Java para Concursos – Módulo 2

Vitor Almeida
http://www.itnerante.com.br/profile/vitor

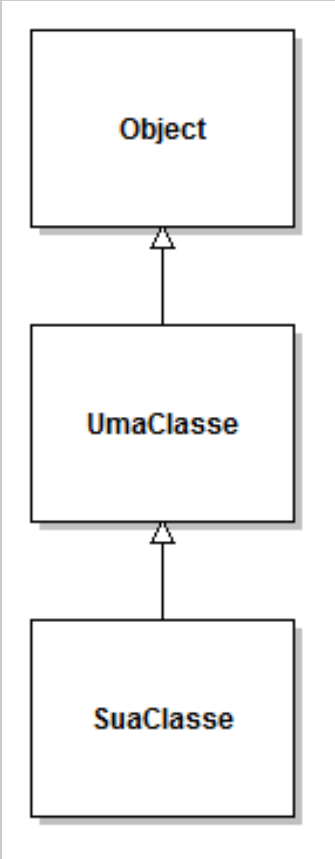
Agenda

- 1 Classes do Core Java
- 2 Herança
- 3 Tratamento de Exceção

1 Classes do Core Java

1.1 java.lang.Object

- A Classe Object representa um objeto em Java
- Todas as classes descendem diretamente ou indiretamente de Object



1.1 java.lang.Object

- Métodos de Object

```
protected Object clone()
```

  - Cria e retorna uma cópia do objeto

```
1 MinhaClasse copy = (MinhaClasse)obj.clone();
```

1.1 java.lang.Object

- Métodos de Object

```
public boolean equals(Object obj )
```

  - Compara o objeto com o objeto passado como argumento

```
1 String nome1 = "FCC";
2 String nome2 = "Wikipedia";
3 if (nome1.equals(nome2)) {
4   //instrução
5 }
```

1.1 java.lang.Object

- Métodos de Object

```
protected void finalize()
```

  - É chamado pelo Garbage Collector em um objeto prestes a ir pro lixo

1.1 java.lang.Object

- Métodos de Object

```
public final Class getClass()
```

  - Retorna um objeto da classe java.lang.Class referente ao objeto em questão

1.1 java.lang.Object

- Métodos de Object

```
public int hashCode()
```

  - Retorna o *hash code* do objeto

1.1 java.lang.Object

- Métodos de Object

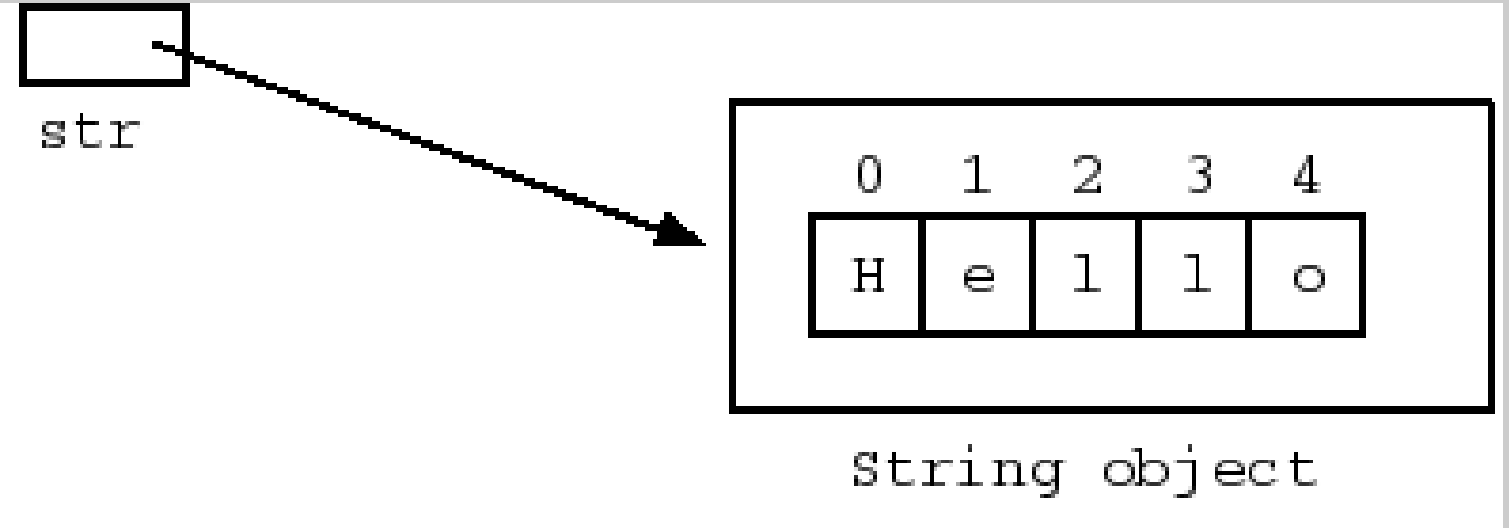
```
public String toString()
```

  - Retorna uma descrição textual do objeto

```
1  public static void main(String[] args) {
2     Pessoa pessoa = new Pessoa("Vitor", 34);
3     System.out.println(pessoa);
4     //vai imprimir algo como "Pessoa@1b533b0"
5  }
```

1.2 java.lang.String

- Uma String é uma sequência de caracteres Unicode



1.2 java.lang.String

- Strings são imutáveis
- Um objeto da classe String não precisa do new para ser criado

```
1 String s = "oi oi oi";
2 String t = new String("oi oi oi");
```

1.2 java.lang.String

- Podemos comparar Strings com o método equals

```
1 String s1 = "Java";
2 if (s1.equals("Java"))
3 if ("Java".equals(s1))
4 if (s1 != null && s1.equals("Java"))
```

1.2 java.lang.String

- Não corte uma String no meio, concatene

```
1 String s2 = "provasdeti"
2 ".com.br";
```

```
1 String s1 = "provasdeti" + ".com.br";
2 String s2 = "Adoro estudar para concursos" +
3 "com as videoaulas do provasdeti";
```

1.2 java.lang.String

- Você pode concatenar uma String com um tipo primitivo ou outro objeto

```
1 String s3 = "String número " + 3;
```

1.2 java.lang.String

- Você pode inserir sequências de escape no meio da String

```
1 String s1 = "Tem um enter \n no meio";
2 String s2 = "Vitor, o \"Bonitão\u0022";
```

1.2 java.lang.String

- A partir de Java 7, você pode usar String com o switch

```
01 String input = ...;
02 switch (input) {
03 case "hum" :
04 System.out.println("Você digitou 1");
05 break;
06 case "dois" :
07 System.out.println("Você digitou 2");
08 break;
09 default:
10 System.out.println("Valor inválido");}
```

1.2 java.lang.String

- Alguns construtores da classe String

```
public String()
public String(String original)
public String(char[] value)
public String(byte[] bytes)
public String(byte[] bytes, String charset)
```

1.2 java.lang.String

- Métodos da classe String

```
public char charAt(int index)
```

  - Retorna o char posicionado na posição indicada pelo index

```
1 "Java Java".charAt(0); //'J'
```

