The Comparable Interface

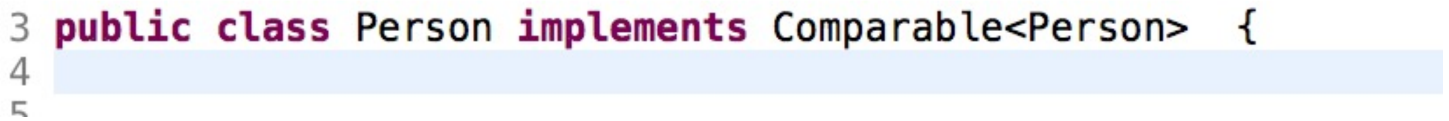
https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

- Define a *compareTo* method to order objects
- *String class defines compareTo*
- *For example if str* and *other* are Strings, *str.compareTo(other)* returns
  - **Negative** if *str* comes **before** *other*
    - *str <- "Virginia"   other <- "Wyoming"*
  - **Zero** if *str* and *other* are **equal**
    - str <- "Virginia"   other <- "Virginia"
  - **Positive** if *str* comes after *other*
    - *str <- "Virginia"   other <- "Alabama"*

The Comparator Interface

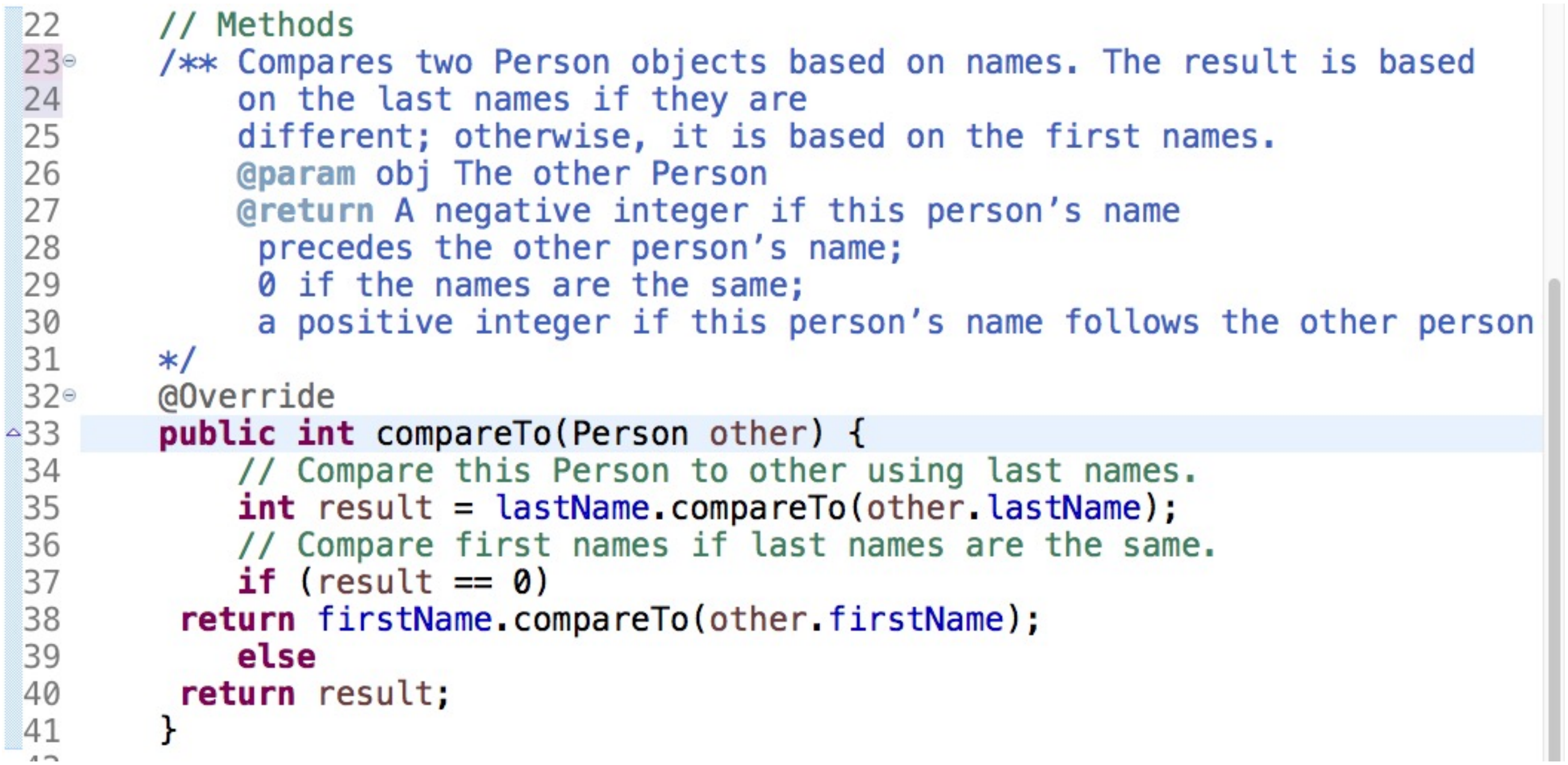
http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html

