

# Linguagem JAVA

Curso de Exercícios  
CESGRANRIO



PROF. VINICIUS REIS - [professor.vreis@gmail.com](mailto:professor.vreis@gmail.com)

# Apresentação Pessoal

## Vinicius Reis

- Formação Acadêmica
  - Graduação em Tecnologia de Sistemas de Informação – Faculdade São José
  - Pós Graduação em Análise e Projeto de Sistemas – Universidade Gama Filho
- Atuação
  - Analista de Sistemas – SERPRO
- Aprovações
  - SERPRO, DATAPREV, MPU, INPI, Petrobras Distribuidora, entre outras.

# Motivação

- **PETROBRAS 2012**

- 50 questões específicas
- 8 questões relacionadas à JAVA(**16%**)

- **BNDES 2013**

- 40 questões específicas
- 8 relacionadas à JAVA (**20%**)

- **FINEP 2014**

- 25 questões específicas
- 4 relacionadas à JAVA (**16%**)

# Bibliografia Recomendada



**Livro:** Use a Cabeça! Java  
**Autor:** Sierra, Kathy  
**Editora:** Alta Books



**Livro:** Sun Certified  
Programmer for Java 6  
**Autores:** Sierra, Kathy;  
Bates, Bert  
**Editora:** Mcgraw Hill



**Livro:** Java – Como  
Programar – 8ª Ed. 2010  
**Autor:** Deitel  
**Editora:** Prentice Hall -  
BR

Ainda está com dúvidas? Recorra à especificação:  
<http://docs.oracle.com/javase/specs/>



## ■ Módulo 2

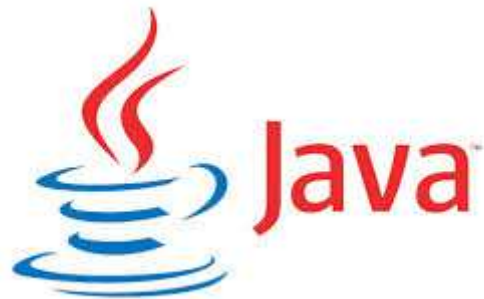
- Classe x Objeto
- Construtores
- Métodos Estáticos
- Modificadores Não-Referentes a Acesso
- Palavra Reservada Static
- Características da Programação Orientada a Objetos
- Interfaces
- Classes Abstratas

# Ementa

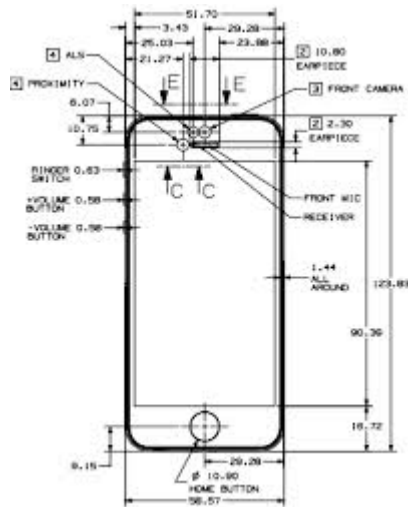
## ■ Módulo 2

- Regras de Sobreposição de Métodos
- Regras de Sobrecarga de Métodos
- Casting
- Método Equals()
- Método Hashcode()
- Garbage Collector
- Threads

# Questões



# Classe x Objeto



- Um exemplo da diferença entre classe e objeto pode ser vista acima: um projeto de um celular é uma **classe** e as unidades do celular a serem comercializadas são os **objetos** (instâncias).



# Classe x Objeto

- O projeto da **classe** Celular mapeado para a linguagem Java ficaria assim:

```
public class Celular {  
    private Integer numeroSerie;  
    private String cor;  
    private String fabricante;  
  
    public void ligar(){  
        System.out.println("Chamada ao método ligar");  
    }  
    public void despertar(){  
        System.out.println("Chamada ao método despertar");  
    }  
}
```

Celular
+Integer numeroSerie +String cor +String fabricante
+ligar() +despertar()

# Classe x Objeto

- As unidades fabricadas do tipo de celular projetado são **objetos** ou **instâncias** da classe Celular.

```
public class TesteCelular {  
    public static void main(String[] args) {  
        Celular objetoCelular = new Celular();  
        objetoCelular.numeroSerie = 1234;  
        objetoCelular.cor = "Preto";  
        objetoCelular.fabricante = "Apple"  
        objetoCelular.despertar();  
        objetoCelular.ligar();  
        Celular.despertar(); //O que ocorrerá nesta linha?  
        Celular.ligar(); //O que ocorrerá nesta linha?  
    }  
}
```

Celular
+Integer numeroSerie +String cor +String fabricante
+ligar() +despertar()

# Construtores (Conceitos Básicos)

- A palavra reservada **new** faz com que o método construtor da classe Celular seja chamado. Como não implementamos nenhum método construtor, o que ocorrerá?

Celular
+Integer numeroSerie +String cor +String fabricante
+ligar() +despertar()

```
public class TesteCelular {  
    public static void main(String[] args) {  
        Celular objetoCelular = new Celular();  
        objetoCelular.numeroSerie = 1234;  
        objetoCelular.cor = "Preto";  
        objetoCelular.fabricante = "Apple"  
        objetoCelular.despertar();  
        objetoCelular.ligar();  
        Celular.despertar();  
        Celular.ligar(); }  
}
```

# Construtores (Conceitos Básicos)

- Poderíamos ter implementado a classe Celular de forma que obrigássemos o usuário a informar o numero de série.

```
public class Celular {  
    public Integer numeroSerie;  
    public String cor;  
    public String fabricante;  
  
    public Celular(Integer numeroSerieInformado){  
        numeroSerie = numeroSerieInformado;  
    }  
    public void ligar(){  
        System.out.println("Chamada ao método ligar");  
    }  
    public void despertar(){  
        System.out.println("Chamada ao método despertar");  
    }  
}
```

Celular
+Integer numeroSerie +String cor +String fabricante
+ligar() +despertar()

# Métodos Estáticos

- Vimos no exemplo anterior, que os métodos `ligar()` e `despertar()` eram específicos de um modelo de celular. Logo, eles se referem às instâncias da classe `Celular`. E se quiséssemos saber o total de celulares fabricados?

```
public class Celular {  
    //Variáveis de instância foram omitidas  
    public static int quantidade;  
    public Celular(Integer numeroSerieInformado){  
        numeroSerie = numeroSerieInformado;  
        quantidade++;  
    }  
    //Métodos da instância foram omitidos  
    public static int obterQuantidadeCelulares(){  
        return quantidade;  
    }  
}
```

# Métodos Estáticos

➤ Temos abaixo a chamada ao método estático criado no exemplo anterior:

```
public class TesteMetodoEstatico {  
    public static void main(String[] args) {  
        Celular objetoCelular1 = new Celular(1234);  
        objetoCelular1.fabricante = "Apple";  
        Celular objetoCelular2 = new Celular(5678);  
        objetoCelular2.fabricante = "Samsung";  
        System.out.println (Celular.obterQuantidadeCelulares());  
        //O que ocorrerá abaixo?  
        System.out.println(objetoCelular2.obterQuantidadeCelulares());  
    }  
}
```

# Modificadores Não-Referentes a Acesso

- **Abstract** – Pode ser utilizado em métodos ou em classes.
  - Método Abstract: A implementação do método será postergada para sua subclasse. Se houver uma “cadeia de heranças”, a classe concreta deverá implementá-lo.
  - Classe Abstract: Nunca poderá ser instanciada. Seu único propósito é ser estendida (subclassificada).

**OBS:** É possível que uma classe abstrata não contenha nenhum método abstrato, mas se ela possuir pelo menos um, a classe obrigatoriamente deverá ser abstrata.

# Modificadores Não-Referentes a Acesso

## Exemplos do Modificador Abstract

```
abstract class TesteAbstrato{};
```

→ Compila sem problemas.

```
class TesteClasseConcretaComUmMetodoAbstrato{  
    abstract void metodoAbstrato1();  
}
```

→ Erro de compilação

```
abstract class ClasseAbstrata{  
    abstract void metodoNaoImplementado();  
}  
public class ClasseConcreta extends ClasseAbstrata{  
    void metodoNaoImplementado(){  
        System.out.println("Método já implementado");  
    }  
}  
public class ClasseConcretaComErro extends ClasseAbstrata{
```

→ Compila sem problemas.

→ Erro de compilação



# Palavra Reservada Static

- **Static** – Impacta diretamente sobre o comportamento de um método ou variável, por isso será tratado à parte dos demais modificadores.
  - Deverá ser utilizado em um método ou variável quando desejarmos que o mesmo não dependa da instância da classe.
  - As variáveis e métodos estáticos serão sempre executados da mesma forma, ou seja, não pertencerão mais à instância e sim à própria classe.

# Palavra Reservada Static

## Exemplo de utilização de variável estática

```
class TesteAtributoEstatico{

    static int cont = 0; //Declarando e inicializando


    public TesteAtributoEstatico(){
        cont++;
    }

    public static void main(String[] args) {
        new TesteAtributoEstatico();
        new TesteAtributoEstatico();
        new TesteAtributoEstatico();
        System.out.println("Número de classes instanciadas = " + cont);
    }
}
```

# Palavra Reservada Static


## Curiosidades sobre a palavra reservada static

Caso 1:

```
class Aluno{  
    int nota=9;  
    static void imprimirTotalAlunos(){  
        int notaTeste = nota;   
    }  
}
```

Erro nesta linha

Caso 2:

```
class Aluno{  
    int calcularNota();  
    static void imprimirMediaTurma(){  
        int nota = calcularNota();   
    }  
}
```

Erro nesta linha

# Palavra Reservada Static

## Curiosidades sobre a palavra reservada static

Caso 3:

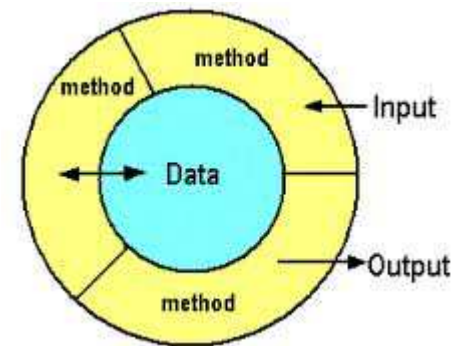
```
class Aluno{  
    static int totalAlunos;  
    static int obterTotalAlunos(){}  
  
    static double obterMediaTurma(){  
        int totalAlunosGeradoPeloMetodo = obterTotalAlunos();  
        int totalAlunosObtidoDaVariavel = totalAlunos;  
    }  
}
```

OK

OK

# Características da Programação Orientada a Objetos

## ▪ Encapsulamento



Ocultar detalhes de implementação por trás de um conjunto de métodos públicos.

- Atributos Privados: As variáveis de instância deverão ser protegidas.
- Métodos de Acesso Públicos: As classes chamadoras deverão utilizar estes métodos para acessar as variáveis de instância.

Obs: É recomendável que se utiliza a convenção JavaBeans para “getters” e “setters”.