1.2 java.lang.String

- Métodos da classe String

```
public String concat(String s)
```

  - Concatena a String s ao fim da String original

```
1 "J J".concat(" Java"); //"J J Java"
```

1.2 java.lang.String

- Métodos da classe String

```
public boolean endsWith(String sufixo)
public boolean startsWith(String prefixo)
```

  - endsWith(): Retorna true se a String sufixo for o sufixo da String original
  - startswith(): Retorna true se a String prefixo for o prefixo da String original

```
1 "Java Java".endsWith("ava"); //true
```

1.2 java.lang.String

- Métodos da classe String

```
public int indexOf(String substring)
public int lastIndexOf(String substring)
```

  - Retorna o índice onde ocorre a primeira (ou última) aparição de substring ou -1, caso não encontre

```
1 "Java Java".indexOf(" Java"); //4
2 "Java Java".lastIndexOf("Java"); //5
```

1.2 java.lang.String

- Métodos da classe String

```
public String substring(int init)
public String substring(int init, int fim)
```

  - Retorna a substring delimitada pelo índice e o fim da String ou pelos índices

```
1 "Java Java".substring(5); //"Java"
2 "Java Java".substring(0, 1); //"Ja"
```

1.2 java.lang.String

- Métodos da classe String

```
public String replace(char old, char new)
```

  - Substitui um char pelo outro

```
1 "Java Java".replace('a', 'e'); //"Jeve Jeve"
```

1.2 java.lang.String

- Métodos da classe String

```
public int length()
public boolean isEmpty()
```

  - length(): Retorna o número de caracteres da String original
  - isEmpty(): Retorna true se a String original não tiver nenhum caractere

```
1 "Java Java".length(); //9
```

1.2 java.lang.String

- Métodos da classe String

```
public String[] split(String s)
public char[] toCharArray()
```

  - split(): Retorna um array com as Strings decorrentes da separação da String original no(s) ponto(s) onde a String s existe
  - toCharArray(): retorna um array com os caracteres da String original

```
1 "Java Java".split(" "); //["Java", "Java"]
2 "Java".toCharArray(); //['J', 'a', 'v', 'a']
```

1.2 java.lang.String

- Métodos da classe String

```
public String toLowerCase()
public String toUpperCase()
```

  - Retorna uma String com todos os caracteres da String original minúsculos ou maiúsculos

```
1 "Java Java".toLowerCase(); //"java java"
2 "Java Java".toUpperCase(); //"JAVA JAVA"
```

1.2 java.lang.String

- Métodos da classe String

```
public String trim()
```

  - Retorna uma String que será igual a String original menos eventuais espaços no início ou no fim da String original

```
1 " Java ".trim(); //"Java"
```

1.2 java.lang.String

- Métodos da classe String

```
public static String valueOf(*)
```

  - Retorna uma String com a representação do valor armazenado no parâmetro
  - Podem ser passados como parâmetros tipos primitivos, arrays e objetos

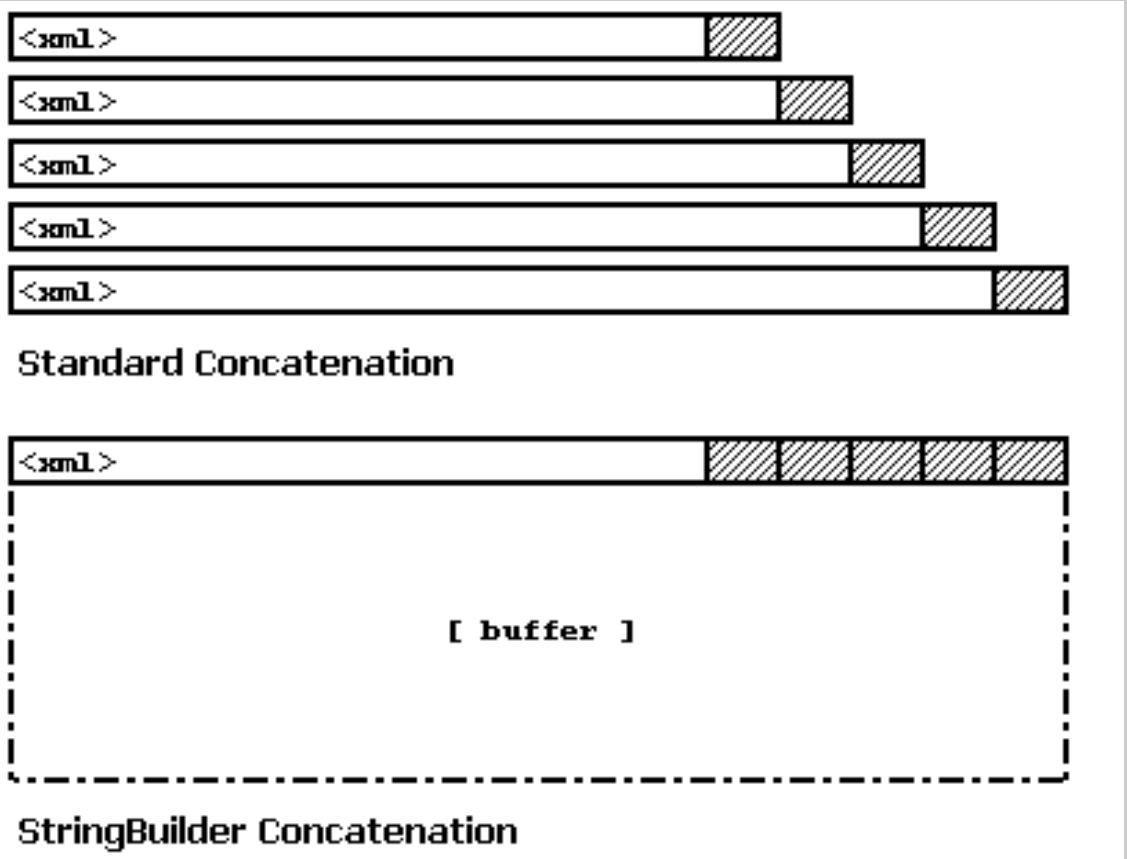
```
1 String.valueOf(23); //"23"
```

1.3 java.lang.StringBuilder

- Strings são imutáveis
- Cada operação em uma String significa um novo objeto como resultado
- Se você vai manipular Strings use as classes java.lang.StringBuffer ou java.lang.StringBuilder
- StringBuffer é sincronizada

1.3 java.lang.StringBuilder

- StringBuilder não cria novos objetos a cada manipulação



1.3 java.lang.StringBuilder

- Construtores da classe StringBuilder

```
public StringBuilder()
public StringBuilder(CharSequence seq)
public StringBuilder(int capacidade)
public StringBuilder(String s)
```

1.3 java.lang.StringBuilder

- Alguns métodos da classe StringBuilder

```
public int capacity()
public int length()
```

