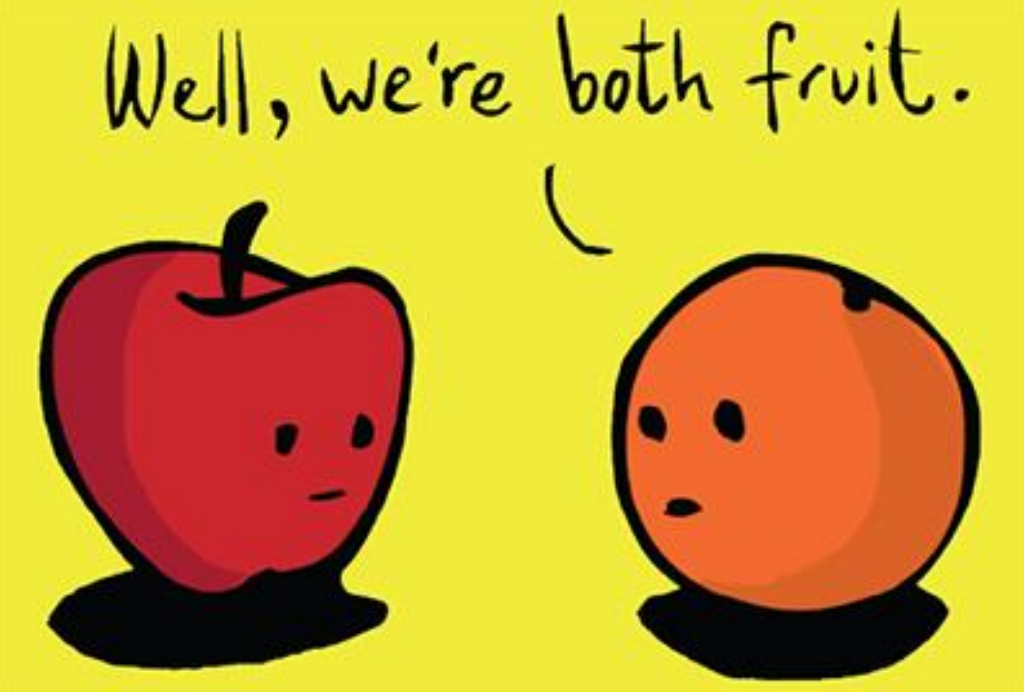
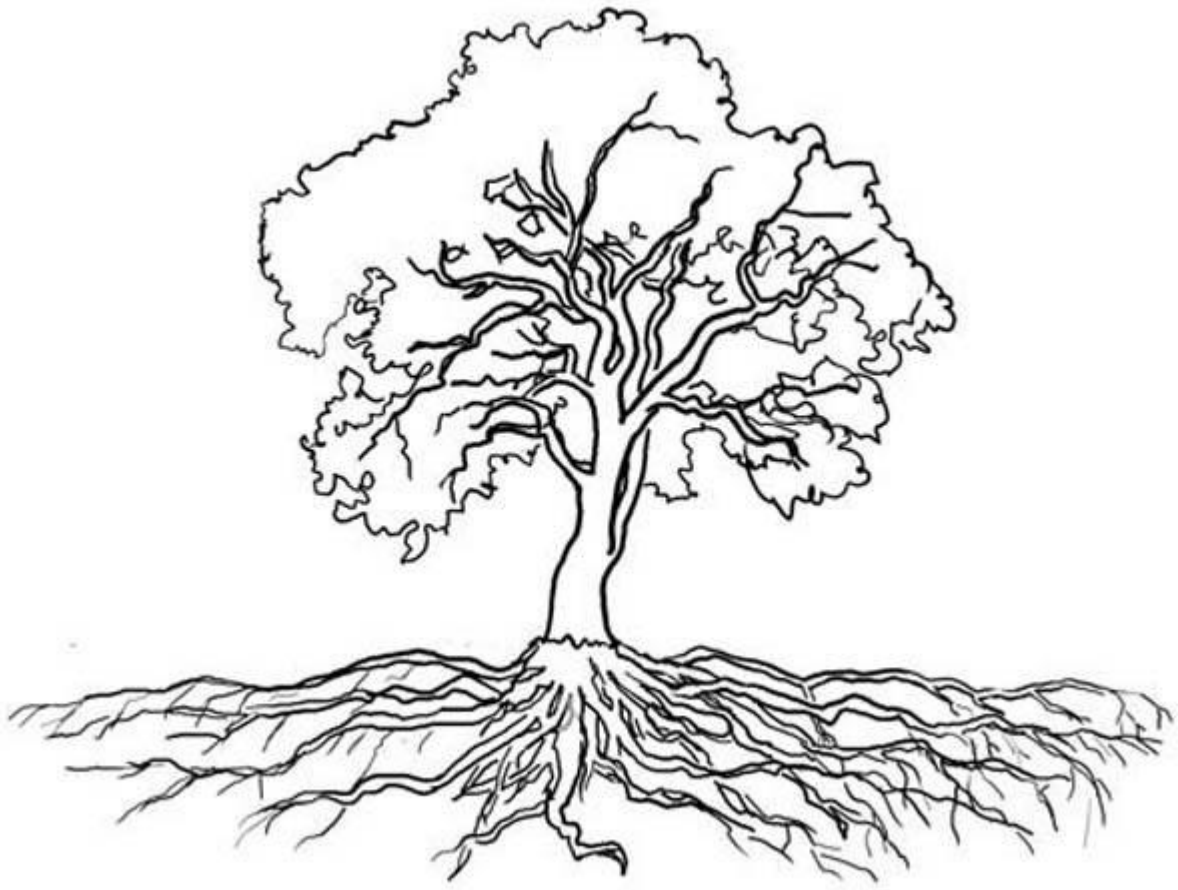
CompareTo