- As we have seen, implementing **Comparable** will define how you compare a class with *compareTo*
- Implementing **Comparator** lets you create various classes that define how to compare two instances of the generic class
  - various classes that compare based on different rules/fields
  - <u>call a sort method with one comparator object to sort on name and then make another call with a different comparator object to sort on age</u>.



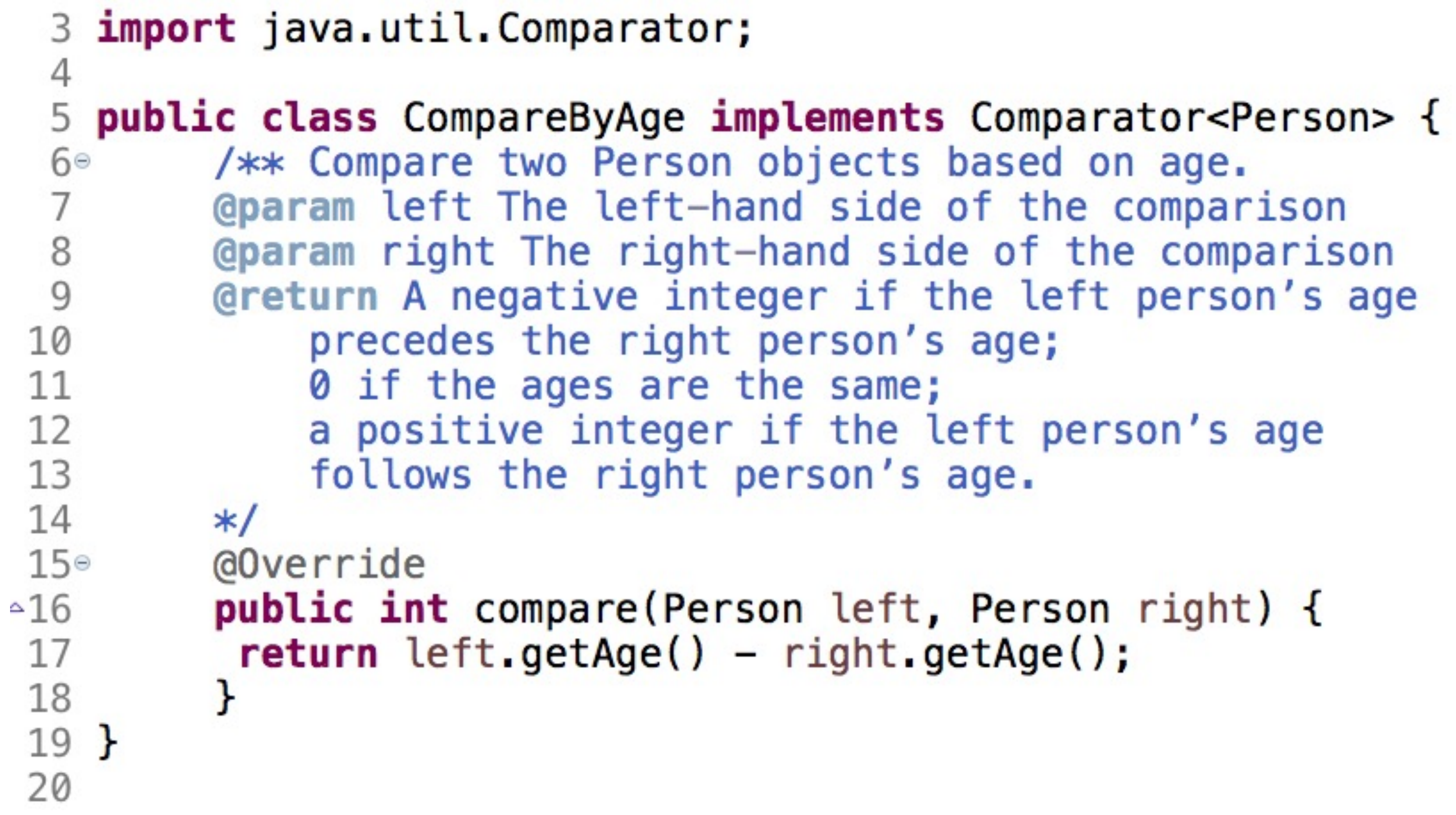
```java
public class Person implements Comparable<Person>  {