  - capacity(): Retorna a capacidade, ou a quantidade máxima de caracteres antes que uma nova alocação de memória precise ocorrer
  - length(): Retorna a quantidade de caracteres armazenados

```
1 StringBuilder sb = new StringBuilder(40)
2 sb.capacity(); //40
3 sb.length(); //0
```

1.3 java.lang.StringBuilder

- Métodos da classe StringBuilder

```
public StringBuilder append(String s)
```

  - Adiciona uma String ao término do StringBuilder

```
1 StringBuilder sb = new StringBuilder(100);
2 sb.append("Matrix ");
3 sb.append(2); //"Matrix 2"

4 sb.append("Matrix ").append(2);
```

1.3 java.lang.StringBuilder

- Métodos da classe StringBuilder

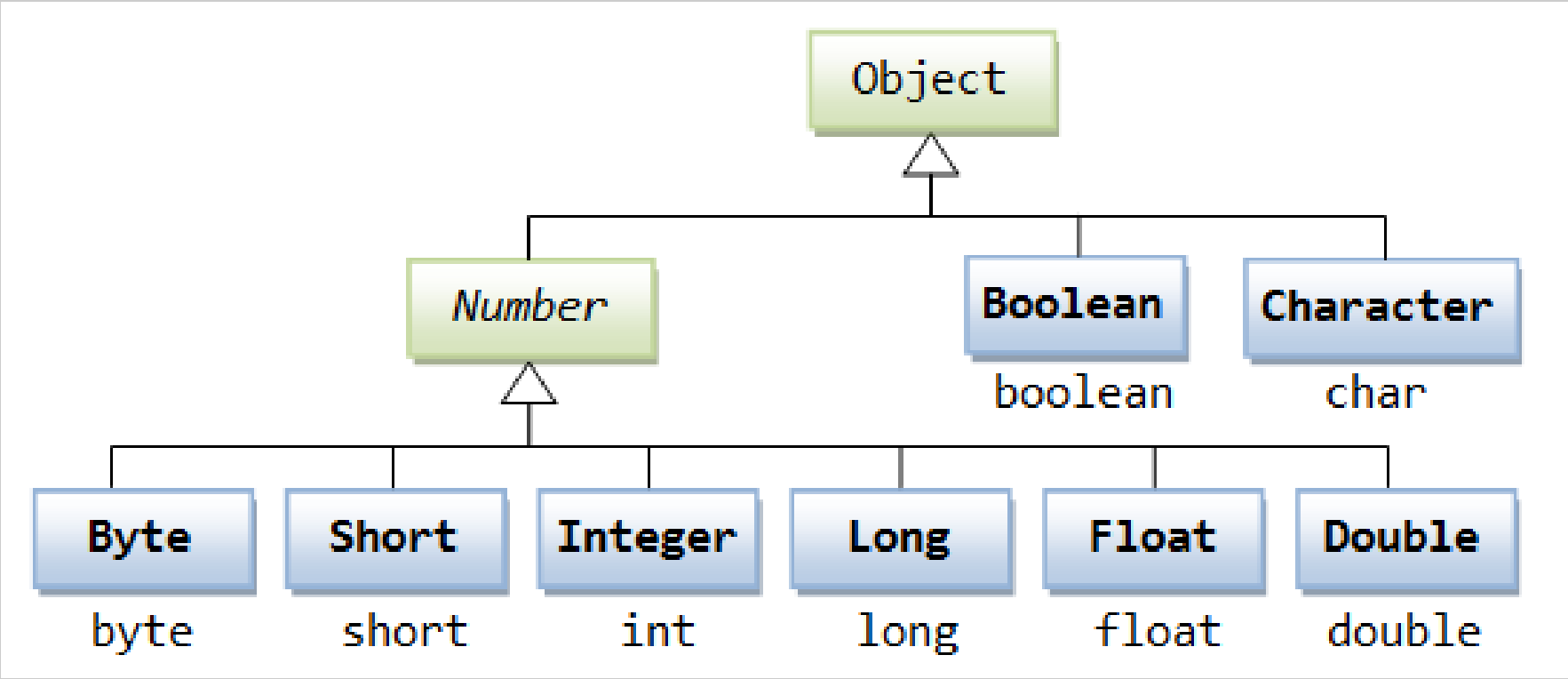
```
public StringBuilder insert(int ind, String s)
```

  - Adiciona uma String na posição indicada por ind

```
1 StringBuilder sb2 = new StringBuilder(100);
2 sb2.append("rova");
3 sb2.insert(0, 'p'); // "prova"
```

1.4 Java Wrappers

- Existem classes para cada um dos tipos primitivos em Java



1.4 Java Wrappers

- A classe java.lang.Integer representa um inteiro e possui duas constantes
  - MIN_VALUE
  - MAX_VALUE
- E dois construtores

```
public Integer(int valor)
public Integer(String valor)
```

1.4 Java Wrappers

- O código abaixo constroi dois objetos

```
Integer i1 = new Integer(12);
Integer i2 = new Integer("123");
```

- Os métodos abaixo retornam os tipos primitivos respectivos com o valor armazenado no Integer
  - ByteValue(), doubleValue(), floatValue() etc.
  - Todos são public static

1.4 Java Wrappers

- A classe java.lang.Boolean representa primitivos boolean
- São dois construtores

```
public Boolean(boolean valor)
public Boolean(String valor)
```

```
1 Boolean b1 = new Boolean(false);
2 Boolean b2 = new Boolean("true");
```

1.4 Java Wrappers

- O construtor criará um objeto true quando for passada uma String “true” (com letras minúsculas ou maiúsculas) e false em qualquer outro caso

```
1 Boolean B = new Boolean("tRuE"); //true
2 Boolean C = new Boolean("TrUE"); //true
3 Boolean D = new Boolean("NotTrue"); //false
4 Boolean E = new Boolean("tru"); //false
```

1.4 Java Wrappers

- Pode-se utilizar o método valueOf() para criação de objetos Boolean

```
public static Boolean valueOf(boolean b)
```

```
1 Boolean B = Boolean.valueOf(true);
```

1.4 Java Wrappers

- Você não fará testes com o ==, mas com o método booleanValue()
- Você pode atribuir o valor armazenado no objeto em um boolean com o método booleanValue()

```
1 if (B.booleanValue()){
2 //instruções;
3 }
4 boolean b = B.booleanValue();
```

1.4 Java Wrappers

- A classe Character é utilizada para criar objetos análogos ao tipo primitivo char
- Veja abaixo o construtor e alguns métodos da classe

```
public Character(char ch)

public char charValue()
public static boolean isDigit(char ch )
public static char toLowerCase(char ch )
public static char toUpperCase(char ch )
```

