Universidade Federal do Ceará
Departamento de Computação
Curso de Ciência da Computação

Disciplina	Técnicas de Programação I - CK0112	**Semestre**	2016/1
Professor	Lincoln Souza Rocha		
Lista de Revisão I			

1) A programação orientada a objetos é baseada em diversos conceitos, tais como encapsulamento, herança, polimorfismo e abstração. Com relação a esses conceitos, marque **V** para verdadeiro e **F** para falso (explique por que é falso):
   a) (__) A herança é um mecanismo que permite que uma classe herde todo o comportamento e os atributos de outra classe. Em Java, pode-se implementar tanto a herança única quanto a herança múltipla.
   b) (__) O conceito de encapsulamento é alcançado por meio da definição de visibilidade dos atributos e métodos da classe. Em Java, a visibilidade `private` indica que o atributo ou método é visível somente dentro daquela classe.
   c) (__) O polimorfismo é o uso de um mesmo nome para identificar diferentes implementações dos métodos. Seu uso é comum na definição de construtores, em que os mesmos podem ser implementados em diferentes versões para as diferentes formas de se instanciar a classe.

2) Associe a primeira coluna com a segunda:

A.	Corretude	(__) É a facilidade de se adaptar produtos de software à mudanças em sua especificação.
B.	Extensibilidade	(__) É a habilidade do produto de software reagir apropriadamente face a condições anormais.
C.	Reusabilidade	(__) É a capacidade de elementos de software servirem para a construção outras aplicações.
D.	Robustez	(__) É a habilidade do software executar sua tarefa, exatamente como definido em sua especificação.

3) Sobre os conceitos de orientação a objetos marque **V** para verdadeiro e **F** para falso (explique por que é falso).
   a) (__) A sobrecarga permite alterar o comportamento de um método na classe filha mantendo a mesma assinatura definida na classe mãe.
   b) (__) Em OO o comportamento de um objeto é descrito por um conjunto de ações pré-definidas (métodos) através das quais o objeto responderá a demanda de processamento por parte de outros objetos.
   c) (__) Em OO o estado de um objeto é descrito pelo conjunto de valores assumidos pelos seus métodos em um dado instante de tempo.
   d) (__) A sobrecarga é o mecanismo que permite uma classe herdar atributos e métodos de outra classe.
   e) (__) A classe `A` está no nível mais baixo de uma hierarquia de classes. Dessa forma, ela pode ser tipo dinâmico de qualquer objeto que possua tipo estático pertencente a esta mesma hierarquia de classes.

4) Assinale a alternativa que apresenta a sequencia de palavras que torna correta a seguinte sentença:
   Na programação orientada a objetos, a unidade de modularização é denominada de _____. A _____ funciona como uma descrição formal (*template*) para a criação de _____. A _____ é o processo que permite criar _____ a partir de uma classe utilizando _____.
   a) (__) objeto, classe, objetos, sobrecarga, objetos, polimorfismo.
   b) (__) objeto, classe, subclasses, herança, subclasses, `extends`.
   c) (__) classe, herança, subclasses, instanciação, objetos, métodos.
   d) (__) classe, superclasses, subclasses, instanciação, subclasses, herança.
   e) (__) classe, classe, objetos, instanciação, objetos, construtores.

5) Em relação a **tipo estático** e **tipo dinâmico** e observando a Figura 1, avalie as afirmações marcando **V** para verdadeiro e **F** para falso (explique por que é falso).

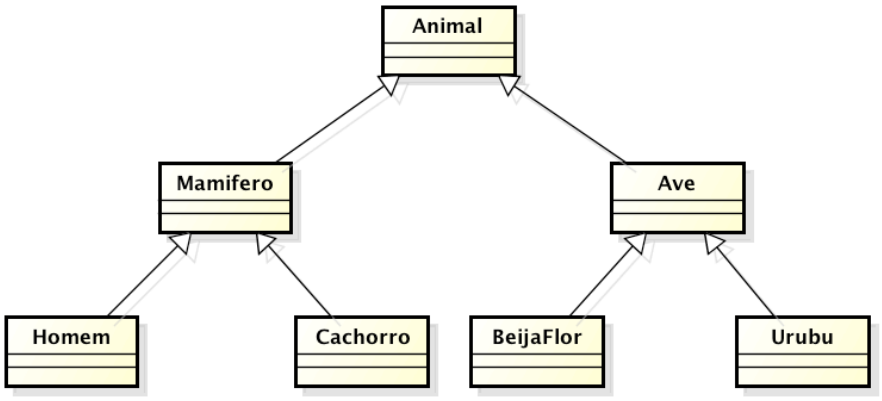

Figura 1 – Diagrama de classes de animais.