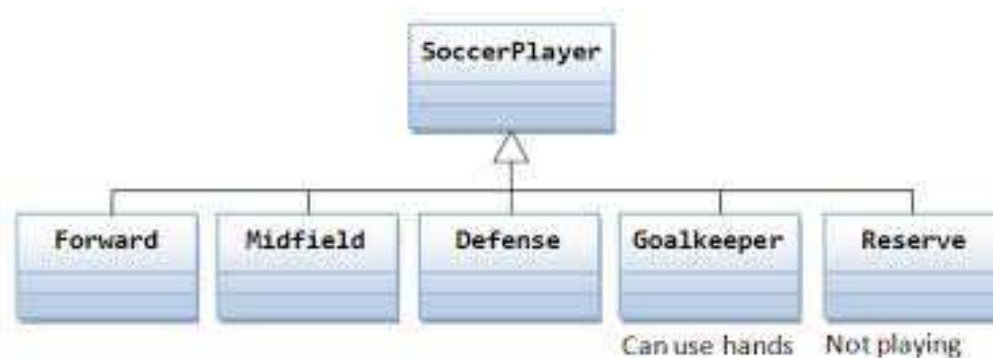
Os benefícios são: facilidade de manutenção, flexibilidade e extensibilidade.

# Características da Programação Orientada a Objetos

## ▪ Herança

Definição: “Forma de reutilização de software na qual uma nova classe é criada, absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas.” (Deitel)

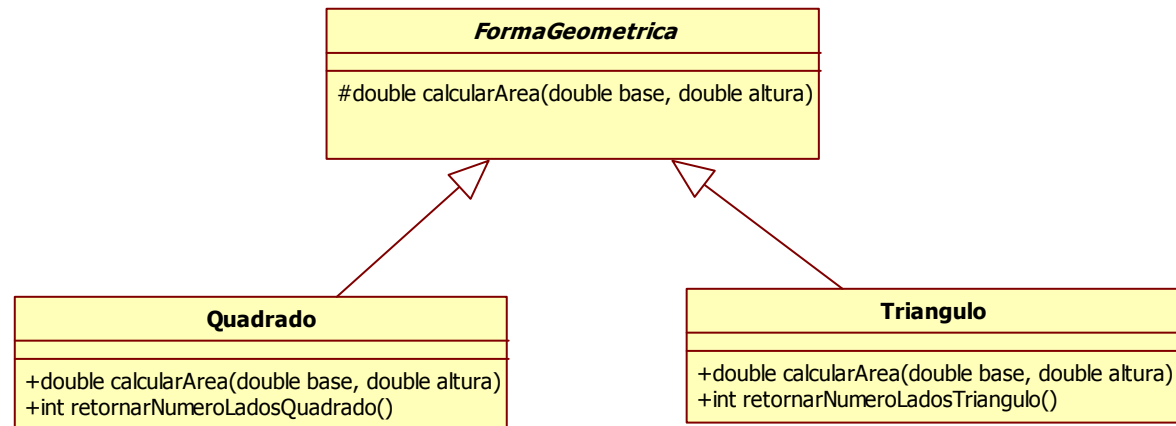
- Em java, a hierarquia de classes se inicia na classe Object (pacote java.lang).
- Os relacionamentos podem ser divididos em: “**é um**” e “**tem um**”.



# Características da Programação Orientada a Objetos

## ▪ Polimorfismo

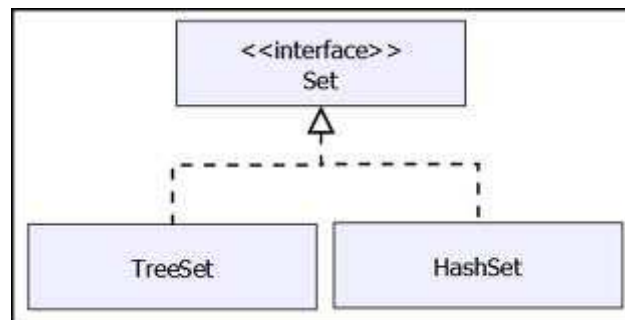
- Uma das maiores vantagens da Orientação a Objetos.
- Nos permite escrever programas que processam objetos que compartilham a mesma superclasse em uma hierarquia de classes como se todas fossem objetos da superclasse.



- Alguns autores tratam o polimorfismo a nível de método como um dos casos de polimorfismo e o chamam de polimorfismo estático.

# Interfaces

- Assemelham-se à assinatura de contratos.
- Seus atributos deverão ser constantes.
- Seus métodos deverão ser abstratos e públicos.
- Não devem conter detalhes de implementação.
- Sua utilização é aconselhável e tida como boa prática de implementação.





# Interfaces

## Exemplo de utilização de interface em Java

```
public interface Nadador {  
    String ESPECIALIDADE = "50 metros";  
    void nadar() throws NadadorException;  
}  
interface Corredor{  
    public abstract void correr();  
}  
public interface Ciclista{  
    public void pedalar();  
}  
class TriAtleta implements Nadador,Corredor,Ciclista{  
    public void nadar() throws NadadorException{  
        System.out.println("Nadando " + ESPECIALIDADE);  
    }  
    public void correr() {  
        System.out.println("Correndo");  
    }  
    public void pedalar(){  
        System.out.println("Pedalando");  
    }  
}
```

palavra-chave

os atributos são sempre constantes.

os modificadores foram omitidos.

o modificador public foi omitido.

implementação de interfaces.

utilização da constante.

# Interfaces

## Observações do Exemplo Anterior

- Se a classe TriAtleta fosse abstrata, e realizasse as mesmas interfaces, ela não seria obrigada a implementar nenhum dos métodos.
- O uso de polimorfismo poderá ser utilizado de diversas formas, como veremos adiante.
- As implementações dos métodos nadar(), correr() e pedalar() na classe TriAtleta não necessitariam ter um “corpo”.
- A visibilidade dos métodos nadar(), correr() e pedalar() na classe TriAtleta só poderá ser declarada como pública. Se a visibilidade for diferente, um erro de compilação será exibido.
- Se os métodos correr() ou pedalar() “levantassem” alguma exceção em sua assinatura, um erro de compilação seria gerado.

# Classes Abstratas

- Não podem ser instanciadas.
- Seu único propósito é ser estendida (subclassificada).
- Permitem extensibilidade e flexibilidade.

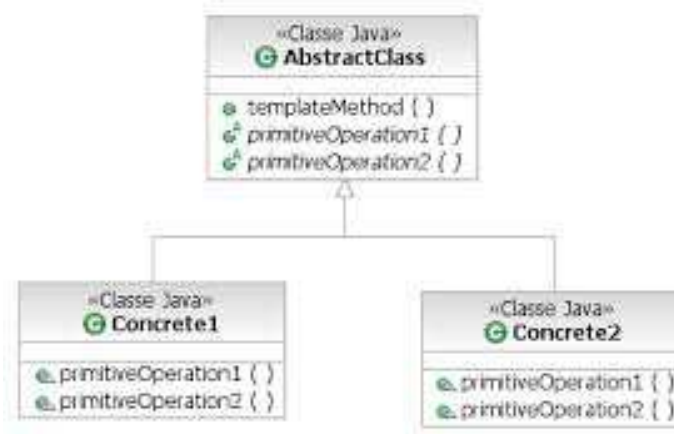
**Então qual o objetivo de criar uma classe se não é possível instanciá-la?**



# Classes Abstratas

As vantagens na utilização das classes abstratas são:

- Possibilidade de uma programação mais genérica, deixando que as classes concretas façam a programação específica.
- Grande utilização em padrões de projeto (design patterns), como por exemplo o **Template Method**.



# Classes Abstratas

## Exemplo de utilização de classe abstrata no Template Method:

```
public abstract class AbstractClass {  
  
    public final void templateMethod() {  
        //aqui vai alguma implementação FIXA  
        primitiveOperation1();  
        primitiveOperation2();  
    }  
  
    public abstract void primitiveOperation1();  
    public abstract void primitiveOperation2();  
}  
  
public class Concrete1 extends AbstractClass {  
  
    public void primitiveOperation1() {  
        //implementação específica  
    }  
  
    public void primitiveOperation2() {  
        //implementação específica  
    }  
}  
  
public class Concrete2 extends AbstractClass {  
  
    public void primitiveOperation1() {  
        //implementação específica  
    }  
  
    public void primitiveOperation2() {  
        //implementação específica  
    }  
}  
  
public class TestTemplateMethod {  
    public static void main(String[] args) {  
        AbstractClass class1 = new Concrete1();  
        AbstractClass class2 = new Concrete2();  
  
        class1.templateMethod();  
        class2.templateMethod();  
    }  
}
```

# Questão 01

**CESGRANRIO - 2010 - PETROBRAS - Analista de Sistemas Jr. – Eng. Software - 01/02**

```
1 package javaapplication2;
2
3 abstract class A {
4     public A() {
5         System.out.print("A");
6     }
7     public abstract void metodo();
8 }
9
10 class B extends A {
11     public void metodo() {
12         System.out.print("B");
13     }
14 }
15
16 class C extends A {
17     public void metodo() {
18         System.out.print("C");
19     }
20 }
21
22 public class Main {
23     public static void main(String[] args) {
24         A obj=new B();
25         obj.metodo();
26         obj=new C();
27         obj.metodo();
28     }
29 }
```

# Questão 01

## CESGRANRIO - 2010 - PETROBRAS - Analista de Sistemas Jr. – Eng. Software - 02/02

Ao tentar compilar e executar o código acima, o resultado será:

- (A) a correta compilação e execução do código, com a exibição na saída padrão da sequência BC.
- (B) a correta compilação e execução do código, com a exibição na saída padrão da sequência ABAC.
- (C) um erro de compilação, pois A é uma classe abstrata e não pode ter instâncias, como *obj*.
- (D) um erro de compilação, pois *obj* é da classe A e tentasse instanciá-lo como sendo um objeto da classe B.
- (E) um erro de execução, pois uma vez feito o binding de *obj* com a classe B, não se pode mudar a classe do mesmo.

# Questão 01

## CESGRANRIO - 2010 - PETROBRAS - Analista de Sistemas Jr. – Eng. Software

Ao tentar compilar e executar o código acima, o resultado será:

(A) a correta compilação e execução do código, com a exibição na saída padrão da sequência BC.

➡ (B) a correta compilação e execução do código, com a exibição na saída padrão da sequência ABAC.

(C) um erro de compilação, pois A é uma classe abstrata e não pode ter instâncias, como *obj*.

(D) um erro de compilação, pois *obj* é da classe A e tentasse instanciá-lo como sendo um objeto da classe B.

(E) um erro de execução, pois uma vez feito o binding de *obj* com a classe B, não se pode mudar a classe do mesmo.