1.4 Java Wrappers

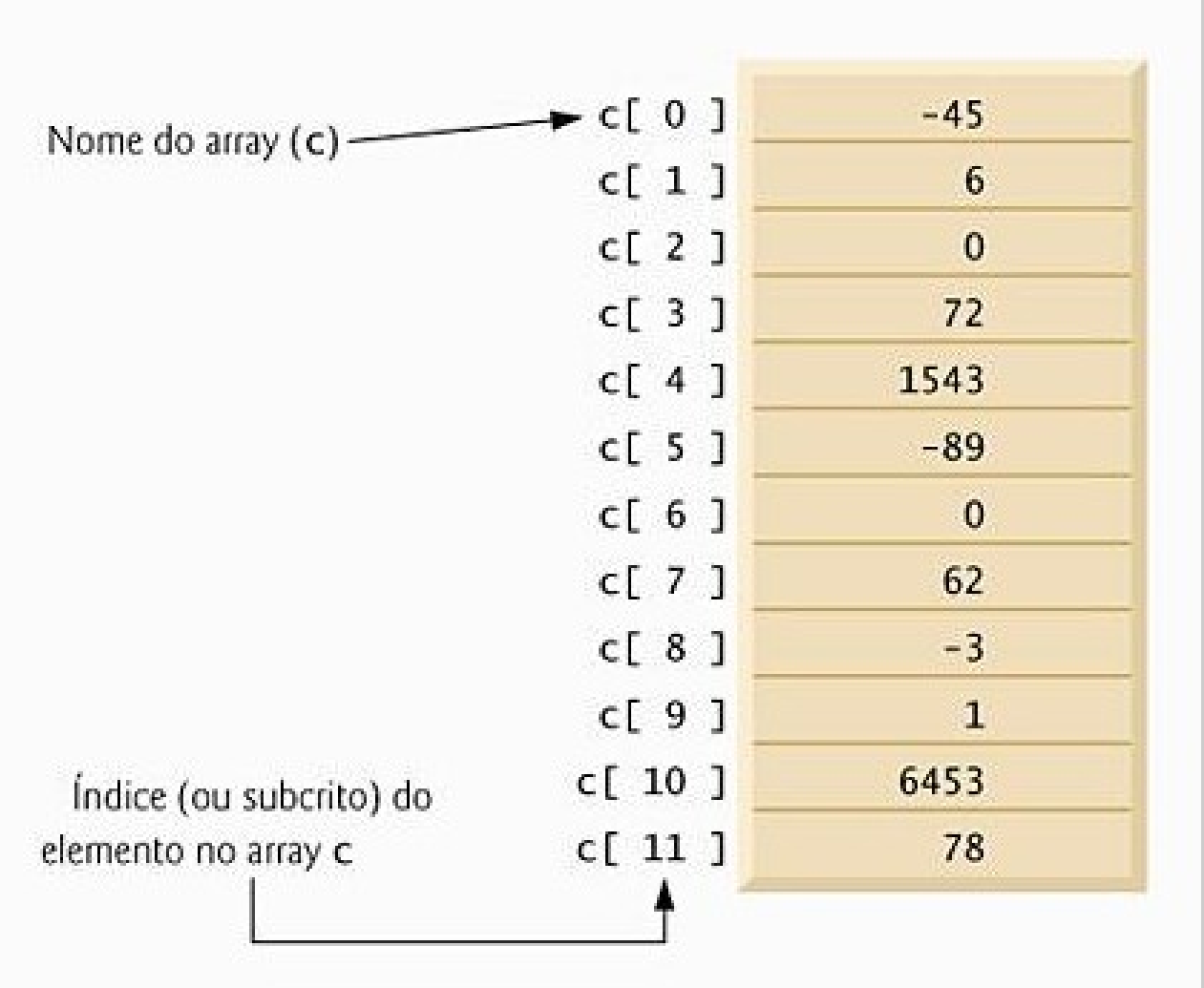
- Encaixotar/desencaixotar (boxing/unboxing) é converter um tipo primitivo para um wrapper e vice-versa

```
1 Integer num = 3; //boxing
2 Integer num2 = new Integer(100);
3 int[] ints = new int[2];
4 ints[0] = num2; //unboxing
```

1.5 Arrays

- Você utiliza arrays (ou listas) para agrupar tipos primitivos ou objetos do mesmo tipo
- Um Array é um objeto que
  - Possui o campo length, com o número de elementos do array
  - Possui um índice especificando cada elemento

1.5 Arrays


Nome do array (c)
c[0]
c[1]
c[2]
c[3]
c[4]
c[5]
c[6]
c[7]
c[8]
c[9]
c[10]
c[11]
-45
6
0
72
1543
-89
0
62
-3
1
6453
78
Índice (ou subcrito) do elemento no array c

1.5 Arrays

- Você utiliza variáveis de referência que apontam para arrays
- Não existe uma classe java.lang.Array ou qualquer coisa parecida
- Para declarar um array

```
tipo[] nome;
tipo nome[];
```

```
1 long[] numeros;
```

1.5 Arrays

- Para criar um objeto do tipo array

```
new tipo[tamanho]
```

```
1 new int[4]
```

- Pode fazer tudo na mesma linha

```
1 int[] ints = new int[4];
```

- Daí você inicializa cada elemento do array

```
1 ints[0] = 10;
```

1.5 Arrays

- Você também pode declarar, criar o array e inicializar seus elementos, tudo na mesma linha

```
1 String[] nomes = {"João", "José", "Ana"};
2 int[] numeros = {1, 2, 3, 10};
```

1.5 Arrays

- Para iterar por um array, a gente precisa de um for

```
1 String[] nomes = {"João", "José", "Ana"};
2 for (int i = 0; i < 3; i++) {
3     System.out.println(nomes[i]);
4 } //João José Ana
5 //ou
6 for (String nome : nomes) {
7     System.out.println(nome);
8 }
```

1.5 Arrays

- Uma vez criado, o array não pode ter seu tamanho mudado
- A saída mais elegante para aumentar o tamanho de um array é usar o copyOf
- O copyOfRange é outro método parecido

```
1 int[] ints = { 1, 2, 3 };
2 int[] novoArray = Arrays.copyOf(ints, 4);
3 int[] int2 = Arrays.copyOfRange(ints, 1, 2);
```

1.5 Arrays

- Relembrando a assinatura de main

```
public static void main(String[] args)
```

- Você pode passar argumentos para o main via linha de comando ao executar o programa, de acordo com a sintaxe

```
java NomedaClasse arg1 arg2 arg3 argn
```

1.5 Arrays

- O programa abaixo mostra na tela os argumentos passados via linha de comando para dentro do main

```
1 public class MainTeste {
2    public static void main(String[] args) {
3       for (String arg : args) {
4          System.out.println(arg);
5       }}}
```

1.6 java.lang.Class

- Para cada objeto criado pela JVM, ela cria um objeto da classe Class, que descreve esse objeto
- Todas as instâncias da mesma classe, compartilham o mesmo objeto da classe Class
- O método getClass retorna esse objeto

```
1 String nome = "Java";
2 Class myClass = nome.getClass();
3 System.out.println(myClass.getName());
```

1.7 java.lang.System

- Classe com uma série de métodos estáticos úteis
- Ela possui três atributos

```
public static final java.io.PrintStream out;
public static final java.io.InputStream in;
public static final java.io.PrintStream err;
```

