Sorting with Template Method

What's something we often need to do with arrays? Sort them!

Recognizing that, the designers of the Java Arrays class have provided us with a handy template method for sorting. Let's take a look at how this method operates:

We've pared down this code a little to make it easier to explain. If you'd like to see it all, grab the Java source code and check it out...

We actually have two methods here and they act together to provide the sort functionality.

The first method, sort(), is just a helper method that creates a copy of the array and passes it along as the destination array to the mergeSort() method. It also passes along the length of the array and tells the sort to start at the first element.

```
public static void sort(Object[] a) {
    Object aux[] = (Object[])a.clone();
    mergeSort(aux, a, 0, a.length, 0);
}
```

The mergeSort() method contains the sort algorithm, and relies on an implementation of the compareTo() method to complete the algorithm. If you're interested in the nitty-gritty of how the sorting happens, you'll want to check out the Java source code.

Think of this as the **template method**.

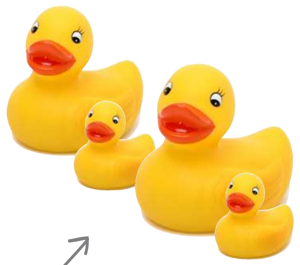
```
private static void mergeSort(Object src[], Object dest[],
                int low, int high, int off)
{
    // a lot of other code here
    for (int i=low; i<high; i++){
        for (int j=i; j>low &&
                ((Comparable)dest[j-1]).compareTo((Comparable)dest[j])>0; j--)
        {
            swap(dest, j, j-1);
        }
    }
    // and a lot of other code here
}
```

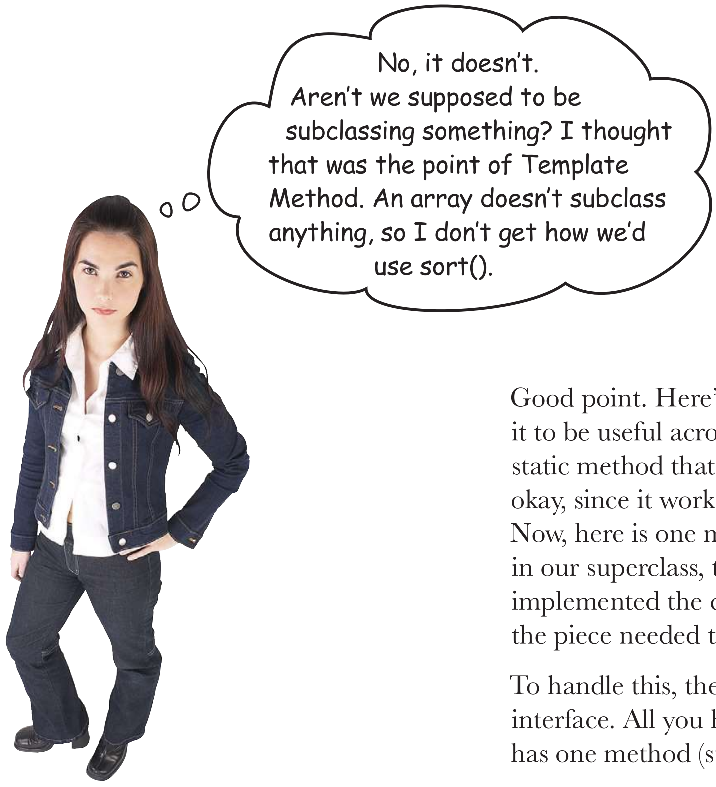
compareTo() is the method we need to implement to "fill out" the template method.

This is a concrete method, already defined in the Arrays class.

We've got some ducks to sort...

Let's say you have an array of ducks that you'd like to sort. How do you do it? Well, the sort() template method in Arrays gives us the algorithm, but you need to tell it how to compare ducks, which you do by implementing the compareTo() method... Make sense?



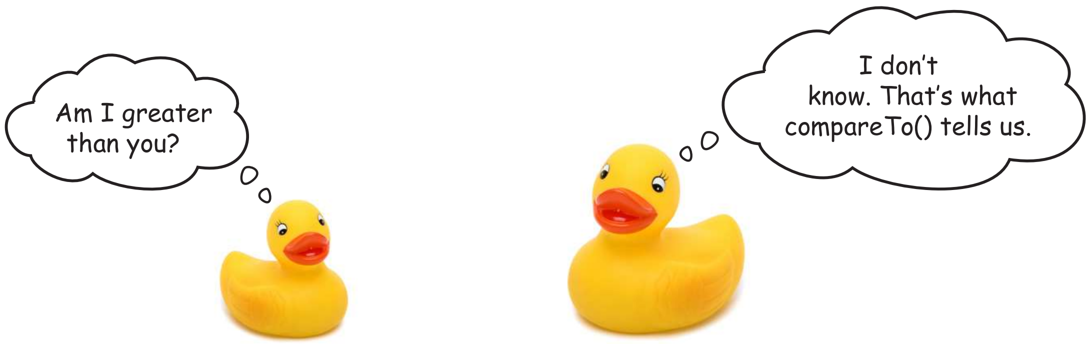

No, it doesn't. Aren't we supposed to be subclassing something? I thought that was the point of Template Method. An array doesn't subclass anything, so I don't get how we'd use sort().