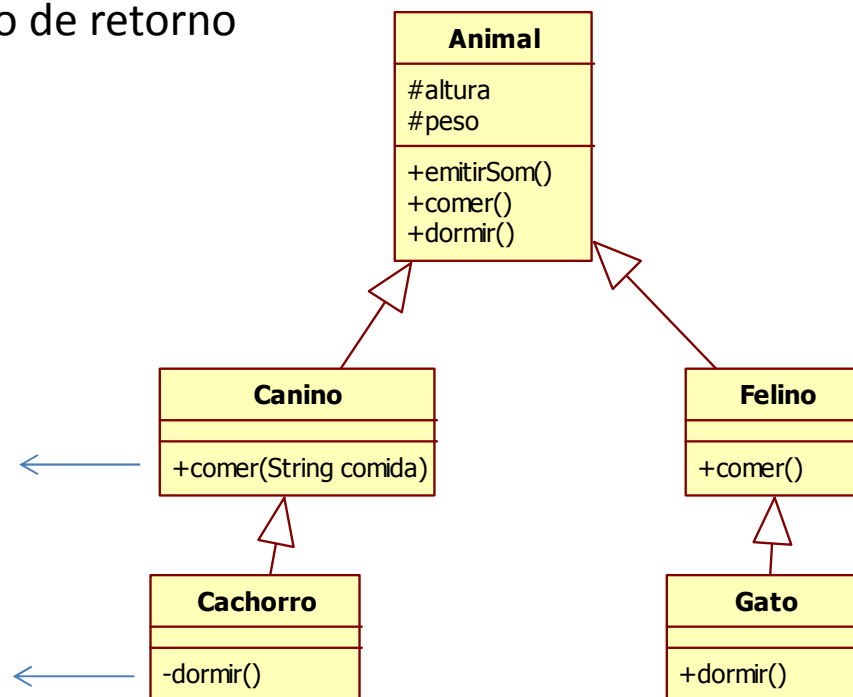


# Regras de Sobreposição de Métodos

- Os argumentos deverão ser iguais e o tipo de retorno compatível.

Isto não é sobreposição. É sobrecarga.

Isto não é sobreposição. O modificador de acesso não pode ser mais restritivo. Ele deverá ser o mesmo ou “superior”.



# Regras de Sobrecarga de Métodos

```
Class Animal{  
    public String dormir(){}  
}
```

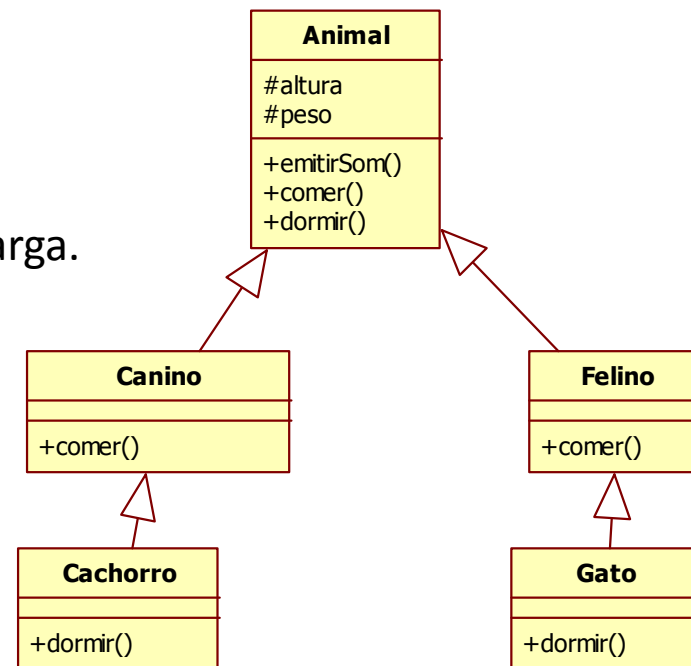
```
Class Cachorro{  
    public Integer dormir(){}  
}
```

Não é sobrecarga.

```
Class Animal{  
    public String dormir(){}  
}
```

```
Class Cachorro{  
    public Integer dormir(Int parametro1){}  
}
```

Sobrecarga válida.



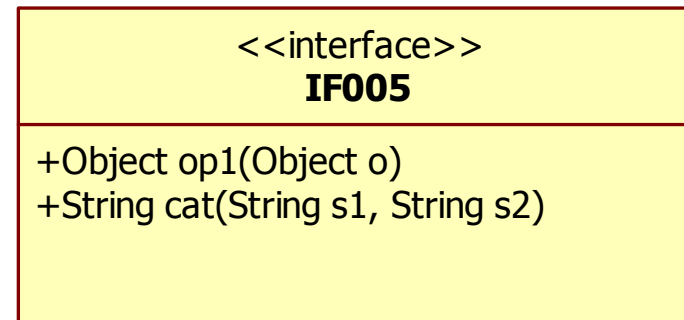
## Questão 02

### CESGRANRIO - 2013 - BNDES - Analista de Sistemas - Desenvolvimento

Seja a seguinte interface Java:

```
public interface IF005 {  
    Object op1(Object o);  
    String cat(String s1,String s2);  
}
```

Qual classe implementa IF005 corretamente?



## Questão 02

### CESGRANRIO - 2013 - BNDES - Analista de Sistemas - Desenvolvimento

(A)

```
public class CL01 implements IF005 {  
    private final Object op1(Object o){  
        return new Object();  
    }  
    public String cat(String s1,String s2) {  
        return s1+s2;  
    }  
}
```

<b>&lt;&lt;interface&gt;&gt;</b> <b>IF005</b>
+Object op1(Object o) +String cat(String s1, String s2)

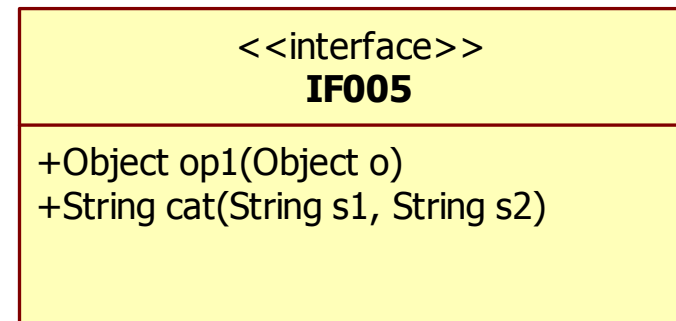
- Não se pode reduzir a visibilidade de um método implementado.
- Quando a visibilidade de um método declarado na interface não é informada, ela assume por padrão a visibilidade public.

## Questão 02

### CESGRANRIO - 2013 - BNDES - Analista de Sistemas - Desenvolvimento

(C)

```
public class CL03 implements IF005 {  
    public Object op1(String s) {  
        return "";  
    }  
    public String cat(String a,String b) {  
        return "";  
    }  
}
```



- Neste caso, o método op1 não foi implementado corretamente, pois o parâmetro deveria obrigatoriamente ser do tipo Object.

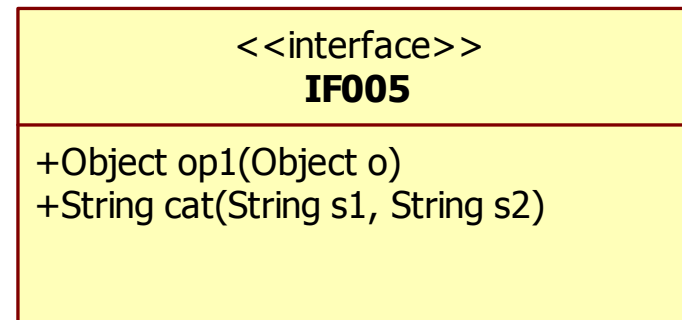
**CURIOSIDADE:** Se o retorno do método op1 fosse de um tipo mais específico do que o tipo Object, o código estaria correto.

## Questão 02

### CESGRANRIO - 2013 - BNDES - Analista de Sistemas - Desenvolvimento

(D)

```
public class CL04 implements IF005 {  
    protected String op1(Object s) {  
        return "";  
    }  
    public String cat(String a,String b) {  
        return "";  
    }  
}
```



- Não se pode reduzir a visibilidade de um método implementado.
- Quando a visibilidade de um método declarado na interface não é informada, ela assume por padrão a visibilidade public.

## Questão 02

### CESGRANRIO - 2013 - BNDES - Analista de Sistemas - Desenvolvimento

(E)

```
public class CL05 implements IF005 {  
    Object op1(Object o) {  
        return "";  
    }  
    String cat(String a,String b) {  
        return "";  
    }  
}
```

<b>&lt;&lt;interface&gt;&gt;</b> <b>IF005</b>
+Object op1(Object o) +String cat(String s1, String s2)

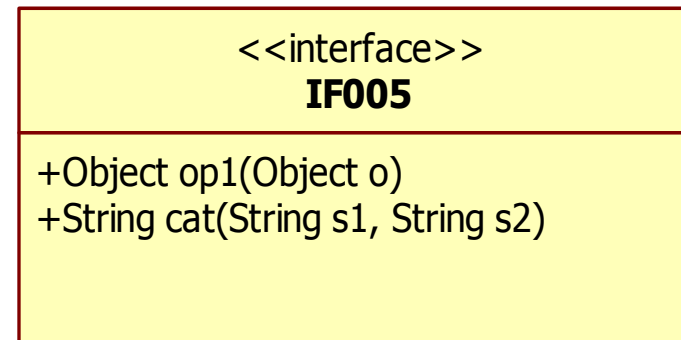
- Uma observação importante é que ao omitirmos a visibilidade de um método em uma classe, o valor assumido será package. No caso da omissão da visibilidade de métodos em interfaces, o valor assumido será public.

## Questão 02

### CESGRANRIO - 2013 - BNDES - Analista de Sistemas - Desenvolvimento

→ (B)

```
public class CL02 implements IF005 {  
    public final String op1(Object o) {  
        return "";  
    }  
    public String cat(String a,String b) {  
        return "";  
    }  
}
```



- O retorno de um método implementado poderá ser de um tipo mais específico. Neste caso, poderia ser qualquer descendente de Object.
- O fato de incluir o modificador final no método implemetado não produz erro de compilação.



## Questão 03

CESGRANRIO - 2010 - PETROBRAS - Analista de Sistemas – Engenharia de SW

A Classe C2 pode manipular os atributos

- (A) x, y, z
- (B) y, z
- (C) x, y
- (D) y
- (E) x, y, z, w

```
// Arquivo C1.java

package br.com.pk1;

public class C1 {
    int x;
    public int y;
    protected int z;
    private int w;
}

// Arquivo C2.java

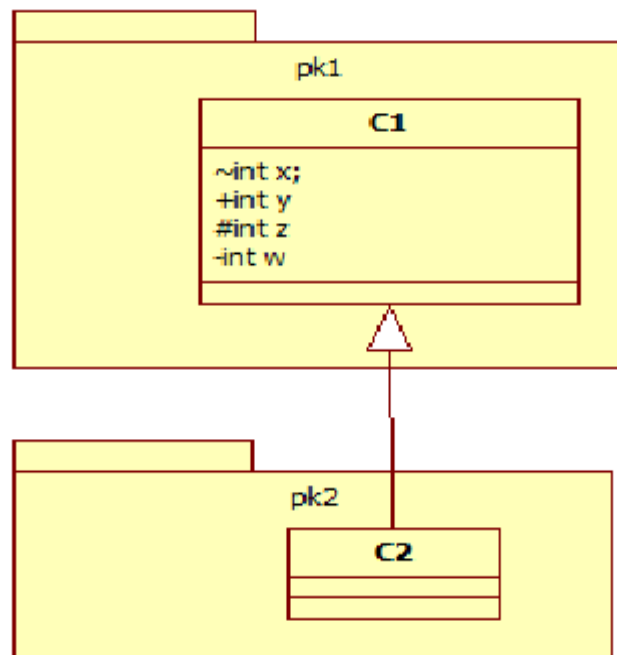
package br.com.pk2;

public class C2 extends C1 {

}
```

## Questão 03

CESGRANRIO - 2010 - PETROBRAS - Analista de Sistemas – Engenharia de SW



A Classe C2 pode manipular os atributos

- (A) x, y, z
- (B) y, z
- (C) x, y
- (D) y
- (E) x, y, z, w

## Questão 03

CESGRANRIO - 2010 - PETROBRAS - Analista de Sistemas – Engenharia de SW

A Classe C2 pode manipular os atributos

- (A) x, y, z
- (B) y, z
- (C) x, y
- (D) y
- (E) x, y, z, w

```
// Arquivo C1.java  
  
package br.com.pk1;  
  
public class C1 {  
    int x;  
    public int y;  
    protected int z;  
    private int w;  
}  
  
// Arquivo C2.java  
  
package br.com.pk2;  
  
public class C2 extends C1 {  
  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

Considere as classes e interfaces Java abaixo, em que cada qual ocupa seu próprio arquivo.

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

```
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

Qual classe **NÃO** produz erros de compilação?