1.7 java.lang.System

- Exemplo

```
01 System.out.print(12);
02 System.err.println("Tela Azul");
03 StringBuilder sb = new StringBuilder();
04 try {
05     char c = (char) System.in.read();
06     while (c != '\r') {
07         sb.append(c);
08         c = (char) System.in.read();}
09     }
10 catch (IOException e) {}
```

1.7 java.lang.System

- Métodos de System

```
public static void exit(int status)
```

  - Termina o programa e a JVM
  - Um 0 no status significa um fim normal
  - Outro valor pode implicar problema

1.7 java.lang.System

- Métodos de System

```
public static long currentTimeMillis()
public static long nanoTime()
```

  - Retorna um long com o tempo atual em milisegundos ou nanosegundos

```
1 long start = System.currentTimeMillis();
2 //instruções
3 long end = System.currentTimeMillis();
4 System.out.print("Tempo:" + (end - start));
```

1.7 java.lang.System

- Métodos de System

```
public static String getProperty(String key)
public static void setProperty(String p,
String n)
```

  - Retorna ou altera um valor armazenado em determinada propriedade do “sistema”

```
1 System.getProperty("user.dir");
2 System.setProperty("password", "tarzan");
3 System.getProperty("password")
```

1.8 java.util.Scanner

- Usamos a classe Scanner para trazer textos da entrada do usuário
- Para fazer isso é simples

```
1 Scanner scanner = new Scanner(System.in);
2 String s = scanner.next();
```

1.9 Varargs

- Varargs possibilita a existência de métodos com número variável de argumentos
- Sem varargs, você precisa colocar os argumentos dentro de um array
- A sintaxe é um … do lado do tipo

1.9 Varargs

- Exemplo:

```
1 public class MainTeste2 {
2   public static void main(String... args) {
3       for (String arg : args) {
4         System.out.println(arg);
5       }
6   }
7 }
```

1.9 Varargs

- Uso do format, no lugar do println:

```
1 String nome = "Vitor";
2 String sn = "Almeida";
3 System.out.format("Nome: %s %s",nome, sn);
4 System.out.println("Nome: " + nome + " " +
  sn);
```

1.10 Questões de Concurso

- 1 Cesgranrio BR 2011 Analise o código de um programa Java a seguir.

```
public class TestaArgs {
public static void main(String[] args) {
  System.out.println(args[5]);
    }
}
```

Considere o seguinte comando:

java –hotspot TestaArgs um dois três quatro cinco seis sete

O que será impresso pelo programa ao executar esse comando?

(A) dois (B) três (C) quatro (D) cinco (E) seis

1.10 Questões de Concurso

- 1 Cesgranrio BR 2011 Analise o código de um programa Java a seguir.

```
public class TestaArgs {
public static void main(String[] args) {
  System.out.println(args[5]);
    }
}
```

Considere o seguinte comando:

java –hotspot TestaArgs um dois três quatro cinco seis sete

O que será impresso pelo programa ao executar esse comando?

(A) dois (B) três (C) quatro (D) cinco (E) seis

1.10 Questões de Concurso

- 2 Cesgranrio PETROBRAS 2011 Entre outros métodos da linguagem Java, o método pertencente à Classe String que remove espaços em branco existentes no início ou no final de uma string é o

(A) abs()

(B) trim()

(C) exit()

(D) load()

(E) random()

1.10 Questões de Concurso

- 2 Cesgranrio PETROBRAS 2011 Entre outros métodos da linguagem Java, o método pertencente à Classe String que remove espaços em branco existentes no início ou no final de uma string é o

(A) abs()

(B) trim()

(C) exit()

(D) load()

(E) random()

1.10 Questões de Concurso

- 3 CESPE TRT-RN 2010 O Java oferece uma biblioteca de classes predefinidas (APIs do Java), entre elas o pacote java.lang que, por sua utilidade, deve ser importado pelo programador no código, antes de as classes serem declaradas.

1.10 Questões de Concurso

- 3 CESPE TRT-RN 2010 O Java oferece uma biblioteca de classes predefinidas (APIs do Java), entre elas o pacote java.lang que, por sua utilidade, deve ser importado pelo programador no código, antes de as classes serem declaradas.

1.10 Questões de Concurso

- 4 CESPE MPU 2010

```
1 public class MpuJava2 {
2  public static void main(String  args[]){
3    Integer i = null;
4    int j = i;
5    System.out.println(j);}}
```

O código da linha 5 produzirá a impressão do conteúdo da variável j, que terá o valor null.

1.10 Questões de Concurso

- 4 CESPE MPU 2010

```
1 public class MpuJava2 {
2  public static void main(String  args[]){
3    Integer i = null;
4    int j = i;
5    System.out.println(j);}}
```

O código da linha 5 produzirá a impressão do conteúdo da variável j, que terá o valor null.

1.10 Questões de Concurso

- 5 CESPE MPU 2010 Na linguagem Java, um objeto do tipo Integer pode receber valor nulo, porém uma variável primitiva int não pode.

1.10 Questões de Concurso

- 5 CESPE MPU 2010 Na linguagem Java, um objeto do tipo Integer pode receber valor nulo, porém uma variável primitiva int não pode.

