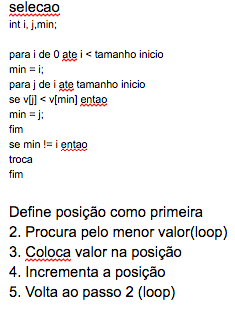
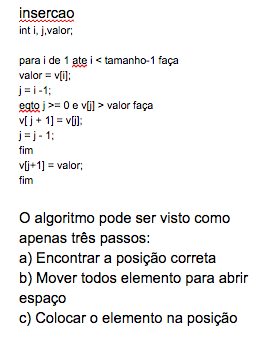
Algoritmo de bolha melhorado

int i = 0, j;

int k = tamanho - 1;

boolean houveTroca = true;

enquanto houveTroca && i < tamanho faca

inicio

i++;

houveTroca = false;

para j de 0 ate j < k inicio

se v[j] > v[j+1] entao inicio

troca;

houveTroca = true;

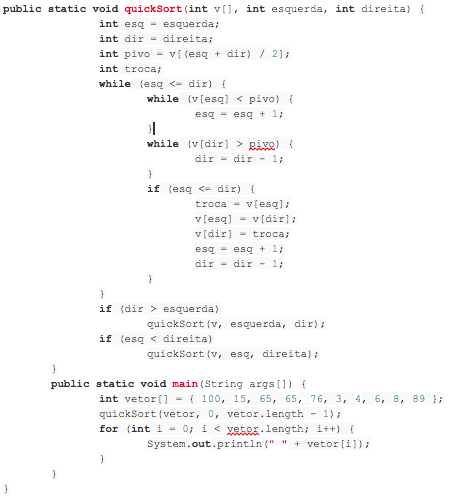
fim

fim

k--;

fim

Define posição como primeira

2. Procura pelo menor valor(loop)

3. Coloca valor na posição

4. Incrementa a posição

5. Volta ao passo 2 (loop)

Quick

Use a mediana do array (difícil de encontrar)

• Compare apenas 3 elementos

• O primeiro,

• O último,

• O Central

• Para estes 3 elementos garanta que

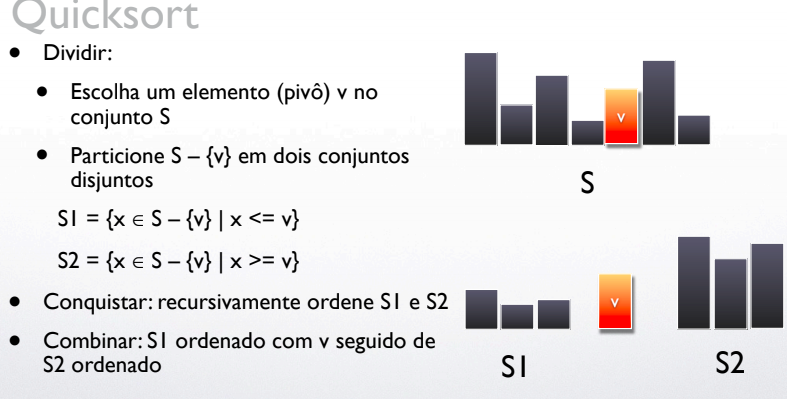
• Na primeira posição ficará o menor dos 3

• Na última posição ficará o maior dos 3

• E no meio aquele que será usado como pivô

Usar um array adicional

• Semelhante ao merge

• Fácil de programar

• Ineficiente para o Quick

Merge

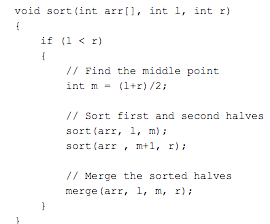
1. Problema pequeno, resolva o problema

2. Dividir o problema em dois ou mais problemas

menores e similares

3. Recursão resolva o problema

4. Conquistar, reuna as soluções parciais unindo as



seqüências

public static void main(String args[])

{

int arr[] = {12, 11, 13, 5, 6, 7}

System.out.println("Given Array");

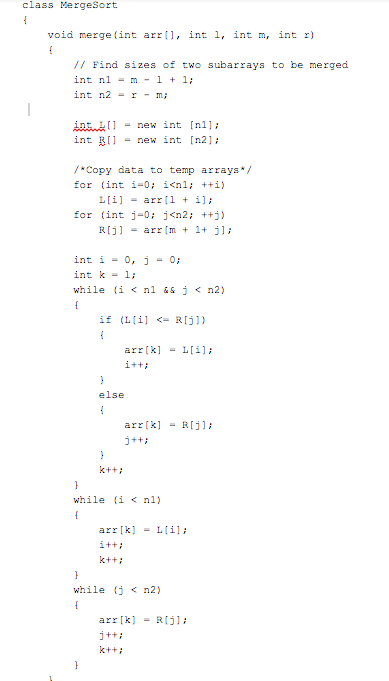
printArray(arr);

MergeSort ob = new MergeSort();

ob.sort(arr, 0, arr.length-1);

}

}

Design Pattern: Descreve como objetos e classes se relacionam

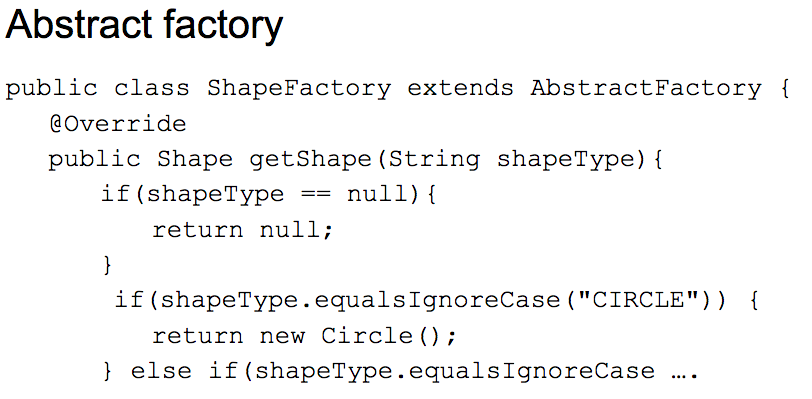
de forma customizada para resolver problema generico;

* Facilita o reuso; Tecnicas de funcionamento comprovado;

ajuda a esoclher um caminho; documentacao e caminho;

Comand: Armazena algoritmo ou acao em uma aplicação;

Abstract Factory: Fabrica de fabricas;



Decorator: Esconde as estruturas internas de um conjunto de objeto. ex: interface;

FACADE: Esconde a estrutura interna. ex: dois objetos passam pela mesma classe e armazena la.

Adapter: Uma classe e adaptada para enviar dados para qlqr outra classe;

Singleton: Garante uma unica instancia de uma classe.

public class ClassicSingleton {

private static ClassicSingleton **instance** = null;

**private** ClassicSingleton() {} // Por quê?

public static ClassicSingleton getInstance() {

if(**instance** == null)

**instance** = new ClassicSingleton();

return instance;

}

}

Builder:

Separa tudo, cada coisa no seu lugar;