Some more examples





Let’s say, I’m recording information about trees for the Ministry of Forestry.

```
public class Tree {
   private int height;
   private int age;
   private String type;

   public int getHeight()
   public int compareTo(Tree t)
}
```

Eventually, once I get the basics out of the way, I'm going to need to make a report.

I'm going to need to find the max, or - maybe - sort.

Someone might ask, which are the tallest trees?

The Big Tree

BCParks

The largest tree in the park is this giant Douglas-fir. It is over 800 years old, 76 m tall and 9 m round. It was over 300 years old when Christopher Columbus came to North America in 1492.

Douglas-fir is one of Canada's oldest living tree species and can live to be over 1000 years old.

The Big Tree 76 m

The Leaning Tower of Pisa 56 m

PHOTO © STEVE EDLEY

The Leaning Tower of Pisa, one of Europe's most famous landmarks, is dwarfed by the size of the Big Tree.

Tree age can be determined by counting the number of rings on a burl or by a core sample from a living tree.

Tree profile courtesy: Ministry of Forests
(See the *Tree Book* for more information)

Edley Maps & Graphics Inc.
Chemainus B.C. 1997

```
String max = array[0];
for (int i = 0; i < array.length; i++){
  if(array[i].compareTo(max)>0)
     max = array[i];
}
```

Objects, like Strings, can't use > or < or == to compare them. Objects are too complex.

```
Tree max = array[0];
for (int i = 0; i < array.length; i++){
  if(array[i].compareTo(max)>0)
     max = array[i];
}
```

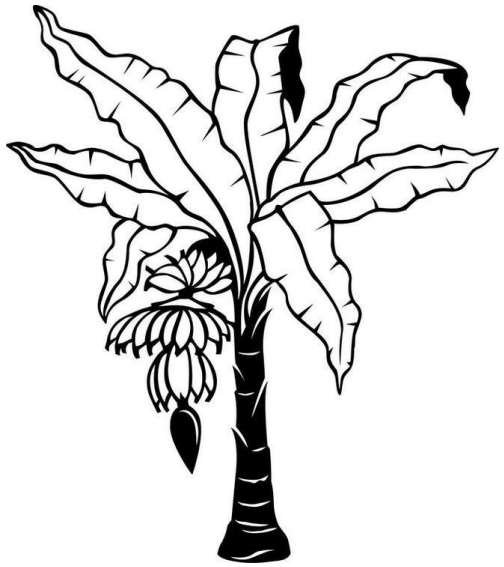
We need to write our own method to compare them.