Good point. Here's the deal: the designers of sort() wanted it to be useful across all arrays, so they had to make sort() a static method that could be used from anywhere. But that's okay, since it works almost the same as if it were in a superclass. Now, here is one more detail: because sort() really isn't defined in our superclass, the sort() method needs to know that you've implemented the compareTo() method, or else you don't have the piece needed to complete the sort algorithm.

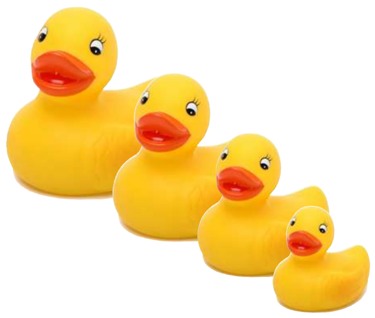
To handle this, the designers made use of the Comparable interface. All you have to do is implement this interface, which has one method (surprise): compareTo().

What is compareTo()?

The compareTo() method compares two objects and returns whether one is less than, greater than, or equal to the other. sort() uses this as the basis of its comparison of objects in the array.



Comparing Ducks and Ducks



Okay, so you know that if you want to sort Ducks, you're going to have to implement this compareTo() method; by doing that, you'll give the Arrays class what it needs to complete the algorithm and sort your ducks.

Here's the duck implementation:

Remember, we need to implement the Comparable interface since we aren't really subclassing.

```
public class Duck implements Comparable<Duck> {
    String name;
    int weight;

    public Duck(String name, int weight) {
        this.name = name;
        this.weight = weight;
    }

    public String toString() {
        return name + " weighs " + weight;
    }


    public int compareTo(Duck otherDuck) {

        if (this.weight < otherDuck.weight) {
            return -1;
        } else if (this.weight == otherDuck.weight) {
            return 0;
        } else { // this.weight > otherDuck.weight
            return 1;
        }
    }
}
```

Our Ducks have a name and a weight.

We're keepin' it simple; all Ducks do is print their name and weight!

Okay, here's what sort() needs...

compareTo() takes another Duck to compare THIS Duck to.

Here's where we specify how Ducks compare. If THIS Duck weighs less than otherDuck, we return -1; if they are equal, we return 0; and if THIS Duck weighs more, we return 1.

Let's sort some Ducks

Here's the test drive for sorting Ducks...

```
public class DuckSortTestDrive {

    public static void main(String[] args) {
        Duck[] ducks = {
                        new Duck("Daffy", 8),
                        new Duck("Dewey", 2),
                        new Duck("Howard", 7),
                        new Duck("Louie", 2),
                        new Duck("Donald", 10),
                        new Duck("Huey", 2)
         };

        System.out.println("Before sorting:");
        display(ducks);

        Arrays.sort(ducks);

        System.out.println("\nAfter sorting:");
        display(ducks);
    }

    public static void display(Duck[] ducks) {
        for (Duck d : ducks) {
            System.out.println(d);
        }
    }
}
```

We need an array of Ducks; these look good.

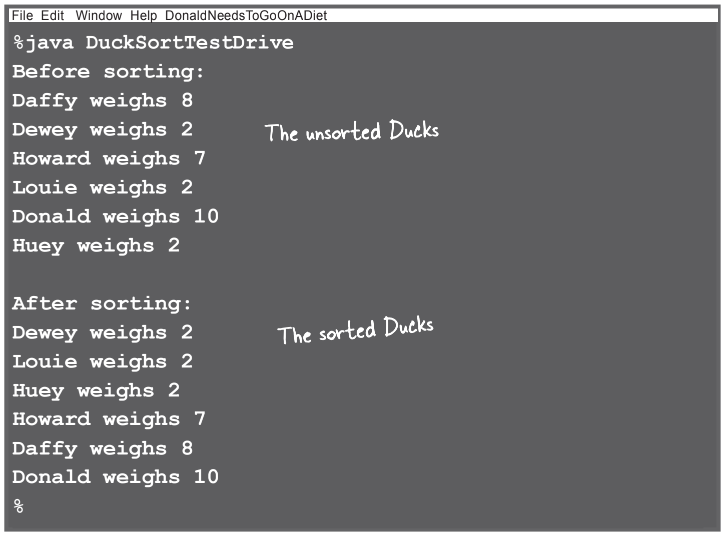
Let's print them to see their names and weights.

