Universidade Federal de Campina Grande
Centro de Engenharia Eletrica e Informática
Departamento de Sistemas e Computação
Graduação em Ciência da Computação

Exercícios sobre Conceitos Básicos de Java

Objetivo: Relembrar os conceitos de Java. Consulte a documentação da classe java.util.Date, quando necessário. Procure usar como fonte de estudo suas aulas do semestre passado (P2 e LP2). Este exerccício não precisa ser entregue. Entretanto, é importante que voce resolva com atenção pois diversos conceitos e práticas de programação serão essenciais no decorrer da disciplina. Dessa forma, se voce tiver qualquer dúvida, procure o professor ou monitor. Utilize o horário da aula para resolver os exercícios.

1. (SEM ECLIPSE) Qual a diferença entre overload (sobrecarga) e override (sobreposição)? EXEMPLIFIQUE.

R = overload (ou polimorfismo ad hoc) é a possibilidade de se ter mais de um método (ou construtor) com mesmo nome mas com parâmetros diferentes. Isso é possível porque os métodos se diferenciam pelo nome e pela lista de parâmetros, onde importa a ordem, o tipo e a quantidade. Override é a redefinoção de método. Envolve uma superclasse e uma subclasse. Na redefinicao o método na subclasse tem que ter a mesma assinatura do método da superclasse e deve preservar a semântica e ter, pelo menos, a mesma visibilidade (private, public, protected, *default*) do método da classe pai. Exemplos:

```
//exemplo de overload
public void m(){ ... }
public void m(int x){ ... }
public void m(String x){ ... }
```

```
//exemplo de override
public class A{
        public void m(String x, int y){ ... }
}

public class B extends A{
        public void m(String x, int y){ ... }
}
```

2. (SEM ECLIPSE) Quando usar o override? EXEMPLIFIQUE.

R = o override serve para os casos em que herança simplesmente nao resolve, pois algum metodo herdado precisa ser adaptado para a subclasse. Dessa forma, herda-se e redefine-se o método. Na redefinicao, é preciso preservar observar alguns itens:

- Preservacao ou aumento da visibilidade – um metodo redefinido nao pode ter sua visibilidade restringida. Se é protected na classe pai, deve ser protected ou public quando redefinido numa subclasse.
- Preservacao da semantica – um metodo nao pode fazer algo que nao era esperado (diferente) do que foi estabelecido na classe pai. Por exemplo, imagine uma conta bancaria como mostrada abaixo. Imagine um outro tipo de conta (ContaBonificada) em que o metodo de creditar alem de creditar no saldo tambem acumula um percentual do valor creditado em um bonus.

```
public class Conta{
  private String numero;
  private double saldo;
  public void creditar(double v){
    saldo = saldo + v;
  }
```

```
 public class ContaBonificada extends Conta{
  private double bonus;
  public void creditar(double v){
    super.creditar(v);
    bonus = bonus + v*0.02;
  }
  public void renderBonus(){
    super.creditar(bonus);
    bonus = 0;
  }
}
```

Usando herança, o metodo creditar precisa ser adaptado para acumular o bonus. Assim herança apenas nao é suficiente. Note que a visibilidade e a semantica do cretitar de Conta foram preservados em ContaBonificada, pois a mudança de saldo permanece a mesma. Apenas o bonus é acumulado (nao rende automaticamente) em ContaBonificada.

3. (SEM ECLIPSE) O que são classes abstratas e quais situações requerem seu uso? EXEMPLIFIQUE.
   R = classes abstratas ja sao um recurso a mais que resolvem casos em que herança e herança com redefinicao nao resolvem. Classes abstratas sao um recurso importante para estensibilidade e reusabilidade. Nao podem ser instanciadas e agrupam categorias que se diferenciam em alguma funcionalidade. Vamos ver um exemplo.

```
public class Conta{
  private String numero;
  private double saldo;
  public void debitar(double v){
    saldo = saldo - v;
  }
```