Tree a = new Tree(12, 34, “apple”);

Tree b = new Tree(10, 12, “banana”);

Tree c = new Tree(11, 6, “willow”);

Tree d = new Tree(6, 3, “pine”);



a: height = 12
age = 34
type = apple

b: height = 10
age = 12
type = banana

c: height = 11
age = 6
type = willow

d: height = 6
age = 3
type = pine

First, I have to make choices

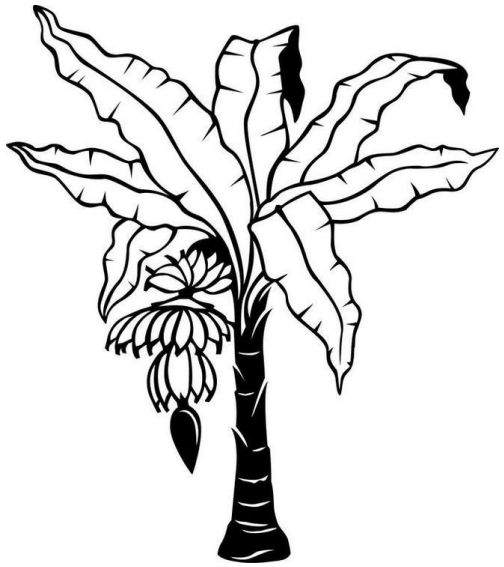
The way I intend to sort them, will impact how I code compareTo.

Tree a = new Tree(12, 34, “apple”);

Tree c = new Tree(11, 6, “willow”);

Tree b = new Tree(10, 12, “banana”);

Tree d = new Tree(6, 3, “pine”);



a: height = 12
age = 34
type = apple

b: height = 10
age = 12
type = banana

c: height = 11
age = 6
type = willow

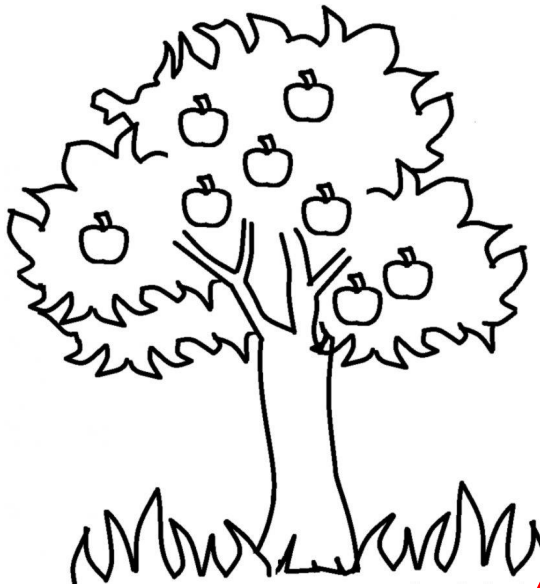
d: height = 6
age = 3
type = pine

Let’s sort by height.

me.compareTo(them)

Can use the instance variables directly

Use the accessors and the parameter name

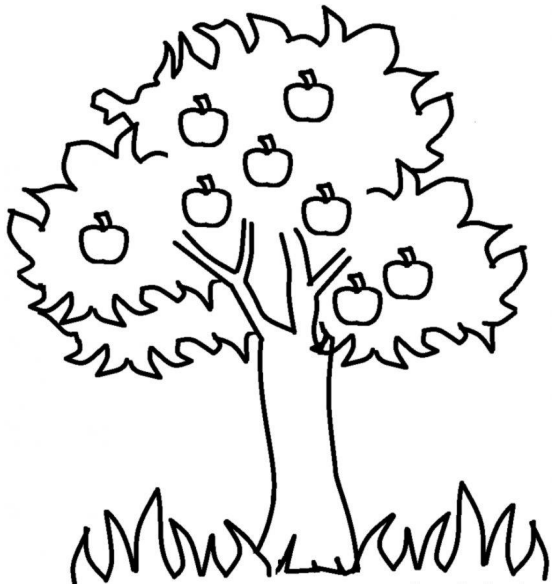


a: height = 12
age = 34
type = apple

```
Tree a = new Tree (12, 34, “apple”);
Tree b = new Tree (10, 12, “banana”);
a.compareTo(b);
```