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}  
  
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(A)

```
public class A extends ClsA implements ItX {  
    String x;  
    Integer l;  
    public void op1(Object x) {}  
    public String op2(Object x,String y) {  
        return null;  
    }  
    public void opA1(int a) {}  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

# Questão 04

## CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}  
  
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(A)

```
public class A extends ClsA implements ItX {  
    String x;  
    Integer l;  
    public void op1(Object x) {}  
    public String op2(Object x,String y) {  
        return null;  
    }  
    public void opA1(int a) {}  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}
```

```
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(B)

```
public class B extends ClsB implements ItX {  
    float f,g;  
    public void op1(Object x) {}  
    public String op2(Object x,String y) {  
        return null;  
    }  
    public void opB3(int a) {}  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}
```

```
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(B)

```
public class B extends ClsB implements ItX {  
    float f,g;  
    public void op1(Object x) {}  
    public String op2(Object x,String y) {  
        return null;  
    }  
    public void opB3(int a) {}  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```



## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}  
  
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(C)

```
public class C extends ClsA implements ItX {  
    String s1,s2;  
    public void op1(Object x) {}  
    public String op2(String x,String y) {  
        return null;  
    }  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}  
  
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(C)

```
public class C extends ClsA implements ItX {  
    String s1,s2;  
    public void op1(Object x) {}  
    public String op2(String x,String y) {  
        return null;  
    }  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}  
  
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(D)

```
public abstract class D extends ClsB implements ItX {  
    String s1;  
    int x,y,z;  
    float f;  
    public void op1(Object x) {}  
    public String op2(String x,String y) {  
        return null;  
    }  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}
```

```
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```



(D)

```
public abstract class D extends ClsB implements ItX {  
    String s1;  
    int x,y,z;  
    float f;  
    public void op1(Object x) {}  
    public String op2(String x,String y) {  
        return null;  
    }  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}  
  
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(E)

```
public class E extends ClsA implements ItX {  
    String s1,s2;  
    public void op1(Object x) {}  
    public String op2(Object x,String y) {  
        return null;  
    }  
    public void opA2(float b) {}  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

## Questão 04

### CESGRANRIO - 2013 - FINEP - Analista de Sistemas – Desenvolvimento

```
public class ClsA {  
    int x;  
    int y;  
    public final void opA1(int a) {}  
    public static void opA2(float b) {}  
    public void opA3(int a,String b) {}  
}  
  
public abstract class ClsB {  
    String s1;  
    int i;  
    public void opB1(String s) {}  
    public abstract void opB2(String s,String t);  
    public void opB3(int a,int b) {}  
}
```

(E)

```
public class E extends ClsA implements ItX {  
    String s1,s2;  
    public void op1(Object x) {}  
    public String op2(Object x,String y) {  
        return null;  
    }  
    public void opA2(float b) {}  
}
```

```
public interface ItX {  
    void op1(Object x);  
    String op2(Object x,String y);  
}
```

# Casting

- O casting não modifica o objeto ou o valor que está sendo moldado. O receptor do cast constitui um novo objeto ou um novo tipo.
- Uma das formas mais comuns de casting é o de tipos primitivos. Sempre que houver uma transferência de dados de um tipo menos preciso para um tipo mais preciso, o casting não precisa ser explícito.

```
int i = 7;  
double d = i + 4.9587;
```

exemplo de casting automático  
(implícito)

- Sempre que houver uma transferência de dados de um tipo mais preciso para um menos preciso, o casting precisa ser explícito. Neste caso, poderá ocorrer uma perda de dados.

```
double d = 4.9587;  
int i = (int)d; // perda de precisão
```

exemplo de casting explícito

# Casting

A conversão de objetos funciona como no exemplo abaixo:

Se Canino herda de Animal, então:

```
Animal animal = new Canino();
```

O comando abaixo é similar e também funciona:

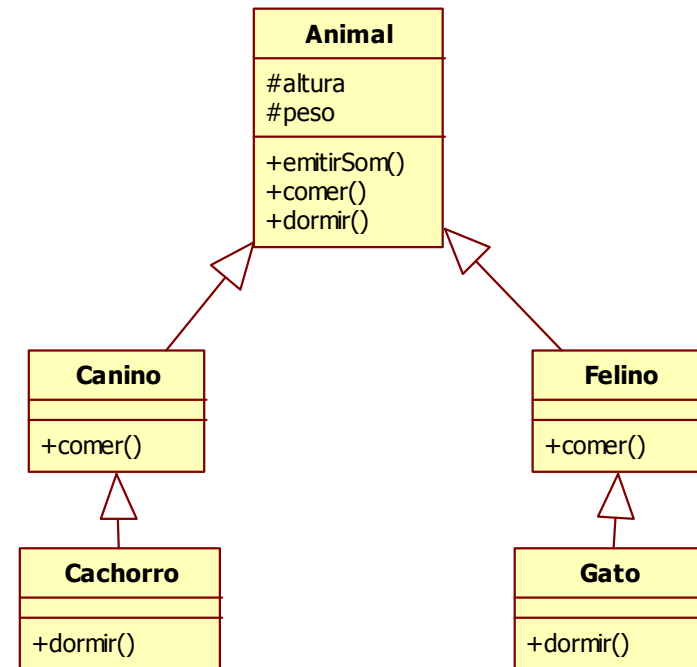
```
Animal animal = (Animal) new Canino();
```

```
Animal animal = new Canino();  
Canino canino = (Canino) animal;
```

Neste exemplo, seria possível acessar um método exclusivo de canino.

Para testar se a conversão explícita acima realmente funciona, devemos utilizar o operador **instanceof**.

upcasting é automático



downcasting exige conversão explícita



# Operador instanceof

- O operador instanceof é usado para verificar se um objeto é de um **tipo** específico.
  - O tipo poderá ser uma interface ou uma classe.
  - O teste poderá ser feito da seguinte forma: o objeto referenciado pela variável da esquerda **É-MEMBRO** do tipo de interface ou classe do lado direito do operador.

```
public static void main(String[] args) {  
    Integer i = new Integer(5);  
    if (i instanceof Integer){  
        System.out.println("i é do tipo Integer");  
    }  
}
```

Verifica se a variável **i** é uma instância da classe **Integer**.

**DICA:** Mesmo se o objeto não pertencer à classe comparada, o método retornará true se o objeto puder ser atribuído a esse tipo.

# Método equals()

## ▪ Características

- Método da classe **Object**.
- Utilizado na comparação entre objetos.
- Cada classe pode sobrescrevê-lo se achar conveniente.
- Determina se duas classes são “significativamente equivalentes”.
- O método equals da classe Object considera apenas o operador “==”.
- Ao criarmos nossas classes, é recomendável que implementemos os métodos **equals** e **hashCode**.

# Método equals()

```
class Carro{
    private String placa;
    public Carro(String placa){
        this.placa=placa;
    }
    public String getPlaca(){
        return this.placa;
    }
    public boolean equals(Object o){
        if ((o instanceof Carro && ((Carro)o).getPlaca()==this.getPlaca())){
            return true;
        }
        else{
            return false;
        }
    }
}
```

```
public class TesteEquals {
    public static void main(String[] args) {
        Carro carro1=new Carro("ABC1234");
        Carro carro2=new Carro("ABC1234");
        if (carro1.equals(carro2)){
            System.out.println("Carros iguais!");
        }
    }
}
```

**Primeiro verificamos se o objeto passa no teste do “instanceof” e só depois usamos a regra de igualdade desejada.**

# Método equals()

## ➤ O Contrato de equals()

Definido pela própria documentação Java.

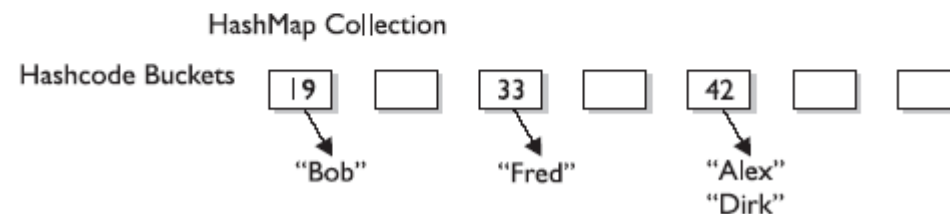
- **É reflexivo:** Para qualquer valor de referência de x, x.equals() deve retornar true .
- **É simétrico:** Para qualquer valor de referência de x e y, x.equals(y) deve retornar true se, e somente se, y.equals(x) retornar true.
- **É transitivo:** Para qualquer valor de referência de x, y e z, se x.equals(y) retornar true e y.equals(z) também retornar true, então, x.equals(z) deve retornar true.
- **É consistente:** Para qualquer valor de referência de x e y, múltiplas chamadas de x.equals(y), retornarão consistentemente true.

# Método hashCode()

- Substituir hashCode() é necessário para:
  - Usar o objeto como chave em uma tabela de hashing.
  - Definir um tipo de número identificador do objeto, não necessariamente exclusivo.

Observe um exemplo de um algoritmo de Hashcode.

Key	Hashcode Algorithm	Hashcode
Alex	$A(1) + L(12) + E(5) + X(24) = 42$	
Bob	$B(2) + O(15) + B(2) = 19$	
Dirk	$D(4) + I(9) + R(18) + K(11) = 42$	
Fred	$F(6) + R(18) + E(5) + (D) = 33$	



# Método hashCode()

## ➤ Substituindo hashCode()

- A recuperação adequada baseada em hashing é um processo de duas etapas:

1. Encontrar o depósito certo.
2. Procurar no depósito o elemento certo.

**Observação:** Portanto para obter eficiência seu objetivo será distribuir os elementos da maneira mais regular possível por todos os depósitos.

# Método hashCode()

- O contrato de hashCode()
  - Definido pela própria documentação Java.
  - Sempre que for chamado no mesmo objeto mais de uma vez, durante a execução de um aplicativo Java, o método deve retornar consistentemente o mesmo valor.
  - Se dois objetos forem iguais de acordo com o método equals(), então o hashCode(), dos dois objetos devem ser iguais.
  - Se dois objetos forem diferentes de acordo com o método equals(), então o hashCode() dos dois objetos não necessariamente deve ser diferente. Mas deve ser ficar alerta que produzindo resultados distintos, pode melhorar o desempenho em tabelas de hashing.
  - Se a chamada em dois objetos produzir hashCode() diferentes, então a chamada ao método equals() deve ser diferente (false).

# Garbage Collector

A forma mais básica de remover a referência a um objeto é configurar com null a variável de referência que estiver apontando para ele.