1.10 Questões de Concurso

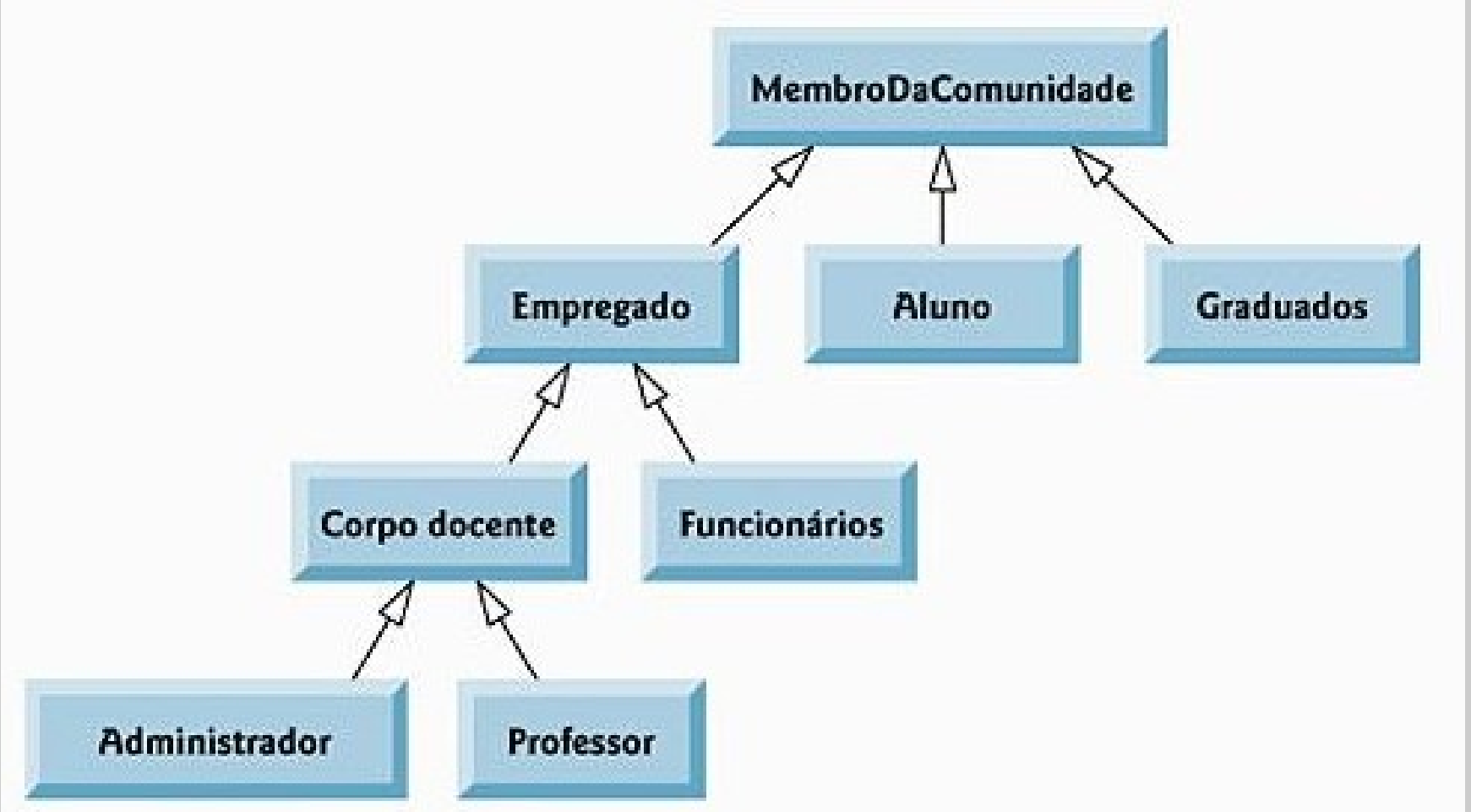
- Gabarito
  - 1 E
  - 2 B
  - 3 E
  - 4 E
  - 5 C

2 Herança

2.1 Introdução

- Você estende uma classe, criando uma nova classe
- Cria-se, então, uma relação mãe-filha entre as classes
- Esse processo de estender classes é chamado de herança
- Quando você estende uma classe, cria-se um relacionamento IS-A

2.1 Introdução



2.1 Introdução

- Você usa extend para criar subclasses

```
1 public class Mae {
2 }
3 public class Filha extends Mae {
4 }
```

2.1 Introdução

- Exemplo:

```
01 class Animal {
02     public float peso;
03     public void comer(){}}
04 class Passaro extends Animal {
05     public int asas = 2;
06     public void voar(){}}
07 class Peixe extends Animal {
08     public int barbatanas = 2;
09     public void nadar(){}}
10 class Cachorro extends Animal {
11     public int patas = 4;
12     public void andar(){}}
```

2.1 Introdução

- Exemplo2:

```
1 Cachorro cachorro = new Cachorro();
2 cachorro.comer();
3 Animal animal = new Passaro();
```

2.1 Introdução

- Uma classe declarada com a palavra final vira o final da hierarquia de classes, não pode ser estendida

```
1 public final class Xpto{};
2 final public class Xpto{};
```

2.2 Controle de Acesso

Nível de acesso	Classes de outros pacotes	Classes do mesmo pacote	Subclasses	Mesma classe
public	sim	sim	sim	sim
protected	não	sim	sim	sim
default	não	sim	não	sim
private	não	não	não	sim

2.2 Controle de Acesso

- Exemplo

```
01 package teste;
02 public class P {
03     public void metodoPublico(){}
04     protected void metodoProtegido(){}
05     void metodoDefaut(){}}
06 class C extends P {
07     public void testeMetodos(){
08         metodoPublico();
09         metodoProtegido();
10         metodoDefault();}}
```

2.3 Sobrescrita de Métodos

- Quando você estende uma classe, os comportamentos dos métodos da superclasse podem ser mudados
- A assinatura do método da subclasse é idêntica à assinatura da superclasse
- As regras de controle de acesso devem ser levadas em consideração na sobrescrita de métodos

2.3 Sobrescrita de Métodos

- Exemplo

```
01 public class Caixa {
02 public int larg, alt, peso;
03 public Caixa(int larg, int alt, int peso){
04     this.larg = larg;
05     this.alt = alt;
06     this.peso = peso;}
07 public String toString() {
08     return "Eu sou uma caixa.";}
09 public Object clone() {
10     return new Caixa(1, 1, 1);}}
```

2.4 super

- Quando você instancia uma classe, todos os construtores das superclasses, até chegar em Object, são chamados

```
01 class Base {
02     public Base() {
03         System.out.println("Base");}
04     public Base(String s) {
05         System.out.println("Base." + s);}}
06 public class Sub extends Base {
07     public Sub(String s) {
08         System.out.println(s);}
09     public static void main(String[] args){
10         Sub sub = new Sub("Start");}}
```

2.4 super

- O que aconteceu foi que Java mudou silenciosamente o construtor da subclasse

```
01 public Sub(String s) {
02     super();
03     System.out.println(s);
04 }
```

- super é uma instância da superclasse

2.4 super

- Você pode chamar explicitamente o super no seu construtor para escolher um construtor específico da superclasse

```
01 public Sub(String s) {
02     super(s);
03     System.out.println(s);
04 }
```

- super deve ser a primeira instrução do construtor

2.4 super

- super também pode ser utilizado para chamar um método sobrescrito da superclasse

```
01 class MatDidatico {
02    public String toString() {
03       return "Material Didático";}}
04 public class Lapis extends MatDidatico {
05    public String toString() {
06       return "Eu sou um lápis";}
07    public void escrever() {
08       System.out.println(super.toString());
09       System.out.println(toString());}
10    public static void main(String[] args){
11       Lapis lapis = new Lapis();
12       lapis.escrever();}}
```

2.5 Casting de Objetos

- Você pode fazer casting de uma subclasse para a superclasse (upcasting)
- E depois fazer o downcasting

```
1 Subc subc = new Subc();
2 Superc superc = subc;
3 Subc subc2 = (Subc) superc;
```

```
1 //erro de compilação
2 Superc superc = new Superc();
3 Subc subc = (Subc) superc;
```

2.6 instanceof

- O operador instanceof verifica se uma variável de referência aponta para um objeto de determinada classe (ou superclasse)
- Quando a variável de referência é null, o resultado é false

```
1 String s = "Hello";
2 if (s instanceof java.lang.String) //true
```

```
1 String s = null;
2 if (s instanceof java.lang.String) //false
```

2.7 Questões de Concurso

- 1 CESPE 2012 BASA O operador instanceof só pode ser usado para testar valores null.

2.7 Questões de Concurso

- 1 CESPE 2012 BASA O operador instanceof só pode ser usado para testar valores null.

