Sorting an array using an anonymous function

We show how easy it is to sort an array in various ways using anonymous functions.

```
public class Person {
    public String first;
    public String last;
    public int age;
    …
}
```

Check out class `Person` to the right. We have made the fields public only to make our anonymous functions shorter and easier to understand. Generally, they would be private, and getter functions would have to be used.

Suppose we have declared array

```
Person[] p= …;
```

and have filled `p` with a bunch of `Person` objects. We want to sort `p` based on the ages of the Persons. API class `Arrays` contains a procedure that lets us do it with this procedure call[1]:

```
Arrays.sort(p, (Person b, Person c) -> b.age - c.age);
```

The second argument of the call on `sort` is an anonymous function with two `Person` parameters. It returns:

- a negative number if `Person b` is younger than `Person c` (i.e. `b` comes before `c`)
- 0 if `Person b` and `Person c` are the same age (i.e. neither comes before the other)
- a positive number if `Person b` is older than `Person c` (i.e. `c` comes after `b`)

We can leave out the types of `b` and `c`, since they can be inferred. So, we write this call more simply as:

```
Arrays.sort(p, (b, c) -> b.age - c.age);
```

We have flexibility! To sort `p` in reverse order of age, use:

```
Arrays.sort(p, (b, c) -> c.age - b.age);
```

So you see how easy it is to sort `p` in some order; just give a second argument, an anonymous function that indicates which of `b` and `c` comes before the other:

- a negative number if `b` comes before `c`
- 0 if `b` and `c` are the same (i.e. neither comes before the other)
- a positive number if `b` comes after `c`)

Sorting by Strings

Class `String` already has a function `s.compareTo(t)` that returns a negative number, 0, or a positive number depending on whether `String s` is smaller than, equal to, or larger than `String t`. Therefore, to sort `p` on the last name of the people, use

```
Arrays.sort(p, (b, c) -> b.last.compareTo(c.last));
```

More complicated sorting

Suppose we want to sort `p` by first name, but sort people with the same first name by age. Thus, if John Doe is age 9 and John Woe is age 5, put John Woe first.

Putting this all into an anonymous function is messy. Instead, we write static function `before` (look to the right) and write this call on sort:

```
Arrays.sort(p, (b, c) -> before(b, c));
```

```
/** Return a negative (positive) number if b's first
  * name comes before (or after) c's first name.
  * If b's and c's first names are the same,
  * return b's age - c's age. */
public static int before(Person b, Person c) {
    int n= b.first.compareTo(c.first);
    if (n != 0) return n;
    return b.age - c.age;
}
```

[1] Sorting procedures also exist to sort only a portion of an array. For example, to sort only array segment p[1..6], use Arrays.sort(p, 1, 7, (Person b, Person c) -> b.age - c.age);