Notice that we call Arrays' static method sort(), and pass it our Ducks.

It's sort time!

Let's print them (again) to see their names and weights.

Let the sorting commence!



```
%java DuckSortTestDrive
Before sorting:
Daffy weighs 8
Dewey weighs 2
Howard weighs 7
Louie weighs 2
Donald weighs 10
Huey weighs 2

After sorting:
Dewey weighs 2
Louie weighs 2
Huey weighs 2
Howard weighs 7
Daffy weighs 8
Donald weighs 10
%
```

The making of the sorting duck machine

Behind the Scenes

Let's trace through how the Arrays sort() template method works. We'll check out how the template method controls the algorithm, and at certain points in the algorithm, how it asks our Ducks to supply the implementation of a step...

```
for (int i=low; i<high; i++){
        ... compareTo() ...
        ... swap() ...
}
```

The sort() method controls the algorithm; no class can change this. sort() counts on a Comparable class to provide the implementation of compareTo().

1. First, we need an array of Ducks:

   ```
   Duck[] ducks = {new Duck("Daffy", 8), ... };
   ```

2. Then we call the sort() template method in the Arrays class and pass it our ducks:

   ```
   Arrays.sort(ducks);
   ```

   The sort() method (and its helper, mergeSort()) control the sort procedure.

3. To sort an array, you need to compare two items one by one until the entire list is in sorted order.

   When it comes to comparing two ducks, the sort() method relies on the Duck's compareTo() method to know how to do this. The compareTo() method is called on the first duck and passed the duck to be compared to:

   ```
   ducks[0].compareTo(ducks[1]);
   ```

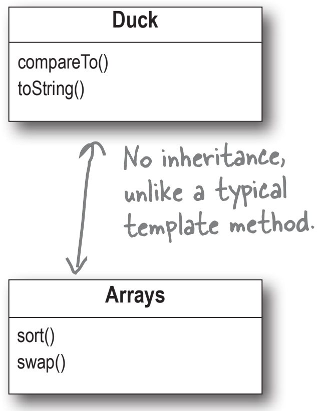
   First Duck

   Duck to compare it to

4. If the Ducks are not in sorted order, they're swapped with the concrete swap() method in Arrays:

   ```
   swap()
   ```

5. The sort() method continues comparing and swapping Ducks until the array is in the correct order!



there are no Dumb Questions

Q: Is this really the Template Method Pattern, or are you trying too hard?

A: The pattern calls for implementing an algorithm and letting subclasses supply the implementation of the steps—and the Arrays sort() is clearly not doing that! But, as we know, patterns in the wild aren't always just like the textbook patterns. They have to be modified to fit the context and implementation constraints.

The designers of the Arrays sort() method had a few constraints. In general, you can't subclass a Java array and they wanted the sort to be used on all arrays (and each array is a different class). So they defined a static method and deferred the comparison part of the algorithm to the items being sorted.

So, while it's not a textbook template method, this implementation is still in the spirit of the Template Method Pattern. Also, by eliminating the requirement that you have to subclass Arrays to use this algorithm, they've made sorting in some ways more flexible and useful.

Q: This implementation of sorting actually seems more like the Strategy Pattern than the Template Method Pattern. Why do we consider it Template Method?

A: You're probably thinking that because the Strategy Pattern uses object composition. You're right in a way—we're using the Arrays object to sort our array, so that's similar to Strategy. But remember, in Strategy, the class that you compose with implements the entire algorithm. The algorithm that Arrays implements for sort() is incomplete; it needs a class to fill in the missing compareTo() method. So, in that way, it's more like Template Method.

Q: Are there other examples of template methods in the Java API?

A: Yes, you'll find them in a few places. For example, java.io has a read() method in InputStream that subclasses must implement and is used by the template method read(byte b[], int off, int len).

BRAIN POWER

We know that we should favor composition over inheritance, right? Well, the implementers of the sort() template method decided not to use inheritance and instead to implement sort() as a static method that is composed with a Comparable at runtime. How is this better? How is it worse? How would you approach this problem? Do Java arrays make this particularly tricky?

BRAIN² POWER

Think of another pattern that is a specialization of the template method. In this specialization, primitive operations are used to create and return objects. What pattern is this?