```
public static void main(String[] args){  
    Aluno aluno = new Aluno("João");  
    System.out.println(aluno.getNome());  
    //Aqui o objeto não está qualificado para a coleta.  
    aluno = null;  
    //Neste ponto, o objeto está qualificado para a coleta.  
}
```



# Garbage Collector

Também podemos desassociar uma variável de referência de um objeto configurando-a para referenciar outro objeto.

```
public static void main(String[] args){
    Aluno aluno1 = new Aluno("João");
    Aluno aluno2 = new Aluno("José");
    System.out.println(aluno1.getNome());
    //Aqui o objeto aluno1 não está qualificado para a coleta.
    aluno1 = aluno2; //redireciona aluno1 para referenciar o objeto aluno2
    //Agora o objeto aluno1 está qualificado para a coleta.
}
```

## Questão 05

### CESGRANRIO - 2011 - TRANSPETRO - Analista de Sistemas Jr. – Software

```
1 import java.util.ArrayList;
2
3 class C1 {
4 }
5
6 class C2 extends C1 {
7 }
8
9 public class Teste {
10     public static void main(String args[]) {
11         C1 a = new C1();
12         C2 b = new C2();
13         C1 c = new C2();
14         ArrayList<C1> lista = new ArrayList<C1>();
15         lista.add(a);
16         lista.add(b);
17         lista.add(c);
18         lista.add(new C1());
19         lista.add(lista.get(1));
20         lista.add(lista.get(2));
21         lista.remove(0);
22         a = null;
23         b = null;
24         c = null;
25         /* AQUI */
26     }
27 }
```

Analisando-se o código acima, na linha 25, qual a quantidade de objetos, que são instâncias de C1, elegível à coleta de lixo?

- (A) 0
- (B) 1
- (C) 2
- (D) 3
- (E) 4

## Questão 05

### CESGRANRIO - 2011 - TRANSPETRO - Analista de Sistemas Jr. – Software

```
1 import java.util.ArrayList;
2
3 class C1 {
4 }
5
6 class C2 extends C1 {
7 }
8
9 public class Teste {
10     public static void main(String args[]) {
11         C1 a = new C1();
12         C2 b = new C2();
13         C1 c = new C2();
14         ArrayList<C1> lista = new ArrayList<C1>();
15         lista.add(a);
16         lista.add(b);
17         lista.add(c);
18         lista.add(new C1());
19         lista.add(lista.get(1));
20         lista.add(lista.get(2));
21         lista.remove(0);
22         a = null;
23         b = null;
24         c = null;
25         /* AQUI */
26     }
27 }
```

Analisando-se o código acima, na linha 25, qual a quantidade de objetos, que são instâncias de C1, elegível à coleta de lixo?

(A) 0

→ (B) 1

(C) 2

(D) 3

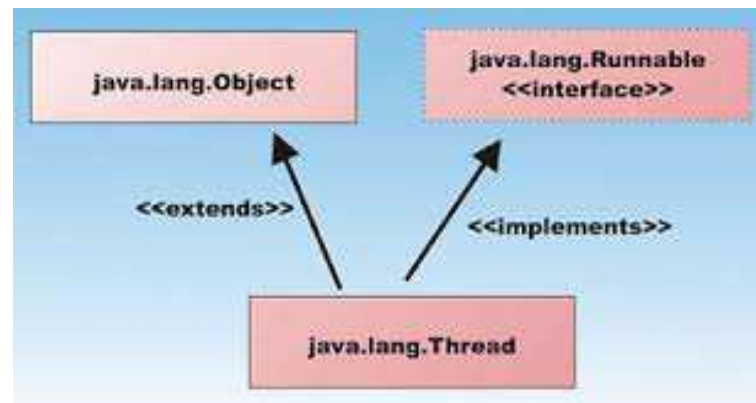
(E) 4

# Threads

- Características de um Thread Java;
- Criando um Thread;
- Ciclo de Vida de um Thread;
- Principais Métodos;

# Threads

- Um thread em Java pode significar dois conceitos:
  - Uma instância da classe `java.lang.Thread`;
  - Um thread de execução;



- A JVM possui a sua “própria parcela” da CPU, independentemente do mecanismo de agendamento que o sistema operacional usar, e agenda seus próprios threads.

# Threads

- Para utilizarmos threads separados, devemos começar implementando o método run().

```
public void run(){  
    //código a ser implementado  
}
```

- Podemos criar um thread de **duas formas**:

- Estendendo a classe java.lang.Thread;
- Implementando a interface Runnable;

# Threads

- 1ª Forma: Estendendo java.lang.Thread

```
class MinhaThread extends Thread{  
    public void run(){  
        //Código desejado  
    }  
}
```

```
MinhaThread t = new MinhaThread();
```

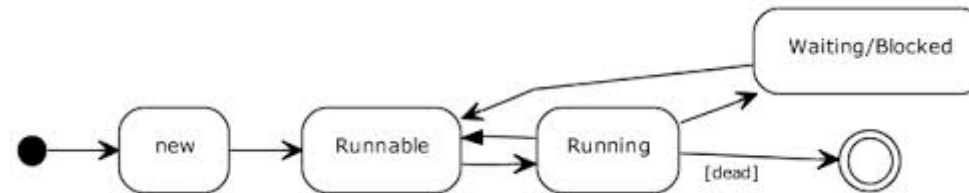
- 2ª Forma: Implementando java.lang.Runnable

```
class RunnableThread implements Runnable{  
    public void run(){  
        //Código desejado  
    }  
}
```

```
RunnableThread r = new RunnableThread();  
Thread t = new Thread(r);
```

# Threads

- Quando um thread é instanciado, ele está no estado “**novo**”. Apenas quando o método `start()` é chamado que o thread é considerado **ativo**.



- O método **isAlive()** é a melhor forma de saber se um thread que foi iniciado concluiu seu método `run()`.
- Os principais métodos da classe `Thread` são: **start()**, **yield()**, **sleep()** e **run()**.



# Threads

- Segue um exemplo prático da utilização de Threads:

```
public class ClasseRunnable implements Runnable{  
    public void run(){  
        for(int i=0; i<10; i++){  
            System.out.println("Execução: " + i);  
        }  
    }  
}
```

```
public class TesteThread{  
    public static void main(String[] args) {  
        ClasseRunnable r = new ClasseRunnable();  
        Thread t = new Thread(r);  
        t.start();  
    }  
}
```

# Threads

- Quando threads são executados de forma simultânea, não há como prever o comportamento da JVM.

```
public class ClasseRunnable implements Runnable{  
    public void run(){  
        for(int i=1; i<3; i++){  
            System.out.println("Executando: " + Thread.currentThread().getName() +  
                " i= " + i);  
        }  
    }  
}
```

```
public class TesteThread{  
    public static void main(String[] args) {  
        ClasseRunnable r = new ClasseRunnable();  
        Thread t1 = new Thread(r);  
        t1.setName("Carro 1");  
        Thread t2 = new Thread(r);  
        t2.setName("Carro 2");  
        Thread t3 = new Thread(r);  
        t3.setName("Carro 3");  
        t1.start(); t2.start(); t3.start();  
    }  
}
```



# Threads

## ➤ Resultado da primeira execução:

```
Executando: Carro 1 i= 1  
Executando: Carro 1 i= 2  
Executando: Carro 3 i= 1  
Executando: Carro 2 i= 1  
Executando: Carro 2 i= 2  
Executando: Carro 3 i= 2
```

## ➤ Resultado da segunda execução:

```
Executando: Carro 2 i= 1  
Executando: Carro 2 i= 2  
Executando: Carro 1 i= 1  
Executando: Carro 1 i= 2  
Executando: Carro 3 i= 1  
Executando: Carro 3 i= 2
```

# Threads

- Os métodos de um Thread são:
  - **Sleep:** Método estático da classe Thread que força uma thread a entrar em seu modo de suspensão.

```
try {  
    Thread.sleep(1000);  
} catch (InterruptedException e) {  
    //Tratamento apropriado  
}
```



**IMPORTANTE:** O uso do método sleep() é a melhor maneira de tentar fazer que um thread não seja executado do início ao fim em detrimento de outro.

**CURIOSIDADE:** Quando um thread “despertar” do método sleep, qual será o seu estado?

# Threads

- **Yield:** Método estático da classe Thread que faz com que um thread que está sendo executado (running) volte para o estado executável (runnable).

**IMPORTANTE:** O método yield faz com que threads com a mesma prioridade tenham oportunidade de serem processados.

➤ Podemos definir **prioridades** para um thread da seguinte forma:

```
ClasseExecutavel r = new ClasseExecutavel();  
Thread t = new Thread(r);  
t.setPriority(8);  
t.start();
```

**OBS:** Embora a prioridade padrão seja 5, a classe Thread possui 3 constantes, as quais definem o intervalo de prioridade dos Threads:

Thread.MIN\_PRIORITY(1), Thread.NORM\_PRIORITY(5) e  
Thread.MAX\_PRIORITY(10)

# Threads

- **Join:** Método não-estático da classe Thread que faz com que um thread “seja adicionado ao final de outro thread”.

```
Thread t = new Thread();  
t.start();  
t.join();
```

- O método acima significa: “me anexe (o thread atual – método main) ao final de t, de forma que t precise finalizar antes de mim”.

**IMPORTANTE:** Até agora vimos 3 formas pelas quais um thread pode sair do estado de execução (método sleep, método yield e método join).

## Questão 06

**CESGRANRIO - 2011 – TRANSPETRO – Analista de Sistemas Jr. – 01/02**

Uma das características da linguagem Java é a possibilidade de implementar programas com múltiplas linhas de execução (multithreaded execution). Nesse contexto, é fornecido o programa em Java abaixo.

# Questão 06

## CESGRANRIO - 2011 – TRANSPETRO – Analista de Sistemas Jr. – 02/02

```
public class TesteThread {
    private static boolean stop;

    private static synchronized void setStop() {
        stop = true;
    }

    private static synchronized boolean getStop() {
        return stop;
    }

    public void run() {
        System.out.print("W");
    }

    public static void main(String[] args)
        throws InterruptedException {
        Thread t = new Thread(new Runnable() {
            public void run() {
                int n = 0;
                System.out.print("X");
                while (!getStop())
                    n++;
                System.out.print("Y");
            }
        });
        t.start();
        Thread.sleep(10000);
        setStop();
        System.out.print("Z");
    }
}
```

Entre as possibilidades de saída resultantes da execução do programa fornecido, inclui-se a impressão de

- (A) XY
- (B) XYZ
- (C) XYZW
- (D) ZWYZ
- (E) ZXY



# Questão 06

## CESGRANRIO - 2011 – TRANSPETRO – Analista de Sistemas Jr.