2.7 Questões de Concurso

- 2 CESPE 2009 Detran-DF A implementação de herança múltipla em Java não é possível.

2.7 Questões de Concurso

- 2 CESPE 2009 Detran-DF A implementação de herança múltipla em Java não é possível.

2.7 Questões de Concurso

- 3 CESPE 2009 TRT-ES O código a seguir, caso fosse inserido entre as linhas 20 e 21, permitiria criar uma classe que herdasse as características da classe Bicycle.

```
class MountainBike inherit Bicycle {
// inserir nesse ponto novos campos e métodos
// relativos a uma mountain bike.
}
```

2.7 Questões de Concurso

- 3 CESPE 2009 TRT-ES O código a seguir, caso fosse inserido entre as linhas 20 e 21, permitiria criar uma classe que herdasse as características da classe Bicycle.

```
class MountainBike inherit Bicycle {
// inserir nesse ponto novos campos e métodos
// relativos a uma mountain bike.
}
```

2.7 Questões de Concurso

- 4 CESPE 2009 TRT-ES A palavra super deve ser a primeira do construtor, pois o início da construção do objeto é a construção da parte da superclasse. Caso não esteja presente, está implícita uma invocação para o construtor padrão, sem argumentos, da superclasse, equivalente à forma super( ).

2.7 Questões de Concurso

- 4 CESPE 2009 TRT-ES A palavra super deve ser a primeira do construtor, pois o início da construção do objeto é a construção da parte da superclasse. Caso não esteja presente, está implícita uma invocação para o construtor padrão, sem argumentos, da superclasse, equivalente à forma super( ).

2.7 Questões de Concurso

- 5 CESPE 2009 ANAC Na programação orientada a objetos, o conceito de herança pode ser utilizado mediante a criação de subclasses a partir de classes anteriormente criadas. Em Java, as subclasses herdam as variáveis de instância e os métodos de instância da superclasse, podendo ter acesso a todos os membros private e protected da referida superclasse.

2.7 Questões de Concurso

- 5 CESPE 2009 ANAC Na programação orientada a objetos, o conceito de herança pode ser utilizado mediante a criação de subclasses a partir de classes anteriormente criadas. Em Java, as subclasses herdam as variáveis de instância e os métodos de instância da superclasse, podendo ter acesso a todos os membros private e protected da referida superclasse.

2.7 Questões de Concurso

- Gabarito
  - 1 E
  - 2 C
  - 3 E
  - 4 C
  - 5 E

3 Tratamento de Exceções

3.1 Tratando Exceções

- Java oferece o tratamento de exceções por meio da instrução try
- A estratégia é envolver um pedaço de código com potencial para lançar uma exceção dentro de um bloco
- As instruções catch e finally acompanham o try

3.1 Tratando Exceções

- Se uma exceção ocorre dentro do bloco try, a JVM passa o processamento imediatamente para o bloco catch adequado
- Dentro do bloco catch você trata o erro.
- Caso esteja dentro de um método, é possível lançar a exceção para quem chamou o método

3.1 Tratando Exceções

- Sintaxe

```
01 try {
02    //código “problemático”
03 }catch (exceçãotipo1  e) {
04    //código para tratar a exceção tipo 1
05 }catch (exceçãotipo2  e) {
06    //código para tratar exceção tipo 2
07 }
08 ...
09 finally {
10    //código que roda com ou sem exceção
11 }
```

3.1 Tratando Exceções

- Exemplo

```
01 import java.util.Scanner;
02 public class NumDouble {
03  public static void main(String[] args) {
04   Scanner scanner = new Scanner(System.in);
05   String input = scanner.next();
06   try {
07    double num = Double.parseDouble(input);
08    System.out.printf("Res: " + num);}
09   catch (NumberFormatException e) {
10    System.out.println("Entrada Inválida");
11   }}}
```

3.1 Tratando Exceções

- Exemplo 2

```
1 Connection connection = null;
2 try {
3    //abrir conexão
4    //trabalhar na conexão
5 }
6 finally {
7    if (connection != null){
8       //fechar conexão
9    }}
```

3.1 Tratando Exceções

- Exemplo 3

```
1 try {
2     serverSocket.accept();
3 }
4 catch (SocketTimeoutException |
  SecurityException |
  IllegalBlockingModeException e){
5     //tratar exceções
6 }
7 catch (IOException e){
8     //tratar IOException
9 }
```

3.1 Tratando Exceções

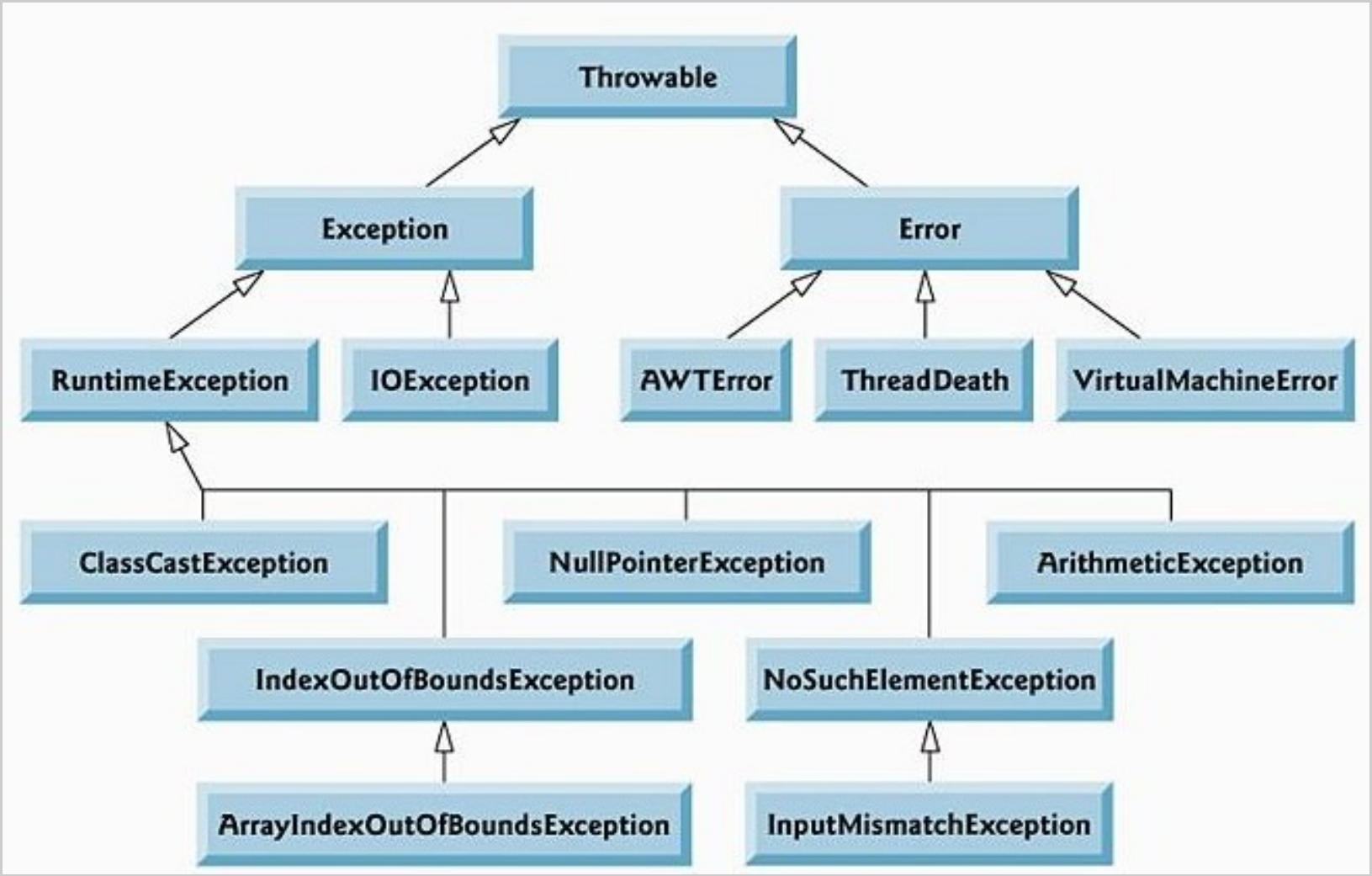
- Java 7 criou um try “com recursos”

```
1 Connection connection = null;
2 try (Connection connection =
  openConnection();
  //abrir outros recursos) {
3     //trabalhar com as conexões
4 }
5 catch (SQLException e) {
6     //Tratar exceção
7 }
```