Imagine novamente o exemplo da classe conta bancaria com o método de debitar. Imagine agora que surgiu um outro tipo de conta, chamada ContaImposto, que cobra automaticamente um imposto de 0,2% do valor movimentado nas operacoes de debitar.
Assim, a operacao de debitar ao inves de retirar determinado valor do saldo, retira mais do que o esperado. Neste caso, usar heranca com redefinicao nao é aconselhavel porque o debitar de ContaImposto produzirá um saldo diferente do esperado em Conta. A solucao é entao usar uma classe contendo tudo em comum a todas as contas e o método debitar em aberto (sem semantica) para que Conta e ContaImposto herdem dela e implementem, o debitar de seu jeito.

```
public abstract class ContaAbstrata{
  private String numero;
  private double saldo;
  public abstract void debitar(double v);
}
```

```
public class Conta extends ContaAbstrata{

  public void debitar(double v){
    this.setSaldo(this.getSaldo()– v);
  }
```

```
public class ContaImposto extends ContaAbstrata{
 public void debitar(double v){
   double valorComImposto = v + v*0.002;
   this.setSaldo(this.getSaldo()– valorComImposto);
 }
}
```

Note que Conta e ContaImposto tem todo o restante em comum, exceto o metodo debitar. Eles tem semantica incompativeis que nao podem ser resolvidas com herança e redefinicao. Por isso se utiliza classe abstrata. Assim, a ContaAbstrata está agrupando de alguma forma os tipos Conta e CotnaImposto, que tem diferenca semantica em uma funcionalidade.

Um outro exemplo de uso de classe abstrata é quando se deseja agrupar tipos numa superclasse, mas que, em tempo de execucao, nao se é permitido criar objetos da superclasse. Dessa forma, a superclasse é declarada como abstrata (nao poderá ser instanciada) e existe apenas como um recurso de modelagem para aumentar a reusabilidade e estruturacao.

4. (SEM ECLIPSE) O que são *casts*? Para que servem *casts* e *instanceof*? EXEMPLIFIQUE.
R = casts sao conversoes dinamicas de tipos e servem para eliminar erros de compilacao. Sao necessarios quando se deseja atribuir um tipo primitivo de maior precisao para um tipo de menor precisao. Ou entao quando se deseja atribuir um tipo (referenciado) mais generico para um tipo mais especifico, ou acessar servicos de tipos mais especificos via uma variavel que é do tipo mais generico. Casts podem originar potenciais erros de tempo de execucao. Exemplos:

```
int x = 349854;
short y = (int) x;
```

```
ContaAbstrata c = ….;
Conta c2 = (Conta) c;
```

```
ContaAbstrata c = ….;
ContaBonificada c2 = (ContaBonificada) c;
((ContaBonificada)c2).renderBonus();
```

instanceof é um operador de Java para confirmar, em tempo de execucao, o tipo de uma determinada referencia. A melhor forma de entender o instanceof é traduzindo ele para a pergunta: é do tipo? O instanceof serve para retirar potenciais erros que acontecem apenas em tempo de execucao originados por cast. Exemplo:

```
ContaAbstrata c = new ContaImposto();
c instanceof Object //retorna true
c instanteof ContaAbstrata //retorna true
c instanteof ContaImposto //retorna true
c instanceof Conta //retorna false
```

Note que o instanceof observa o tipo do objeto para o qual c aponta. Como é objeto do tipo ContaImposto, todos os tres instanceof retornam true. O ultimo retorna false. J ano exemplo seguinte, o instanceof é para evitar o possivel erro em tempo de execucao que o cast nao resolve:

```
ContaAbstrata c =new ContaImposto();
ContaBonificada c2 = (ContaBonificada) c;
if (c2 instanceof ContaBonificada){
  ((ContaBonificada)c2).renderBonus();
}
```

5. (SEM ECLIPSE) No que interfaces diferem de classes abstratas? Que situações requerem o uso de interface e não de classes abstratas?

   R = interfaces sao um recurso mais abstrato que classes abstratas. Relembre que as classes abstratas servem para agrupar tipos que tem alguma coisa (código) em comum mas que se diferenciam em algumas funcionalidades. Ja no caso de interfaces, os tipos agrupados tem apenas os nomes das funcionalidades em comum, mas as implementacoes sao completamente diferentes. Num cenario onde uma classe precisa apenas utilizar apenas os servicos de outra classe (sem se preocupar com sua estrutura interna) nao se deve usar classe abstrata e sim interface.

