado nos métodos get's e set's.

6. Aponte e corrija o erros da classe *AponteErros*:

```
public class AponteErros {
    int id;
    String nome;
    Double velocidade;

    public aponteErros(int identificador, String nome){
        id=identificador;
        this.nome=nome;
```

```
    }

    private void setVelocidade(String v){
        velocidade=v;
    }

    private Double getVelocidade(){
        return velocidade;
    }
}
```

Resposta:

```
public class AponteErros {
    private int id;
    private String nome;
    private Double velocidade;

    public AponteErros(int identificador, String nome){
        id=identificador;
        this.nome=nome;
    }

    public  void setVelocidade(Double v){
        velocidade=v;
    }

    public Double getVelocidade(){
        return velocidade;
    }
}
```

7. Descreva com suas palavras e através de exemplo como o encapsulamento pode facilitar tarefas de manutenção de software.

   Em manutenções futuras os atributos que se encontram “protegidos” poderão sofrer alteração sem que os outros objetos ou partes de

códigos que os utilizem sofram alterações, pois seu acesso esta encapsulado nos métodos get's e set's.

8. Decomponha o método *setLimpeza* da classe *Limpar*, de forma que reduza o acoplamento de dados existente.

```
public class Limpar {
    public void setLimpeza(String locais[]){
        int i;
        for(i=0;i < locais.length; i++){
            if(locais[i]=="quarto"){
                //codigo omitido
            }else if(locais[i]=="sala"){
                //codigo omitido
            }else if(locais[i]=="cozinha"){
                //codigo omitido
            }else{
                System.out.println("Informe um local válido!");
            }
        }
    }
}
```

Resposta:

```
public class Limpar {
        public void setLimpezaQuarto();
        public void setLimpezaSala();
        public void setLimpezaCozinha();
}
```

9. Descreva as diferenças entre superclasse, classe abstrata e interface, utilize de exemplos de códigos para apresentar sua solução.

Superclasse são classes que são herdadas por outras classes aproveitando o seu comportamento (atributos e métodos). Já nas classes abstratas elas podem herdar de uma superclasse e definir assinaturas de métodos (métodos abstratos), não podem ser instanciadas, podem implementar funções e conter atributos, por fim, interfaces também podem definir assinaturas de métodos, mas não podem implementar funções e definir atributos, exceto atributos finais e estáticos.

10. Programe as classes do diagrama de classes a seguir de acordo com as regras de POO.