```

...

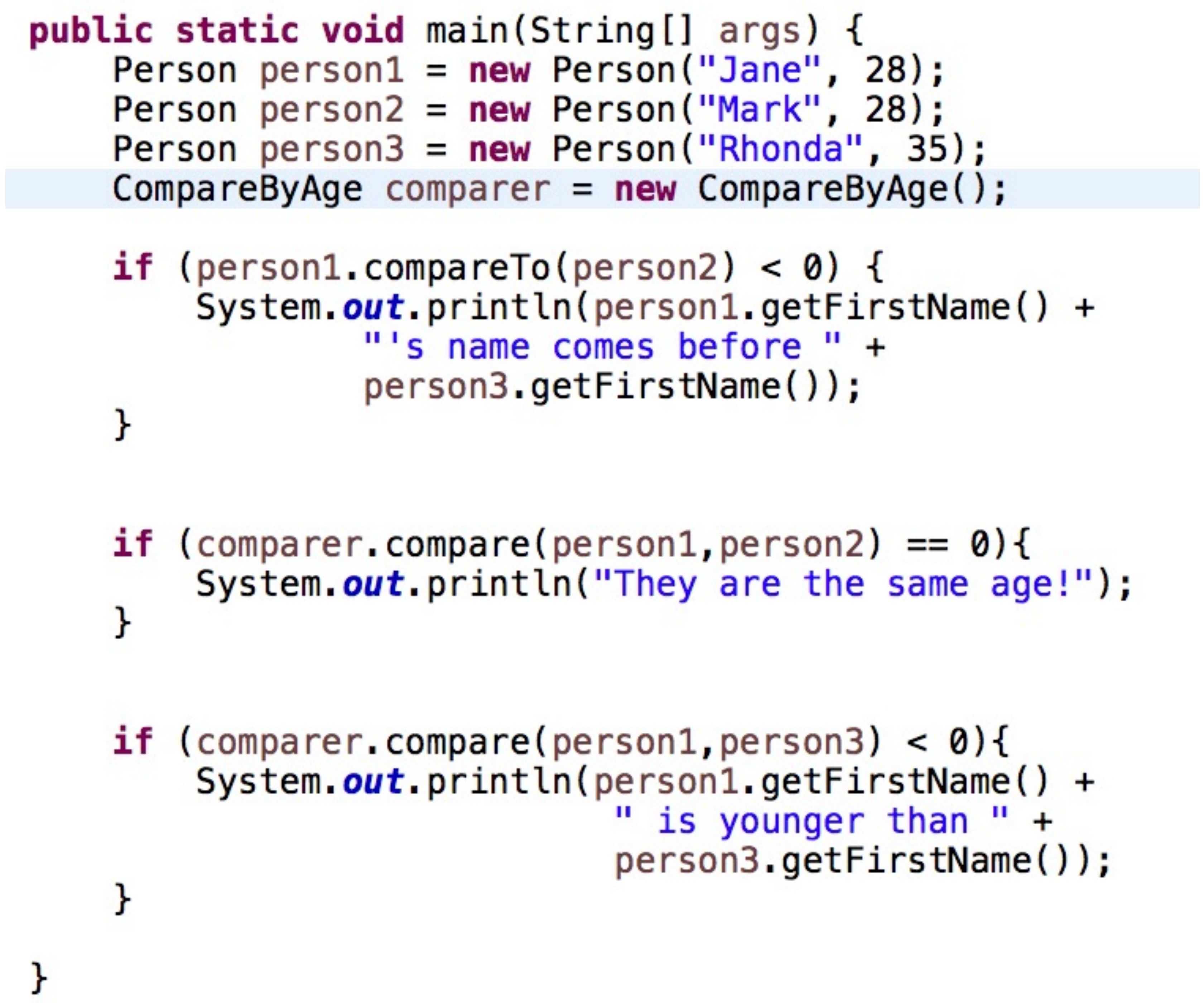


```java
    // Methods
    /** Compares two Person objects based on names. The result is based
        on the last names if they are
        different; otherwise, it is based on the first names.
        @param obj The other Person
        @return A negative integer if this person's name
         precedes the other person's name;
         0 if the names are the same;
         a positive integer if this person's name follows the other person
    */
    @Override
    public int compareTo(Person other) {
        // Compare this Person to other using last names.
        int result = lastName.compareTo(other.lastName);
        // Compare first names if last names are the same.
        if (result == 0)
     return firstName.compareTo(other.firstName);
        else
     return result;
    }
```