a) (__) É possível atribuir um tipo dinâmico `Homem` à variável `u` do tipo `Urubu`.
b) (__) Uma variável `h` do tipo `Homem` pode assumir um tipo dinâmico `Animal`.
c) (__) É possível atribuir um tipo dinâmico `Ave` à variável `u` do tipo `Urubu`.
d) (__) Uma variável `a` do tipo `Animal` pode assumir tipos dinâmicos `Cachorro` e `Ave`.
e) (__) O tipo estático da variável `a` é `Animal`. Logo, essa variável pode possuir como tipo dinâmico qualquer tipo da hierarquia classes.

6) Com base na descrição abaixo das classes `A` e `B`, escritas em Java, marque **V** para verdadeiro e **F** para falso (explique por que é falso).

```
public class A {
   public void fazAlgo() {
      System.out.println("A.fazAlgo()");
   }
   public void fazAlgo(String algo) {
      System.out.println("A.fazAlgo(String)");
   }
}
```

```
public class B extends A {
   public void fazAlgo() {
      System.out.println("B.fazAlgo()");
   }
}
```

a) (__) Devido ao conceito de polimorfismo, a instanciação `B b = new A()` é possível.
b) (__) A classe `B` herda da classe `A`, tornando o método `fazAlgo()` sobrecarregado.
c) (__) Na classe `A` temos um exemplo de sobrescrita de método onde o método `fazAlgo(String algo)` sobrescreve o método `fazAlgo()`.
d) (__) Observando o código Java das duas classes, não é possível encontrar uma relação de sobrescrita de método entre as classes `A` e `B`.
e) (__) Devido ao conceito de ligação tardia (*late binding*), logo após uma instanciação da forma `A a = new B()`, a chamada de método `a.fazAlgo()` resultará na impressão do texto "`B.fazAlgo()`".

7) A `Fila` é um tipo abstrato de dados bem conhecido. Basicamente, ela possui uma disciplina de acesso que permite adicionar um elemento ao fim da fila (`adicionar()`), remover um elemento apenas do início da fila (`remover()`), consultar o primeiro (`primeiro()`) e o último (`ultimo()`) elemento da fila. Considerando a classe `Fila` descrita abaixo, assinale **V** para verdadeiro e **F** para falso os itens a seguir (explique por que é falso).

```
01  public class Fila {
02
03     String[] fila;
04     private int inicio;
05     private int fim;
06
07     public Fila() {
08        fila = new String[100];
09        inicio = -1;
10        fim = inicio;
11     }
12     public Fila(int tamanho) {
13        fila = new String[tamanho];
14        inicio = -1;
15        fim = inicio;
16     }
17
18     public boolean adicionar(String elemento) {
19        if(inicio < 0) {
20           fila[++inicio] = elemento;
21           fim = inicio;
22        } else {
23           fila[++fim] = elemento;
24        }
25        return true;
26     }
27
28     public String remover() {
29        return fila[inicio++];
30     }
31
32     public String primeiro() {
33        return fila[inicio];
34     }
35
36     public String ultimo() {
37       return fila[fim];
38     }
39  }
```

a) (__) Sempre que for adicionado o centésimo elemento na fila, ela ficará completamente cheia.
b) (__) O atributo `fila` pode ser acessado diretamente por qualquer outra classe Java, independente de qual pacote pertença.
c) (__) Não é possível criar instancias da classe `Fila`.
d) (__) Existe apenas um método sobrecarregado na classe `Fila`.
e) (__) `primeiro()` e `ultimo()` são métodos que alteram o estado da fila.