```
public class TesteThread {
    private static boolean stop;

    private static synchronized void setStop() {
        stop = true;
    }

    private static synchronized boolean getStop() {
        return stop;
    }

    public void run() {
        System.out.print("W");
    }

    public static void main(String[] args)
        throws InterruptedException {
        Thread t = new Thread(new Runnable() {
            public void run() {
                int n = 0;
                System.out.print("X");
                while (!getStop())
                    n++;
                System.out.print("Y");
            }
        });
        t.start();
        Thread.sleep(10000);
        setStop();
        System.out.print("Z");
    }
}
```

Entre as possibilidades de saída resultantes da execução do programa fornecido, inclui-se a impressão de

- (A) XY
- ➡ (B) XYZ
- (C) XYZW
- (D) ZWYZ
- (E) ZXY

## Questão 07

### CESGRANRIO - 2012 – PETROBRAS – Analista de Sistemas Jr.

Sejam as seguintes classes Java:

```
public class Xpto implements Runnable {  
    public void run(){  
        try {  
            Thread.currentThread().join(0);  
            System.out.println(10);  
        }  
        catch(SecurityException e) {  
            System.out.println(20);  
        }  
        catch(IllegalMonitorStateException e) {  
            System.out.println(30);  
        }  
        catch(IllegalArgumentException e) {  
            System.out.println(40);  
        }  
        catch(Exception e) {  
            System.out.println(50);  
        }  
    }  
}
```

```
public class Q04 {  
    public static void main(String[] args) {  
        Thread t=new Thread(new Xpto());  
        t.start();  
        t.interrupt();  
    }  
}
```

O que será exibido no console após a execução do comando `t.interrupt()`?

- (A) 10
- (B) 20
- (C) 30
- (D) 40
- (E) 50

## Questão 07

### CESGRANRIO - 2012 – PETROBRAS – Analista de Sistemas Jr.

Sejam as seguintes classes Java:

```
public class Xpto implements Runnable {  
    public void run(){  
        try {  
            Thread.currentThread().join(0);  
            System.out.println(10);  
        }  
        catch(SecurityException e) {  
            System.out.println(20);  
        }  
        catch(IllegalMonitorStateException e) {  
            System.out.println(30);  
        }  
        catch(IllegalArgumentException e) {  
            System.out.println(40);  
        }  
        catch(Exception e) {  
            System.out.println(50);  
        }  
    }  
}
```

```
public class Q04 {  
    public static void main(String[] args) {  
        Thread t=new Thread(new Xpto());  
        t.start();  
        t.interrupt();  
    }  
}
```

O que será exibido no console após a execução do comando `t.interrupt()`?

- (A) 10
- (B) 20
- (C) 30
- (D) 40

➡ (E) 50

## Questão 08

### CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr.

```
class B extends A {  
    int m1() {  
        return a + b + c + d + e;  
    }  
}
```

```
public class A {  
    static int a;  
    public int b;  
    int c;  
    protected int d;  
    private int e;  
}
```

A classe B acima encontra-se no mesmo pacote que a classe A. O método m1 apresenta erro de compilação porque a seguinte variável não pode ser acessada no ponto:

- (A) a.
- (B) b.
- (C) c.
- (D) d.
- (E) e.

## Questão 08

### CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr.

```
class B extends A {  
    int m1() {  
        return a + b + c + d + e;  
    }  
}
```

```
public class A {  
    static int a;  
    public int b;  
    int c;  
    protected int d;  
    private int e;  
}
```

A classe B acima encontra-se no mesmo pacote que a classe A. O método m1 apresenta erro de compilação porque a seguinte variável não pode ser acessada no ponto:

- (A) a.
- (B) b.
- (C) c.
- (D) d.

➡ (E) e.

## Questão 09

### CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr.

```
class B extends A {  
    static int m1() { return 0; }  
    int m2() { return 1; }  
}
```

```
public class A {  
    static int m1() { return 2; }  
    int m2() { return 3; }  
  
    public static void main(String[] args) {  
        A a = new B();  
        System.out.println(a.m1()+a.m2()+B.m1());  
    }  
}
```

A saída da execução da classe A é

- (A) 1
- (B) 2
- (C) 3
- (D) 4
- (E) 5

## Questão 09

**CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr.**

```
class B extends A {  
    static int m1() { return 0; }  
    int m2() { return 1; }  
}
```

```
public class A {  
    static int m1() { return 2; }  
    int m2() { return 3; }  
  
    public static void main(String[] args) {  
        A a = new B();  
        System.out.println(a.m1()+a.m2()+B.m1());  
    }  
}
```

A saída da execução da classe A é

- (A) 1
- (B) 2
- ➡ (C) 3
- (D) 4
- (E) 5

# Questão 10

## CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr. 01/02

```
01 class C1 {  
02     public void f() {  
03         System.out.print(" 1 ");  
04     }  
05  
06     public void g() {  
07         f();  
08     }  
09 }  
10  
11 class C2 extends C1 {  
12     public void f() {  
13         System.out.print(" 2 ");  
14     }  
15 }  
16
```

```
17 public class Prog {  
18     public static void main(String args[]) {  
19         C1 a = new C1();  
20         a.f();  
21         C2 b = new C2();  
22         b.f();  
23         a = b;  
24         a.f();  
25         b.g();  
26     }  
27 }
```



## Questão 10

### CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr. 02/02

Considerando a execução do trecho de código em Java acima, o programa

- (A) sequer compila, pois a atribuição “a = b” (linha 23) está incorreta por incompatibilidade de tipos.
- (B) compila, mas é gerado um erro de execução por incompatibilidade da atribuição “a = b” (linha 23).
- (C) imprime 1 2 1 1.
- (D) imprime 1 2 1 2.
- (E) imprime 1 2 2 2.

## Questão 10

### CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr. 02/02

Considerando a execução do trecho de código em Java acima, o programa

(A) sequer compila, pois a atribuição “a = b” (linha 23) está incorreta por incompatibilidade de tipos.

(B) compila, mas é gerado um erro de execução por incompatibilidade da atribuição “a = b” (linha 23).

(C) imprime 1 2 1 1.

(D) imprime 1 2 1 2.

➡ (E) imprime 1 2 2 2.

# Questão 11

## **CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr.**

```
01 class C1 {  
02     public void mostraDados() {  
03         System.out.print(" 1 ");  
04     }  
05 }  
06  
07 class C2 extends C1 {  
08     public void mostraDados() {  
09         System.out.print(" 2 ");  
10     }  
11 }  
12
```

Após a execução do trecho acima, na saída padrão o programa

- (A) não compila.
- (B) imprime A 1.
- (C) imprime A 2.
- (D) imprime B 1.
- (E) imprime B 2.

```
13 public class Prog {  
14     public static void f(C1 c) {  
15         System.out.print(" A ");  
16         c.mostraDados();  
17     }  
18  
19     public static void f(C2 c) {  
20         System.out.print(" B ");  
21         c.mostraDados();  
22     }  
23  
24     public static void main(String args[]) {  
25         C1 c1 = new C2();  
26         f(c1);  
27     }  
28 }
```

# Questão 11

## CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr.

```
01 class C1 {  
02     public void mostraDados() {  
03         System.out.print(" 1 ");  
04     }  
05 }  
06  
07 class C2 extends C1 {  
08     public void mostraDados() {  
09         System.out.print(" 2 ");  
10     }  
11 }  
12
```

Após a execução do trecho acima, na saída padrão o programa

(A) não compila.

(B) imprime A 1.

→ (C) imprime A 2.

(D) imprime B 1.

(E) imprime B 2.

```
13 public class Prog {  
14     public static void f(C1 c) {  
15         System.out.print(" A ");  
16         c.mostraDados();  
17     }  
18  
19     public static void f(C2 c) {  
20         System.out.print(" B ");  
21         c.mostraDados();  
22     }  
23  
24     public static void main(String args[]) {  
25         C1 c1 = new C2();  
26         f(c1);  
27     }  
28 }
```

# Questão 12

**CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr. 01/02**

```
abstract class C1 {  
    void f() {  
        System.out.println("C1");  
    }  
}  
class C2 extends C1 {  
    void f() {  
        System.out.println("C2");  
    }  
}  
class C3 extends C1 {  
    void f() {  
        System.out.println("C3");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        C1 a, b, c[];  
        a = new C2();  
        b = new C3();  
        c = new C1[] {a,b};  
        for(int i=0;i<c.length;i++) {  
            c[i].f();  
        }  
    }  
}
```

## Questão 12

**CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr. 02/02**

Analizando o código ao lado, verifica-se que o programa

- (A) compila e executa imprimindo na saída padrão C1 duas vezes.
- (B) compila e executa imprimindo na saída padrão C2 e C3.
- (C) não compila, pois classes abstratas não podem ser instanciadas.
- (D) não compila, pois há incompatibilidade de tipos em atribuição.
- (E) não compila, pois um vetor foi construído de forma incorreta.

## Questão 12

**CESGRANRIO - 2010 – PETROBRAS – Analista de Sistemas Jr.**

Analizando o código ao lado, verifica-se que o programa

(A) compila e executa imprimindo na saída padrão C1 duas vezes.

➡ (B) compila e executa imprimindo na saída padrão C2 e C3.

(C) não compila, pois classes abstratas não podem ser instanciadas.

(D) não compila, pois há incompatibilidade de tipos em atribuição.

(E) não compila, pois um vetor foi construído de forma incorreta.

# Questão 13

## CESGRANRIO - 2012 – BNDES – Analista de Sistemas – Desenvolvimento – 01/02

```
----- arquivo CA01.java --
package M1;
public class CA01 {
    protected void mt01(int a) { }
}

----- arquivo CA02.java --
package M2;
import M1.*;
public class CA02 extends CA01{
    public void mt01(int a,int b){ }
    public void mt01(int a){ }
}

----- arquivo CA03.java ----
package M1.M2;
    public class CA03 {
        public double mt03(double b) {
            return 0.0;
        }
    }
}
```

```
----- arquivo CA04.java -----
public class CA04 {
    protected String mt04(String a,String b){
        return a+"-"+b;
    }
}
```

A classe Q03 contém o método main(). Ela é mostrada a seguir:

```
import M2.*;
import M1.*;
public class Q03 {

    public static void main(String[] args) {
        CA02 p=new CA02();
        CA01 q=new CA01();
        M1.M2.CA03 r=new CA03();
        String n=(new CA04()).mt04("Brasil","Brasília");
        p.mt01(2,2);
        p.mt01(7);
    }
}
```



## Questão 13

**CESGRANRIO - 2012 – BNDES – Analista de Sistemas – Desenvolvimento – 02/02**

Qual comando de main() produz um erro de compilação?

(A) CA02 p=new CA02();

(B) M1.M2.CA03 r=new CA03();

(C) String n=(new CA04()).mt04("Brasil","Brasília");

(D) p.mt01(2,2);

(E) p.mt01(7);

# Questão 13

## CESGRANRIO - 2012 – BNDES – Analista de Sistemas – Desenvolvimento

Qual comando de main() produz um erro de compilação?

(A) CA02 p=new CA02();

➡(B) M1.M2.CA03 r=new CA03();

(C) String n=(new CA04()).mt04("Brasil","Brasília");

(D) p.mt01(2,2);

(E) p.mt01(7);

## Questão 14

**CESGRANRIO - 2012 – LIQUIGAS – Profissional Jr. – Des. de Aplicações - 01/02**

Considere o seguinte trecho de código na linguagem Java.