...



```java
import java.util.Comparator;

public class CompareByAge implements Comparator<Person> {
     /** Compare two Person objects based on age.
     @param left The left-hand side of the comparison
     @param right The right-hand side of the comparison
     @return A negative integer if the left person’s age
         precedes the right person’s age;
         0 if the ages are the same;
         a positive integer if the left person’s age
         follows the right person’s age.
     */
     @Override
     public int compare(Person left, Person right) {
      return left.getAge() - right.getAge();
     }
}

```



```java
public static void main(String[] args) {
    Person person1 = new Person("Jane", 28);
    Person person2 = new Person("Mark", 28);
    Person person3 = new Person("Rhonda", 35);
    CompareByAge comparer = new CompareByAge();

    if (person1.compareTo(person2) < 0) {
        System.out.println(person1.getFirstName() +
                "'s name comes before " +
                person3.getFirstName());
    }


    if (comparer.compare(person1,person2) == 0){
        System.out.println("They are the same age!");
    }


    if (comparer.compare(person1,person3) < 0){
        System.out.println(person1.getFirstName() +
                               " is younger than " +
                               person3.getFirstName());
    }

}
```

Comparable vs Comparator

- To define one specific way to compare objects as a part of the class, have the class implement **Comparable** and write a compareTo(T pther) member method
- To define multiple ways to compare objects, define distinct classes that implement **Comparator** and define a compare(T left, T right) method. This way comparator objects can be created and sent two objects to compare.

Examples of Comparator

- The String class implements Comparable in such a way that Strings will be ordered in ascending, alphabetical order -- their “natural” order. For cases where you need something else -- sorting by reverse alphabetical order, string length, etc... -- you can create a Comparator class to handle that.

Using Java Standard Sorting Methods

Object passed in determines how elements get sorted

static <T> void sort(T[] a, Comparator<? super T> c)
Sorts the specified array of objects according to the order induced by the specified comparator.

Object passed in determines how elements get sorted

static <T> void sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c)
Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.

(from https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html)