8) Utilizando os conceitos de **herança**, **subtipo**, **polimorfismo** de **tipo/classe**, e observando a Figura 1, responda se os trechos de código abaixo estão corretos (do ponto de vista do compilador Java), marcando **V**, para os corretos, e **F**, para os não corretos (explique por que é falso).

a) (__) Trecho 1

```
01 Animal a = new Animal();
02 Urubu u = new Urubu();
03 a = u;
```

b) (__) Trecho 2

```
01 Mamifero m = new Cachorro();
02 Cachorro c = (Cachorro) m;
03 BeijaFlor b = new Urubu();
04 Urubu u = b;
```

c) (__) Trecho 3

```
01 Animal a = new Mamifero();
02 Homem h = new Homem();
03 a = h;
```

d) (__) Trecho 4

```
01 Animal a = new Homem();
02 Homem h = new Homem();
03 h = a;
```

e) (__) Trecho 5

```
01 Animal a = new Ave();
02 Mamifero c = new Cachorro();
03 a = c;
```

9) Utilizando os conceitos de **herança**, **sobrescrita**, **polimorfismo de método** e a hierarquia de classes da Figura 2, assinale **V**, caso o trecho de código esteja correto (do ponto de vista do compilador Java) e **F** para o caso contrário (explique por que é falso).

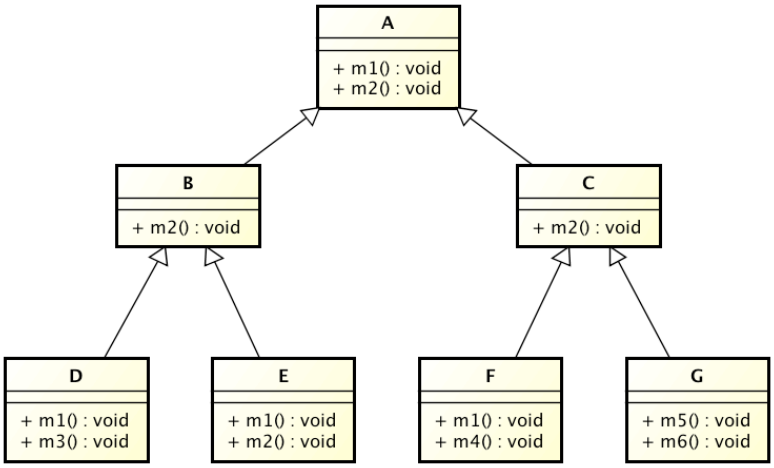

Figura 2 – Hierarquia de classes.

a) (__) Trecho 1

```
01 G g = new G();
02 g.m2();
03 g.m1();
```

b) (__) Trecho 2

```
01 A a = new A();
02 a.m1();
03 a.m3();
```

c) (__) Trecho 3

```
01 E e = new B();
02 e.m1();
03 e.m3();
```

d) (__) Trecho 4

```
01 C c = new G();
02 ((G) c).m5();
03 ((G) c).m6();
```

e) (__) Trecho 5

```
01 B b = new D();
02 b.m1();
03 b.m3();
```

10) Levando em conta a Figura 2, e os fragmentos de código abaixo, assinale **V** para verdadeiro e **F** para falso (explique por que é falso).

```
public abstract class A {
   public void m1() {
      System.out.println("A.m1()");
   }
   public abstract void m2();
}
```

```
public class B extends A {
   public void m2() {
      System.out.println("B.m2()");
   }
}
```

```
public class C extends A {
   public void m2() {
      System.out.println("C.m2()");
   }
}
```

```
public class D extends B {
   public void m1() {
      System.out.println("D.m1()");
   }
}
```

```
public class E extends B {
   public void m1() {
      System.out.println("E.m1()");
   }
   public void m2() {
      System.out.println("E.m2()");
   }
}
```

```
public class F extends C {
   public void m1() {
      System.out.println("F.m1()");
   }
}
```

```
01 public class Main {
02
03    public static void main(String[] args) {
04       A a = new B();
05       a.m1();
06       B b = new D();
07       b.m1();
08       b.m2();
09       ((D) b).m1();
10       b = new E();
11       b.m1();
12       b.m2();
13       G g = new G();
14       g.m1();
15       g.m2();
16       C c = new F();
17       c.m2();
18       c.m1();
19    }
20 }
```

a) (__) Na linha `18` da classe `Main` vai ser impresso "`F.m1()`".
b) (__) Na linha `09` da classe `Main` vai ser impresso "`D.m1()`".
c) (__) Na linha `07` da classe `Main` vai ser impresso "`B.m1()`".
d) (__) Na linha `12` da classe `Main` vai ser impresso "`E.m2()`".
e) (__) Na linha `11` da classe `Main` vai ser impresso "`B.m1()`".