```
class ContaBancaria {  
    protected double saldo;  
    public Conta(double SaldoInicial) // constructor  
    {  
        saldo = SaldoInicial;  
    }  
    public void deposito(double valor){  
        saldo = saldo + valor;  
    }  
    public void retirada(double valor){  
        saldo = saldo - valor;  
    }  
} // fim da classe ContaBancaria  
  
public class ContaBancariaExt extends ContaBancaria  
{  
    public void display(){  
        System.out.println("Saldo = " + saldo);  
    }  
} // fim da classe ContaBancariaExt
```

## Questão 14

**CESGRANRIO - 2012 – LIQUIGAS – Profissional Jr. – Des. de Aplicações - 02/02**

Um programa que utilize a classe ContaBancariaExt e crie uma instância (objeto) dessa classe:

- (A) não poderá chamar o método deposito.
- (B) não poderá chamar o método retirada.
- (C) poderá chamar apenas o método display.
- (D) poderá chamar todos os métodos, menos o método display.
- (E) poderá chamar todos os métodos da classe Conta-Bancaria e o método display.

## Questão 14

### CESGRANRIO - 2012 – LIQUIGAS – Profissional Jr. – Des. de Aplicações

Um programa que utilize a classe ContaBancariaExt e crie uma instância (objeto) dessa classe:

- (A) não poderá chamar o método deposito.
- (B) não poderá chamar o método retirada.
- (C) poderá chamar apenas o método display.
- (D) poderá chamar todos os métodos, menos o método display.
- ➡ (E) poderá chamar todos os métodos da classe Conta-Bancaria e o método display.

## Questão 15

**CESGRANRIO - 2011 – PETROBRAS – Analista de Sistemas Jr. – Eng. de Software– 01/02**

Analise os fragmentos de código dados abaixo.

Arquivo Interface1.java

```
package javaapplication1;  
public interface Interface1 {  
    public int metodoComum();  
}
```

Arquivo Concreta1.java

```
package javaapplication1;  
public class Concreta1 implements Interface1 {  
    public int metodoComum() {  
        return(1);  
    }  
  
    public int metodoExotico() {  
        return(2);  
    }  
}
```

Arquivo Main.java

```
package javaapplication1;  
public class Main {  
  
    public static void main(String[] args) {  
        Interface1 x=new Concreta1();  
        System.out.println(x.metodoComum());  
        System.out.println(x.metodoExotico());  
    }  
}
```

## Questão 15

**CESGRANRIO - 2011 – PETROBRAS – Analista de Sistemas Jr. – Eng. de Software– 02/02**

O resultado, obtido ao tentar compilar e executar esse conjunto de classes, será

- (A) um erro de compilação, indicando que não é possível fazer uma conversão da classe Concreta1 para a classe Interface1.
- (B) um erro de compilação, indicando que, no contexto de x, não existe metodoExotico.
- (C) nenhuma saída e um erro em tempo de execução, indicando que, dada a conversão de Concreta1 para Interface1, não é possível acessar metodoExotico.
- (D) impressão do número 1, seguida de um erro de tempo de execução, indicando que, dada a conversão de Concreta1 para Interface1, não é possível acessar metodoExotico.
- (E) impressão dos números 1 e 2.

## Questão 15

**CESGRANRIO - 2011 – PETROBRAS – Analista de Sistemas Jr. – Eng. de Software**

O resultado, obtido ao tentar compilar e executar esse conjunto de classes, será

- (A) um erro de compilação, indicando que não é possível fazer uma conversão da classe Concreta1 para a classe Interface1.
- ➡ (B) um erro de compilação, indicando que, no contexto de x, não existe metodoExotico.
- (C) nenhuma saída e um erro em tempo de execução, indicando que, dada a conversão de Concreta1 para Interface1, não é possível acessar metodoExotico.
- (D) impressão do número 1, seguida de um erro de tempo de execução, indicando que, dada a conversão de Concreta1 para Interface1, não é possível acessar metodoExotico.
- (E) impressão dos números 1 e 2.



# Questão 16

## **CESGRANRIO - 2011 – PETROBRAS – Analista de Sistemas Jr. – Eng. de Software**

Considere o seguinte código Java, contido no arquivo R.java:

```
1. class P {  
2.     private int id;  
3.     protected void finalize() {System.out.print(id);}  
4.     public P(int i) {id = i;}  
5. }  
6. class R {  
7.     public static void main(String[] args) {  
8.         P p1 = null;  
9.         for (int i = 0; i < 5; i++) {p1 = new P(i);}  
10.        System.gc();  
11.    }}
```

No momento imediatamente anterior à execução da linha 10, quantos objetos do tipo P, que foram criados na linha 9, tornaram-se elegíveis para ser apanhados para a garbage collection?

- (A) 0
- (B) 1
- (C) 4
- (D) 5
- (E) 9

## Questão 16

### CESGRANRIO - 2011 – PETROBRAS – Analista de Sistemas Jr. – Eng. de Software

Considere o seguinte código Java, contido no arquivo R.java:

```
1. class P {  
2.     private int id;  
3.     protected void finalize() {System.out.print(id);}  
4.     public P(int i) {id = i;}  
5. }  
6. class R {  
7.     public static void main(String[] args) {  
8.         P p1 = null;  
9.         for (int i = 0; i < 5; i++) {p1 = new P(i);}  
10.        System.gc();  
11.    }}
```

No momento imediatamente anterior à execução da linha 10, quantos objetos do tipo P, que foram criados na linha 9, tornaram-se elegíveis para ser apanhados para a garbage collection?

- (A) 0
- (B) 1
- (C) 4
- (D) 5
- (E) 9



## Questão 17

### CESGRANRIO - 2011 – BNDES – Analista de Sistemas - Desenvolvimento

Os usuários de um sistema WEB, desenvolvido em JAVA, reclamam de erros nos dados consultados. A equipe técnica, ao analisar a situação, concluiu que determinado método **M** está sendo invocado, simultaneamente, por diferentes threads.

Considerando-se que não há manipulação de locks no restante do sistema, o que pode ser feito para que **M** seja executado, em dado momento, somente por uma thread?

- (A) Ajustar o firewall de borda para permitir uma conexão por IP.
- (B) Implementar connection pooling no acesso ao banco de dados.
- (C) Utilizar a keyword lockable no bloco de **M**.
- (D) Declarar o método **M** como synchronized.
- (E) Reescrever **M** em Assembly, necessariamente.

## Questão 17

### CESGRANRIO - 2011 – BNDES – Analista de Sistemas - Desenvolvimento

Os usuários de um sistema WEB, desenvolvido em JAVA, reclamam de erros nos dados consultados. A equipe técnica, ao analisar a situação, concluiu que determinado método **M** está sendo invocado, simultaneamente, por diferentes threads.

Considerando-se que não há manipulação de locks no restante do sistema, o que pode ser feito para que **M** seja executado, em dado momento, somente por uma thread?

- (A) Ajustar o firewall de borda para permitir uma conexão por IP.
- (B) Implementar connection pooling no acesso ao banco de dados.
- (C) Utilizar a keyword lockable no bloco de **M**.
- ➡ (D) Declarar o método **M** como synchronized.
- (E) Reescrever **M** em Assembly, necessariamente.

## Questão 18

### CESGRANRIO - 2014 – BANCO DA AMAZÔNIA – Tecnologia da Informação – 01/02

Sejam as seguintes classes Java, que ocupam arquivos distintos:

----- arquivo V1.java ---

```
public class V1 {  
    int v[]={2,3,1,4,2,5,3,8,2,3};  
    int mv1() {  
        int s=0;  
        for(int i=0;i<v.length;i+=2)  
            s+=v[i];  
        return s;  
    }  
    float mv2(int x) {  
        return x+mv1();  
    }  
    float mv2(float x) {  
        return 3+x*mv1();  
    }  
}
```

----- arquivo V2.java -----

```
public class V2 extends V1 {  
    int mv1() {  
        int s=0;  
        for(int i=1;i<v.length;i+=2)  
            s+=v[i];  
        return s;  
    }  
    float mv2(float x) {  
        return x*mv1();  
    }  
}
```

## Questão 18

**CESGRANRIO - 2014 – BANCO DA AMAZÔNIA – Tecnologia da Informação – 02/02**

----- arquivo QX.java -----

```
public class QX {  
    public static void main(String[] args) {  
        V1 a=new V2();  
        System.out.printf("%.1f\n",a.mv2(2));  
    }  
}
```

O que será exibido no console quando o método main() for executado?

(A) 12,0

(B) 20,0

(C) 23,0

(D) 25,0

(E) 46,0

# Questão 18

## CESGRANRIO - 2014 – BANCO DA AMAZÔNIA – Tecnologia da Informação

----- arquivo V1.java ----

```
public class V1 {  
    int v[]={2,3,1,4,2,5,3,8,2,3};  
    int mv1() {  
        int s=0;  
        for(int i=0;i<v.length;i+=2)  
            s+=v[i];  
        return s;  
    }  
    float mv2(int x) {  
        return x+mv1(); → return  
        2+mv1()  
    }  
    float mv2(float x) {  
        return 3+x*mv1();  
    }  
}
```

----- arquivo V2.java -----

```
public class V2 extends V1 {  
    int mv1() {  
        int s=0;  
        for(int i=1;i<v.length;i+=2)  
            s+=v[i];  
        return s;  
    }  
    float mv2(float x) {  
        return x*mv1();  
    }  
}
```

# Questão 18

CESGRANRIO - 2014 – BANCO DA AMAZÔNIA – Tecnologia da Informação

**v.length=10**

----- arquivo V1.java ----

```
public class V1 {  
    int v[]={2,3,1,4,2,5,3,8,2,3};  
    int mv1() {  
        int s=0;  
        for(int i=0;i<v.length;i+=2)  
            s+=v[i];  
        return s;  
    }  
    float mv2(int x) {  
        return x+mv1();  
    }  
    float mv2(float x) {  
        return 3+x*mv1();  
    }  
}
```

----- arquivo V2.java -----

```
public class V2 extends V1 {  
    int mv1() {  
        int s=0;  
        for(int i=1;i<v.length;i+=2)  
            s+=v[i]; → s=23  
        return s;  
    }  
    float mv2(float x) {  
        return x*mv1();  
    }  
}
```

**Resultado final: 23 + 2 = 25**



## Questão 18

### CESGRANRIO - 2014 – BANCO DA AMAZÔNIA – Tecnologia da Informação

----- arquivo QX.java -----