6. (SEM ECLIPSE) Considerando os trechos de código-fonte abaixo responda os itens de I até V.

```
public interface I1{
  public abstract void m3(String x);
}
public abstract class A implements I1{
  public abstract void m1(int x);
  public abstract void m2(double x);
}
public abstract class B extends A{
  public abstract void m1();
  public void m2(double x){
   System.out.println("m2 executado");
  }
}
```

```
public interface I2{
  public abstract void m4();
}
public abstract class C extends A
       implements I2{
  public void m1(int x){
   System.out.println("m1 executado");
  }
}
public class E extends C {
}
public class D extends B {
}
```

I) Quais métodos (com nomes e parâmetros) deverão ser implementados pela classe E?

R = m2(double), m3(String) e m4()

II) Quais métodos (com nomes e parâmetros) deverão ser implementados pela classe D?

R = m1(), m1(int) e m3(String)

III) O comando: `I1 x = new E();` vai dar erro de compilação? Por que?

R = nao vai dar erro porque x é do tipo I1 e aceita um I1 ou qualquer subtipo dele. Pelo principio da substituicao, onde I1 é aceito, E pode ser usado (E é subtipo de I1).

IV) Reescreva o trecho abaixo retirando os possíveis erros de *compilação* utilizando casts.

```
I1 x = new D();
x.m3("5");
((A)x).m2(2.0); //ou cast para qualquer subtipo de A
E y = (E)x;
I2 k = (C)x; //ou I2 k = (E)x;
```

V) Considerando o trecho abaixo, qual o resultado produzido pelos comandos de impressão na tela?

```
I1 x = new E();
System.out.println(x instanceof B); false
System.out.println(x instanceof C); true
System.out.println(x instanceof A); true
System.out.println(x instanceof I1); true
System.out.println(x instanceof I2); true
```

7. (COM ECLIPSE) Agora refaça a questão anterior usando o eclipse.
   R: É só criar as classes e interfaces dos dois primeiros quadros. O Eclipse já mostrará erros porque alguns métodos precisam ser implementados nas classes concretas. Em seguida, em qualquer classe concreta implementada, crie um método main que contém os trechos de código dos dois ultimos quadros (IV e V) e observar os erros de compilação (a serem elimi9nados com casts) e depois executar para observar a saída do programa.

8. Observe o trecho de código abaixo e, antes de escreve-lo no Eclipse, tente responder aos questionamentos em comentário e justificar o porque. Depois escreva-o no Eclipse e tente descobrir se voce acertou todas as respostas.

```
Date d1 = new Date();
Date d2 = d1;
d2.setDate(5);
System.out.println(d2.getDate()); //que valor vai ser impresso?
System.out.println(d1.getDate()); //que valor vai ser impresso?
d2 = null;
System.out.println(d1.getDate()); //que valor vai ser impresso?
System.out.println(d2.getDate()); //o que acontece?
```

   R: aqui tipicamente encontra-se um problema de aliasing, onde mais de uma variável referencia o mesmo objeto. Devido a semantica de passagem de mensagem em chamada de métodos Java, as mudanças efetuadas em um objeto via uma variável (d2) ficam visíveis quando se acessa o mesmo objeto usando outra variável (d1). Assim o resultado impresso do código acima será:

```
Date d1 = new Date(); //cria uma data correspondendo a data atual
Date d2 = d1; //d2 passa também a referenciar o objeto referenciado por d1
d2.setDate(5); //muda a data (dia) do objeto usando d2
System.out.println(d2.getDate()); // imprime 5
System.out.println(d1.getDate()); //imprime 5 (o objeto foi modificado via d2)
d2 = null; //apenas d2 foi modificado
System.out.println(d1.getDate());//imprime 5 (d1 ainda aponta para o objeto)
System.out.println(d2.getDate()); //NullPointerException
```

9. (COM ECLIPSE) Conceitos matemáticos como funções, somatório e produtório, por exemplo, podem ser facilmente implementados em Java. A idéia é que cada estrutura matemática seja representada por uma construção (classe, metodo, comando, etc.) da linguagem Java. Implemente a seguinte função em Java.