b: height = 10
age = 12
type = banana

```
public class Tree {
  private int height;
  private int age;
  private String type;
  public int getHeight()
  public int compareTo(Tree t)
}
```



a: height = 12
age = 34
type = apple

```
Tree a = new Tree (12, 34, “apple”);
Tree b = new Tree (10, 12, “banana”);
b.compareTo(a);
```

b: height = 10
age = 12
type = banana

```
public class Tree {
  private int height;
  private int age;
  private String type;
  public int getHeight()
  public int compareTo(Tree t)
}
```

me.equals(them)

Can use the instance variables directly

Use the accessors and the parameter name

me.equals(them)

```
public boolean equals (Tree t){
   if (height == t.getHeight())
      return true;
   else
      return false;
}
```

Equals Side of the Sheet

- The equals method returns a _____.
- It returns _____ if all of the instance variables match and _____ if they don't all match.
- To test if _____, _____, _____ are equal, use ==.

  However, to check if _____ are equal, use .equals.
- The _____ type of the method is the same as the class. This is because we are comparing our instance variable to another of the same type as us.
- Inside the method, think of the _____ variables as belonging to ME and the parameter, which uses _____, as THEM.

Equals Side of the Sheet

- The equals method returns a boolean.
- It returns _____ if all of the instance variables match and _____ if they don't all match.
- To test if _____, _____, _____ are equal, use ==.
  However, to check if _____ are equal, use .equals.
- The _____ type of the method is the same as the class. This is because we are comparing our instance variable to another of the same type as us.
- Inside the method, think of the _____ variables as belonging to ME and the parameter, which uses _____, as THEM.

Equals Side of the Sheet

- The equals method returns a boolean.
- It returns true if all of the instance variables match and false if they don't all match.
- To test if _____, _____, _____ are equal, use ==.

  However, to check if _____ are equal, use .equals.
- The _____ type of the method is the same as the class. This is because we are comparing our instance variable to another of the same type as us.
- Inside the method, think of the _____ variables as belonging to ME and the parameter, which uses _____, as THEM.

Equals Side of the Sheet

- The equals method returns a ___boolean___.
- It returns ___true___ if all of the instance variables match and ___false___ if they don't all match.
- To test if ___int___, ___double___, ___char___ are equal, use ==.
  However, to check if ___String___ are equal, use .equals.
- The _____ type of the method is the same as the class. This is because we are comparing our instance variable to another of the same type as us.
- Inside the method, think of the _____ variables as belonging to ME and the parameter, which uses _____, as THEM.

Equals Side of the Sheet

- The equals method returns a ___boolean___.
- It returns ___true___ if all of the instance variables match and ___false___ if they don't all match.
- To test if ___int___, ___double___, ___char___ are equal, use ==.
  However, to check if ___String___ are equal, use .equals.
- The ___parameter___ type of the method is the same as the class. This is because we are comparing our instance variable to another of the same type as us.
- Inside the method, think of the _____ variables as belonging to ME and the parameter, which uses _____, as THEM.

Equals Side of the Sheet

- The equals method returns a boolean.
- It returns true if all of the instance variables match and false if they don't all match.
- To test if int, double, char are equal, use ==.
  However, to check if String are equal, use .equals.
- The parameter type of the method is the same as the class. This is because we are comparing our instance variable to another of the same type as us.
- Inside the method, think of the instance variables as belonging to ME and the parameter, which uses accessors, as THEM.

me.compareTo(them)

Can use the instance variables directly

Use the accessors and the parameter name

My variables > Their accessors 1

My variables < Their accessors -1

My variables == Their accessors 0

me.compareTo(them)

My variables > Their accessors 1

My variables < Their accessors -1

My variables == Their accessors 0

```
public int compareTo (Tree t){
   if (height > t.getHeight())
      return 1;
   else if (height < t.getHeight())
      return -1;
   else
      return 0;
}
```

The statements can be rearranged.