```
public class QX {  
    public static void main(String[] args) {  
        V1 a=new V2();  
        System.out.printf("%.1f\n",a.mv2(2));  
    }  
}
```

O que será exibido no console quando o método main() for executado?

(A) 12,0

(B) 20,0

(C) 23,0

➡ (D) 25,0

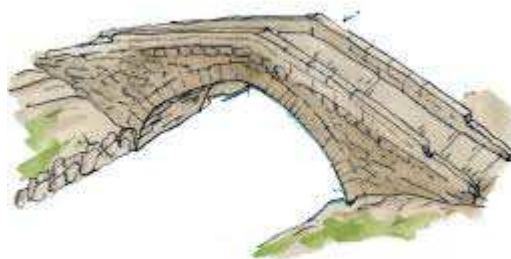
(E) 46,0

# Design Patterns

O **padrão estrutural Bridge** desacopla uma interface de sua implementação, de forma que elas possam variar independentemente. Deve ser usado quando mudanças na implementação de uma abstração não deveriam ter impacto nos clientes, isto é, o código não deveria ser recompilado.

Ex:

A abstração é a idéia (forma geométrica triangular). Ela não deveria estar acoplada às implementações dessa idéia (triângulo vermelho, verde, amarelo).



# Design Patterns

O **padrão comportamental Mediator** define um objeto que encapsula a forma como um conjunto de objetos interage. Ele promove o acoplamento fraco ao evitar que os objetos se refiram explicitamente uns aos outros.

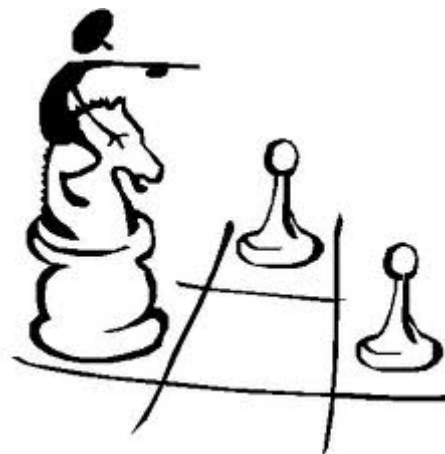
Deverá ser usado quando um conjunto de objetos se comunica de maneira bem definida, porém complexa, e o reuso de um objeto é difícil, porque ele referencia e se comunica com muitos outros objetos.



# Design Patterns

O **padrão comportamental Strategy** define uma família de algoritmos, encapsula cada um, e faz deles intercambiáveis.

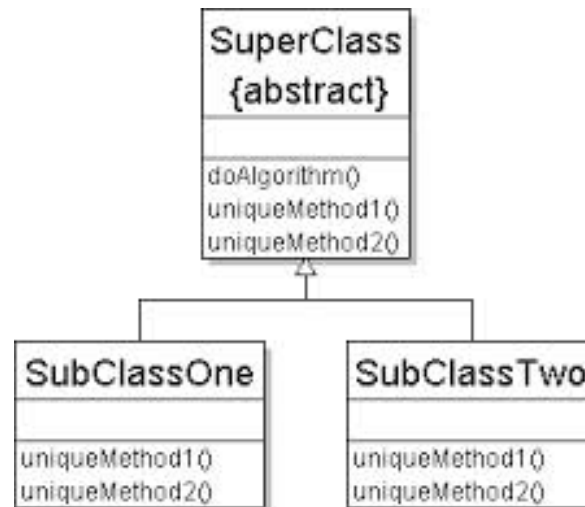
Deverá ser utilizado quando várias classes relacionadas diferem apenas em seus comportamentos ou quando uma classe define muitos comportamentos e eles aparecem como **declarações condicionais** nas suas operações.



# Design Patterns

O **padrão comportamental Template Method** define o esqueleto de um algoritmo em uma operação, deferindo alguns passos para as subclasses.

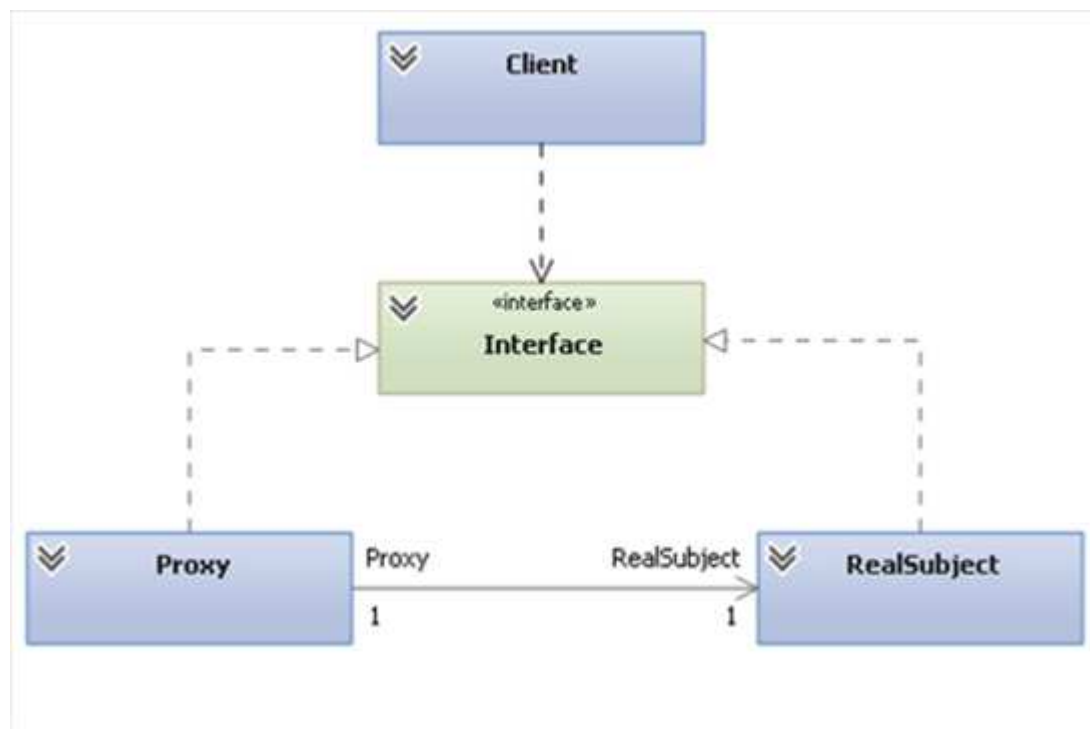
Ele permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura do mesmo.



# Design Patterns

O **padrão estrutural Proxy** provê um substituto ou ponto através do qual um objeto possa controlar o acesso a outro.

Deverá ser usado sempre quando há uma necessidade de uma referência mais versátil ou sofisticada do que um simples ponteiro para um objeto.



# Design Patterns

```
import java.util.*;

interface Imagem {
    public void mostrarImagem();
}

// no Sistema A
class ImagemReal implements Imagem {
    private String nomeDoArquivo;
    public ImagemReal(String nomeDoArquivo) {
        this.nomeDoArquivo = nomeDoArquivo;
        carregarImagemDoDisco();
    }

    private void carregarImagemDoDisco() {
        System.out.println("Carregando " + nomeDoArquivo);
    }

    public void mostrarImagem() {
        System.out.println("Mostrando " + nomeDoArquivo);
    }
}
```

# Design Patterns

```
// no Sistema B
class ImagemProxy implements Imagem {
    private String nomeDoArquivo;
    private Imagem imagem;

    public ImagemProxy(String nomeDoArquivo) {
        this.nomeDoArquivo = nomeDoArquivo;
    }
    public void mostrarImagem() {
        if(imagem == null)
            imagem = new ImagemReal(nomeDoArquivo);
        imagem.mostrarImagem();
    }
}
```



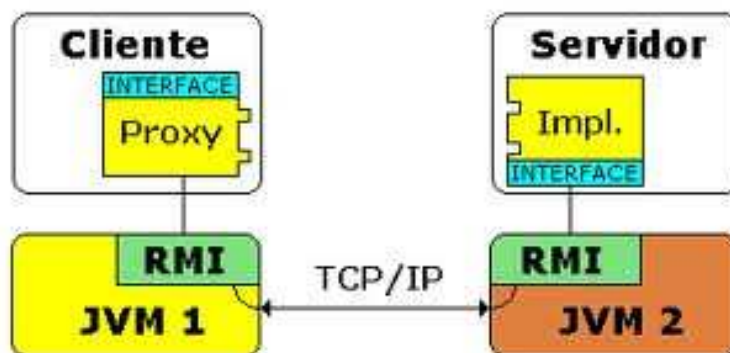
# Design Patterns

```
class ExemploProxy {  
    public static void main(String[] args) {  
        Imagem imagem1 = new ImagemProxy("ResOi_10MB_Foto1");  
        Imagem imagem2 = new ImagemProxy("ResOi_10MB_Foto2");  
  
        imagem1.mostrarImagem(); // é necessário o carregamento  
        imagem1.mostrarImagem(); // não é necessário o carregamento  
        imagem2.mostrarImagem(); // é necessário o carregamento  
        imagem2.mostrarImagem(); // não é necessário o carregamento  
        imagem2.mostrarImagem(); // não é necessário o carregamento  
        imagem1.mostrarImagem(); // não é necessário o carregamento  
    }  
}
```

# Design Patterns

## Definição da Tecnologia RMI:

O proxy (também chamado de stub) reside na máquina do cliente, e o cliente invoca o proxy em como se estivesse invocando o objeto em si. O proxy vai lidar com a comunicação com o objeto remoto, invocar o método no objeto remoto, e retornar o resultado para o cliente.



## Questão 19

### CESGRANRIO - 2014 – BANCO DA AMAZÔNIA – Tecnologia da Informação

A tecnologia RMI, presente em Java, é uma implementação de um esquema presente em qual padrão de projeto?

- (A) Bridge
- (B) Mediator
- (C) Proxy
- (D) Strategy
- (E) Template Method

## Questão 19

### CESGRANRIO - 2014 – BANCO DA AMAZÔNIA – Tecnologia da Informação

A tecnologia RMI, presente em Java, é uma implementação de um esquema presente em qual padrão de projeto?

(A) Bridge

(B) Mediator

➡ (C) Proxy

(D) Strategy

(E) Template Method

## Questão 20

### CESGRANRIO - 2014 – CEFET – Analista de Tecnologia da Informação

O programa Java a seguir foi testado e funciona perfeitamente.

Que número será impresso pelo programa quando o método main da classe Prova for executado?

(A) 1

(B) 2

(C) 3

(D) 4

(E) 5

```
public class Prova {  
    int a=2;  
    public static void main(String[] args) {  
        Prova b=new Questao();  
        Prova c=new Prova();  
        b.a=1;  
        System.out.println(c.a+b.segredo(2));  
    }  
    public int segredo(int a) {  
        return -a+this.a;  
    }  
}  
class Questao extends Prova {  
    public int segredo(int a) {  
        return a*this.a;  
    }  
}
```

## Questão 20

### CESGRANRIO - 2014 – CEFET – Analista de Tecnologia da Informação

```
public class Prova {  
    int a=2;  
    public static void main(String[] args) {  
        Prova b = new Questao();    —————> polimorfismo  
        Prova c = new Prova();  
        b.a=1;  
        System.out.println(c.a + b.segredo(2));  
    }  
    public int segredo(int a){  
        return -a+this.a;  
    }  
}
```

objeto b (Questao)

a=1

objeto c (Prova)

a=2

```
public class Questao extends Prova{  
    public int segredo(int a){  
        return a*this.a;  
    }  
}
```

c.a = 2

b.segredo(2)= 2\*1 = 2

**Resultado final: 2+2 = 4**

## Questão 20

### CESGRANRIO - 2014 – CEFET – Analista de Tecnologia da Informação

O programa Java a seguir foi testado e funciona perfeitamente.

Que número será impresso pelo programa quando o método main da classe Prova for executado?

(A) 1

(B) 2

(C) 3

➡ (D) 4

(E) 5

```
public class Prova {  
    int a=2;  
    public static void main(String[] args) {  
        Prova b=new Questao();  
        Prova c=new Prova();  
        b.a=1;  
        System.out.println(c.a+b.segredo(2));  
    }  
    public int segredo(int a) {  
        return -a+this.a;  
    }  
}  
class Questao extends Prova {  
    public int segredo(int a) {  
        return a*this.a;  
    }  
}
```

# Gabarito

- 1. B
- 2. B
- 3. B
- 4. D
- 5. B
- 6. B
- 7. E
- 8. E
- 9. C
- 10. E
- 11. C
- 12. B
- 13. B
- 14. E
- 15. B
- 16. C
- 17. D
- 18. D
- 19. C
- 20. D



# Fim

Para possíveis dúvidas sobre os assuntos desta revisão, questões de provas ou ajuda na elaboração de recursos, foi criada a lista de discussão abaixo:

[professor\\_vreis@googlegroups.com](mailto:professor_vreis@googlegroups.com)

Para elogios, sugestões de melhoria ou críticas, utilizar o e-mail encontrado no rodapé dos slides.