$$f(n,m) = \sum_{i=1}^{n} i^2 + \prod_{j=1}^{m/2} j$$

   onde

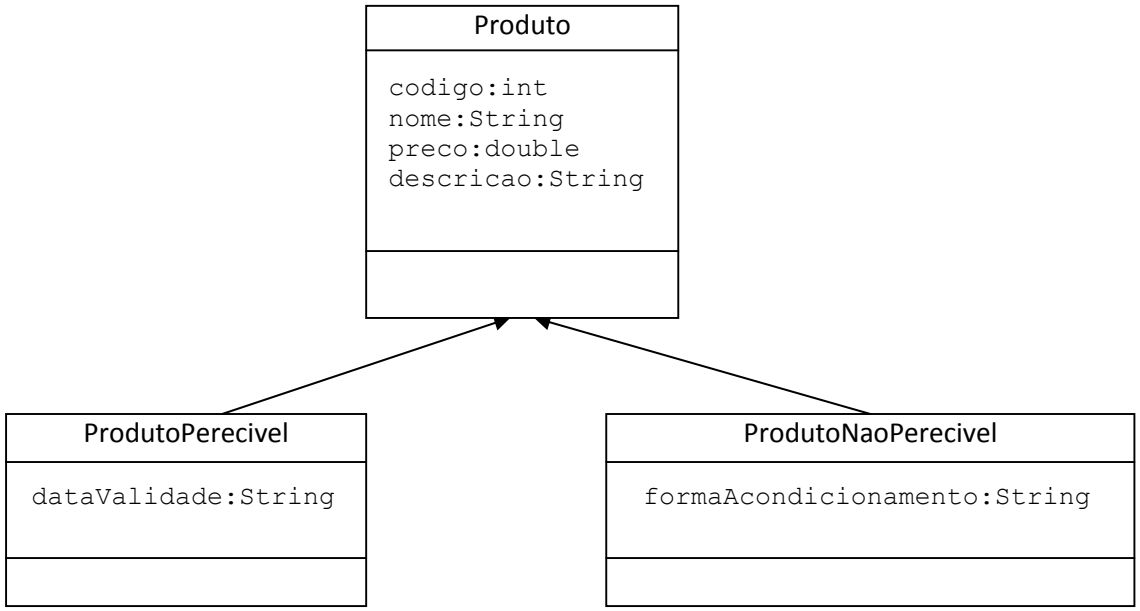
$$\sum_{i=1}^{k} e = e + e + \ldots + e \ (k \ vezes) \qquad \prod_{i=1}^{k} e = e * e * \ldots * e (k \ vezes)$$

R = o recurso sintatico de Java mais adequado para implementar funcoes é metodo. Somatórios e produtórios precisam de um acumulador e podem ser implementados com laços. Como os limites do somatorio/produtorio sao conhecidos estaticamente, usa-se o laço for. Assim, o metodo fica:

```
public int f(int n, int m){
  int somatorio= 0;
  for(int i=1, i<=n; i++){
    somatorio = somatorio + i*i;
  }
  int produtorio = 1;
  for(int j=1, j<=m/2; j++){
    produtorio = produtorio*j;
  }

  return somatorio + produtorio;
}
```

10. (COM ECLIPSE) Imagine que seu sistema vai lidar com dois tipos de produtos: produto perecível e não-perecível. Um produto perecível possui uma data de validade (pode ser tratada como uma String). O produto não-perecível possui uma informação extra sobre o acondicionamento do mesmo (também tratada como uma String). A árvore de herança abaixo mostra a estrutura das classes com herança. Implemente as classes ProdutoPerecivel e ProdutoNaoPerecivel utilizando herança



R: é só criar o código das classes no Eclipse e depois fazer a associação entre as busclasses e superclasse. Abaixo mostramos apenas o código de uma das subclasses.

```
public class ProdutoPerecivel extends Produto{
  private String dataValidade;

  public ProdutoPerecivel(int codigo, String nome,
                          double preco, String descricao,
                          String dataValidade){
    super(codigo,nome,preco,descricao);
    this.dataValidade = dataValidade;
  }
  ...
}
```