If I am bigger than them

```
public int compareTo (Tree t){
  if (height > t.getHeight())
    return 1;
  else if (height < t.getHeight())
    return -1;
  else
    return 0;
}
```

If they are smaller than me

```
public int compareTo (Tree t){
  if (t.getHeight() < height)
    return 1;
  else if (t.getHeight() > height)
    return -1;
  else
    return 0;
}
```

```
public int compareTo (Tree t){
  if (height > t.getHeight())
    return 1;
  else if (height < t.getHeight())
    return -1;
  else
    return 0;
}
```

```
public int compareTo (Tree t){
  if (t.getHeight() < height)
    return 1;
  else if (t.getHeight() > height)
    return -1;
  else
    return 0;
}
```

```
public int compareTo (Tree t){
  if (height == t.getHeight())
    return 0;
  else if (height < t.getHeight())
    return -1;
  else
    return 1;
}
```

```
public int compareTo (Tree t){
  if (t.getHeight() > height)
    return -1;
  else if (t.getHeight() == height)
    return 0;
  else
    return 1;
}
```

CompareTo Side of the Sheet

5. Fill in the blanks.

- In a compareTo, the programmer is choosing the _____ order
- In a compareTo, think of the _____ variables as belonging to the _____ object in the method call. Think of that as being "_____".
- In a compareTo, think of the _____ as belonging to the _____ object in the method call. Think of that as being "_____".
- If ME is bigger than THEM, then ME _____ so return _____
- If ME is smaller than THEM, then ME _____ so return _____
- If ME is the same as THEM, then we _____, so return _____

CompareTo Side of the Sheet

5. Fill in the blanks.

- In a compareTo, the programmer is choosing the _____sort_____ order
- In a compareTo, think of the _____ variables as belonging to the _____ object in the method call. Think of that as being "_____".
- In a compareTo, think of the _____ as belonging to the _____ object in the method call. Think of that as being "_____".
- If ME is bigger than THEM, then ME _____ so return _____
- If ME is smaller than THEM, then ME _____ so return _____
- If ME is the same as THEM, then we _____, so return _____

CompareTo Side of the Sheet

5. Fill in the blanks.

- In a compareTo, the programmer is choosing the _____ order
- In a compareTo, think of the _____ variables as belonging to the _____ object in the method call. Think of that as being "_____".
- In a compareTo, think of the _____ as belonging to the _____ object in the method call. Think of that as being "_____".
- If ME is bigger than THEM, then ME _____ so return _____
- If ME is smaller than THEM, then ME _____ so return _____
- If ME is the same as THEM, then we _____, so return _____

sort

instance

first

Me

CompareTo Side of the Sheet

5. Fill in the blanks.

- In a compareTo, the programmer is choosing the sort order
- In a compareTo, think of the instance variables as belonging to the first object in the method call. Think of that as being "Me".
- In a compareTo, think of the accessors as belonging to the second object in the method call. Think of that as being "THEM".
- If ME is bigger than THEM, then ME _____ so return _____
- If ME is smaller than THEM, then ME _____ so return _____
- If ME is the same as THEM, then we _____, so return _____

CompareTo Side of the Sheet

5. Fill in the blanks.

- In a compareTo, the programmer is choosing the ___sort___ order
- In a compareTo, think of the ___instance___ variables as belonging to the ___first___ object in the method call. Think of that as being "___Me___".
- In a compareTo, think of the ___accessors___ as belonging to the ___second___ object in the method call. Think of that as being "___THEM___".
- If ME is bigger than THEM, then ME ___wins___ so return ___1___
- If ME is smaller than THEM, then ME _____ so return _____
- If ME is the same as THEM, then we _____, so return _____

CompareTo Side of the Sheet

5. Fill in the blanks.

- In a compareTo, the programmer is choosing the ___sort___ order
- In a compareTo, think of the ___instance___ variables as belonging to the ___first___ object in the method call. Think of that as being "___Me___".
- In a compareTo, think of the ___accessors___ as belonging to the ___second___ object in the method call. Think of that as being "___THEM___".
- If ME is bigger than THEM, then ME ___wins___ so return ___1___
- If ME is smaller than THEM, then ME ___loses___ so return ___-1___
- If ME is the same as THEM, then we _____, so return _____

