O método `Arrays.compare()` do pacote *jdk.util* em Java é usado para comparar dois arrays de objetos usando a ordem lexicográfica dos elementos. Aqui está uma explicação detalhada do seu funcionamento:

1. **Assinatura do Método:**
   - A assinatura do método é `public static <T extends Comparable<? super T>> int compare(T[] a, T[] b)`.
   - Ele aceita dois arrays de objetos do tipo `T`, onde `T` é um tipo que estende `Comparable<T>`.
2. **Comparação dos Tamanhos dos Arrays:**

O método `compare` da classe `Arrays` compara os elementos dos arrays `a` e `b` lexicograficamente, isto é, elemento por elemento da esquerda para a direita:

1. Se os arrays `a` e `b` forem exatamente iguais em tamanho e contiverem os mesmos elementos na mesma ordem, o método retornará `0`.
2. Se `a` e `b` forem diferentes em algum ponto:
   - O método compara o primeiro elemento diferente dos dois arrays.
   - Se o elemento de `a` for menor que o elemento correspondente de `b`, ele retorna um valor negativo (tipicamente -1).
   - Se o elemento de `a` for maior que o elemento correspondente de `b`, ele retorna um valor positivo (tipicamente 1).
3. Se um array for um prefixo do outro (ou seja, todos os elementos de um array são iguais aos elementos correspondentes do início do outro array, mas um dos arrays é mais curto):
   - O método `compare` retorna a diferença entre os tamanhos dos arrays (`a.length - b.length`).

Parece meio confuso né? Vamos, como exemplo, estudar diretamente o código fonte encontrado em *java.util* sobrecarregado para os elementos do tipo int:

```
1  public static int compare(int[] a, int[] b) {
2      if (a == b)
3          return 0;
4      if (a == null || b == null)
5          return a == null ? -1 : 1;
6      int i = ArraysSupport.mismatch(a, b,
7                                     Math.min(a.length, b.length));
8      if (i >= 0) {
9          return Integer.compare(a[i], b[i]);
10     }
11     return a.length - b.length;
12 }
```

Dado as variáveis do tipo *Array* abaixo:

```
int[] a3 =null;
int[] b3 = null;
```

```
int[] a4 =null;
int[] b4 = {};

int[] a5 = {1, 2, 5, 6};
int[] b5 = {1, 2, 5, 6,7,8};

int[] a6 = {1, 2, 5, 6};
int[] b6 = {1, 3, 5, 6,7,8};

int[] a7 = {1, 2, 5, 6};
int[] b7 = {1, 2, 3, 6};

int[] a8 = {1, 2, 5, 6};
int[] b8 = {1, 2, 5, 6};
```

E o retorno de `Arrays.compare()` para cada caso:

1. `int[] a3 = null; int[] b3 = null; Arrays.compare(a3, b3);`
   - Ambos os arrays são nulos, então eles são considerados iguais. O resultado será `0`. Veja linhas `1` e `2` do código-fonte.
2. `int[] a4 = null; int[] b4 = {}; Arrays.compare(a4, b4);`
   - `a4` é nulo e `b4` está vazio. Neste caso, um array vazio é considerado maior que um array nulo. Portanto, o resultado será `1`. Veja linhas `4` e `5` do código-fonte,quando um dos arrays for *null*. O *null* sempre será menor que um array não *null*.
3. `int[] a4 = null; int[] b4 = {}; Arrays.compare(b4, a4);`
   - Similar ao caso anterior, mas a ordem dos argumentos foi invertida. Como um array vazio é considerado maior que um array nulo, o resultado será `-1`.
4. `int[] a5 = {1, 2, 5, 6}; int[] b5 = {1, 2, 5, 6, 7, 8}; Arrays.compare(a5, b5);`
   - Veja as linhas `6` e `7` do código-fonte, o método `mismatch(int[] a, int[] b, tamanho)` do pacote *jdk.internal.util* retorna -1 se os arrays forem iguais, ou um valor entre 0 (inclusive) e o tamanho (exclusive) do MENOR array indicando o índice onde os elementos dos arrays não são iguais, o método irá comparar apenas os n-esimos valores do array sendo n o tamanho do menor array. Neste caso, apesar de tamanhos diferentes, o método `mismatch()` irá retornar `-1'` **porque o array menor `a5` é exatamente os 4 primeiros elementos de `b5`, logo `a5` é prefixo de `b5`**. Dessa forma o retorno da função será pela linha `11` que retorna a diferença de tamanhos dos vetores `a5` e `b5`, o valor resultante portanto será `-2`.
5. `int[] a5 = {1, 2, 5, 6}; int[] b5 = {1, 2, 5, 6, 7, 8}; Arrays.compare(b5, a5);`
   - Similar ao caso anterior, mas a ordem dos argumentos foi invertida, logo o resultado será 2.
6. `int[] a6 = {1, 2, 5, 6}; int[] b6 = {1, 3, 5, 6,7,8}; Arrays.compare(a6,`

`b6);`

- Note que os arrays são parecidos com o exemplo 4, entretanto os 4 primeiros elementos de `a6` não são os 4 primeiros elementos de `b6`, logo o método `mismatch()` neste caso não irá retornar `-1'` e sim `1` que é o índice do primeiro elemento onde os valores não são iguais para `a6` e `b6`, assim analisando o código fonte o retorno se dará pela linha `9` onde o resultado será o retorno do método `Integer.compare()` e como o valor `a6[1]=2` é menor que `b6[1]=3` será `-1`.

7. `int[] a6 = {1, 2, 5, 6}; int[] b6 = {1, 3, 5, 6,7,8}; Arrays.compare(b6, a6);`
   - Similar ao caso anterior, mas a ordem dos argumentos foi invertida, logo o resultado de `Integer.compare()` será `1`.
8. `int[] a7 = {1, 2, 5, 6}; int[] b7 = {1, 2, 3, 6}; Arrays.compare(a7, b7);`
   - Note que neste caso os vetores tem igual tamanho, e `mismatch()` linhas `6` e `7` irá retornar um valor >=0 pois os arrays não são iguais, neste caso novamente o retorno se dará pelo retorno do método `Integer.compare()` da linha `9`, como `a7[2]=5` e `b7[2]=3` o resultado será `1`.
9. `int[] a7 = {1, 2, 5, 6}; int[] b7 = {1, 2, 3, 6}; Arrays.compare(b7, a7);`
   - Similar ao caso anterior, mas a ordem dos argumentos foi invertida, logo o resultado de `Integer.compare()` será `-1`.
10. `int[] a8 = {1, 2, 5, 6}; int[] b8 = {1, 2, 5, 6}; Arrays.compare(a8, b8);`

- Os arrays possuem os mesmos elementos e na mesma ordem, veja as linhas `6` e `7` do código-fonte, o método `mismatch()` retornará -1, e portanto o retorno se dará pela linha `11`, como os tamanhos são iguais o valor de retorno será `0`.