3.2 java.lang.Exception

- Todas as exceções, como java.lang.NumberFormatException ou java.lang.NullPointerException, são subclasses de java.lang.Exception

3.2 java.lang.Exception



3.2 java.lang.Exception

- Métodos de Exception

```
public String toString()
public void printStackTrace()
```

  - Retorna, respectivamente, uma descrição da exceção e a stack trace.

```
java.lang.NullPointerException
    at MathUtil.doubleNumber(MathUtil.java:45)
    at CA.metodo(MyClass.java: 18)
    at CA.main(MyClass.java: 90)
```

3.2 java.lang.Exception

- É normal ter um catch no final para Exception

```
1 try {
2    //instruções
3 }
4 catch (NumberFormatException e) {
5    //trata NumberFormatException
6 }
7 catch (Exception e) {
8    //trata outras exceções
9 }
```

3.3 Exceções e Métodos

- Quando uma exceção é lançada dentro de um método existem duas possibilidades
  - Tratar a exceção dentro do método sem incomodar quem chama o método
  - Lançar a exceção para o chamador tratá-la

3.3 Exceções e Métodos

- Para dizer que um método pode lançar uma exceção, usamos o throws

```
1 public String capitalizar(String s) throws
  NullPointerException {
3     if (s == null) {
4         throw new NullPointerException(
          "Você passou uma String null");
5     }
6     Character prim = s.charAt(0);
7     String resto = s.substring(1);
8     return prim.toString().toUpperCase() +
      resto;
9 }
```

3.3 Exceções e Métodos

- O exemplo abaixo mostra o uso de capitalizar()

```
1 String input = null;
2 try {
3    String cap = util.capitalizar(input);
4    System.out.println(cap);
5 }
6 catch (NullPointerException e) {
7    System.out.println(e.toString());
8 }
```

3.4 Criando Exceções

- Você pode ter suas próprias exceções criando subclasses de Exception

```
1 public class JaMaiusculaException
  extends Exception {
2     public String toString() {
3         return "Já tá maiúsculo!";
4     }
5 }
```

3.4 Criando Exceções

- Nosso método com a nossa exceção

```
1 public String capitalizar(String s)
  throws NullPointerException,
  JaMaiusculaException {
2    if (s == null) {
3       throw new NullPointerException(
        "String null não pode");}
4    Character prim = s.charAt(0);
5    if (Character.isUpperCase(prim)) {
6      throw new JaMaiusculaException();}
7    String resto = s.substring(1);
8    return prim.toString().toUpperCase() +
     resto;}
```

3.5 Questões de Concurso

- 1 CESPE TRE-MA 2010 Para definição e manipulação de uma exceção em Java, devem constar no programa, obrigatoriamente, os termos

a) try e catch.

b) try e finally.

c) finally e catch.

d) finally e retry.

e) try e retry.

3.5 Questões de Concurso

- 1 CESPE TRE-MA 2010 Para definição e manipulação de uma exceção em Java, devem constar no programa, obrigatoriamente, os termos

a) try e catch.

b) try e finally.

c) finally e catch.

d) finally e retry.

e) try e retry.

3.5 Questões de Concurso

- 2 CESPE Embasa 2010 O trecho de código a seguir está sintaticamente incorreto.

```
If (ChatUser.GetByName(getName()) != null {
  throw new IllegalArgumentException(“invalid username”);
} else {
   ChatUser.addUser(this);}
```

3.5 Questões de Concurso

- 2 CESPE Embasa 2010 O trecho de código a seguir está sintaticamente incorreto.

```
If (ChatUser.GetByName(getName()) != null {
  throw new IllegalArgumentException(“invalid username”);
} else {
   ChatUser.addUser(this);}
```

3.5 Questões de Concurso

- 3 Cesgranrio CMB 2012 A exceção comum StackOverflowException, em uma applet, é causada quando a(o)

a) applet tenta executar uma ação não permitida pela configuração do browser.

b) applet tenta armazenar um tipo de dado incorreto em um array.

c) conversão entre strings e números gera uma falha.

d) espaço de pilha do sistema se esgota.

e) espaço de memória para alocar um novo objeto se esgota

3.5 Questões de Concurso

- 3 Cesgranrio CMB 2012 A exceção comum StackOverflowException, em uma applet, é causada quando a(o)

a) applet tenta executar uma ação não permitida pela configuração do browser.

b) applet tenta armazenar um tipo de dado incorreto em um array.

c) conversão entre strings e números gera uma falha.

d) espaço de pilha do sistema se esgota.

e) espaço de memória para alocar um novo objeto se esgota

3.5 Questões de Concurso

- 4 FCC PGE-RJ 2009 Um exemplo de exceção incluída na linguagem Java, que indica quando uma aplicação tentou usar uma referência a um objeto que não foi identificado, é

a) ClassNotFoundException.

b) NullPointerException.

c) ArithmeticException.

d) NumberFormatException.

e) IndexOutOfBound.

3.5 Questões de Concurso

- 4 FCC PGE-RJ 2009 Um exemplo de exceção incluída na linguagem Java, que indica quando uma aplicação tentou usar uma referência a um objeto que não foi identificado, é

a) ClassNotFoundException.

b) NullPointerException.

c) ArithmeticException.

d) NumberFormatException.

e) IndexOutOfBound.

3.5 Questões de Concurso

- Gabarito
  - 1 A
  - 2 E
  - 3 D
  - 4 B