CompareTo Side of the Sheet

5. Fill in the blanks.

- In a compareTo, the programmer is choosing the ___sort___ order
- In a compareTo, think of the ___instance___ variables as belonging to the ___first___ object in the method call. Think of that as being "___Me___".
- In a compareTo, think of the ___accessors___ as belonging to the ___second___ object in the method call. Think of that as being "___THEM___".
- If ME is bigger than THEM, then ME ___wins___ so return ___1___
- If ME is smaller than THEM, then ME ___loses___ so return ___-1___
- If ME is the same as THEM, then we ___tie___, so return ___0___

If you think I'm explaining it badly, try the official documentation instead:

Java Comparable interface

Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.

compareTo(Object obj) method

public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns:

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

You won't understand any better, but you will appreciate my explanation more.

3.Consider the following String operations. Fill in the blanks and use it to circle the result.

```
String ap = "apple";
String ban = "banana";
String cant = "cantaloupe";
```

```
First >  Second   => 1
First == Second   => 0
First <  Second   => -1
```

Expression	First's value (look above, fill in)	Relation (circle)	Second's value (look above, fill in)	CompareTo Result (circle)
`ap.compareTo("peach");`		> = <		1, 0 -1
`ap.compareTo("aaaa");`		> = <		1, 0 -1
`ban.compareTo("plum");`		> = <		1, 0 -1
`ban.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(ban);`		> = <		1, 0 -1
`ban.compareTo(cant);`		> = <		1, 0 -1
`cant.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(cant);`		> = <		1, 0 -1

3.Consider the following String operations. Fill in the blanks and use it to circle the result.

```
String ap = "apple";
String ban = "banana";
String cant = "cantaloupe";
```

```
First >  Second   => 1
First == Second   => 0
First <  Second   => -1
```

Expression	First's value (look above, fill in)	Relation (circle)	Second's value (look above, fill in)	CompareTo Result (circle)
`ap.compareTo("peach");`	apple	> = <	peach	1, 0 -1
`ap.compareTo("aaaa");`		> = <		1, 0 -1
`ban.compareTo("plum");`		> = <		1, 0 -1
`ban.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(ban);`		> = <		1, 0 -1
`ban.compareTo(cant);`		> = <		1, 0 -1
`cant.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(cant);`		> = <		1, 0 -1

3.Consider the following String operations. Fill in the blanks and use it to circle the result.

```
String ap = "apple";
String ban = "banana";
String cant = "cantaloupe";
```

```
First >  Second   => 1
First == Second   => 0
First <  Second   => -1
```

Expression	First's value (look above, fill in)	Relation (circle)	Second's value (look above, fill in)	CompareTo Result (circle)
`ap.compareTo("peach");`	apple	> = <	peach	1, 0 -1
`ap.compareTo("aaaa");`		> = <		1, 0 -1
`ban.compareTo("plum");`		> = <		1, 0 -1
`ban.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(ban);`		> = <		1, 0 -1
`ban.compareTo(cant);`		> = <		1, 0 -1
`cant.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(cant);`		> = <		1, 0 -1

3.Consider the following String operations. Fill in the blanks and use it to circle the result.

```
String ap = "apple";
String ban = "banana";
String cant = "cantaloupe";
```

```
First >  Second   => 1
First == Second   => 0
First <  Second   => -1
```

Expression	First's value (look above, fill in)	Relation (circle)	Second's value (look above, fill in)	CompareTo Result (circle)
`ap.compareTo("peach");`	apple	> = <	peach	1, 0 -1
`ap.compareTo("aaaa");`		> = <		1, 0 -1
`ban.compareTo("plum");`		> = <		1, 0 -1
`ban.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(ban);`		> = <		1, 0 -1
`ban.compareTo(cant);`		> = <		1, 0 -1
`cant.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(cant);`		> = <		1, 0 -1