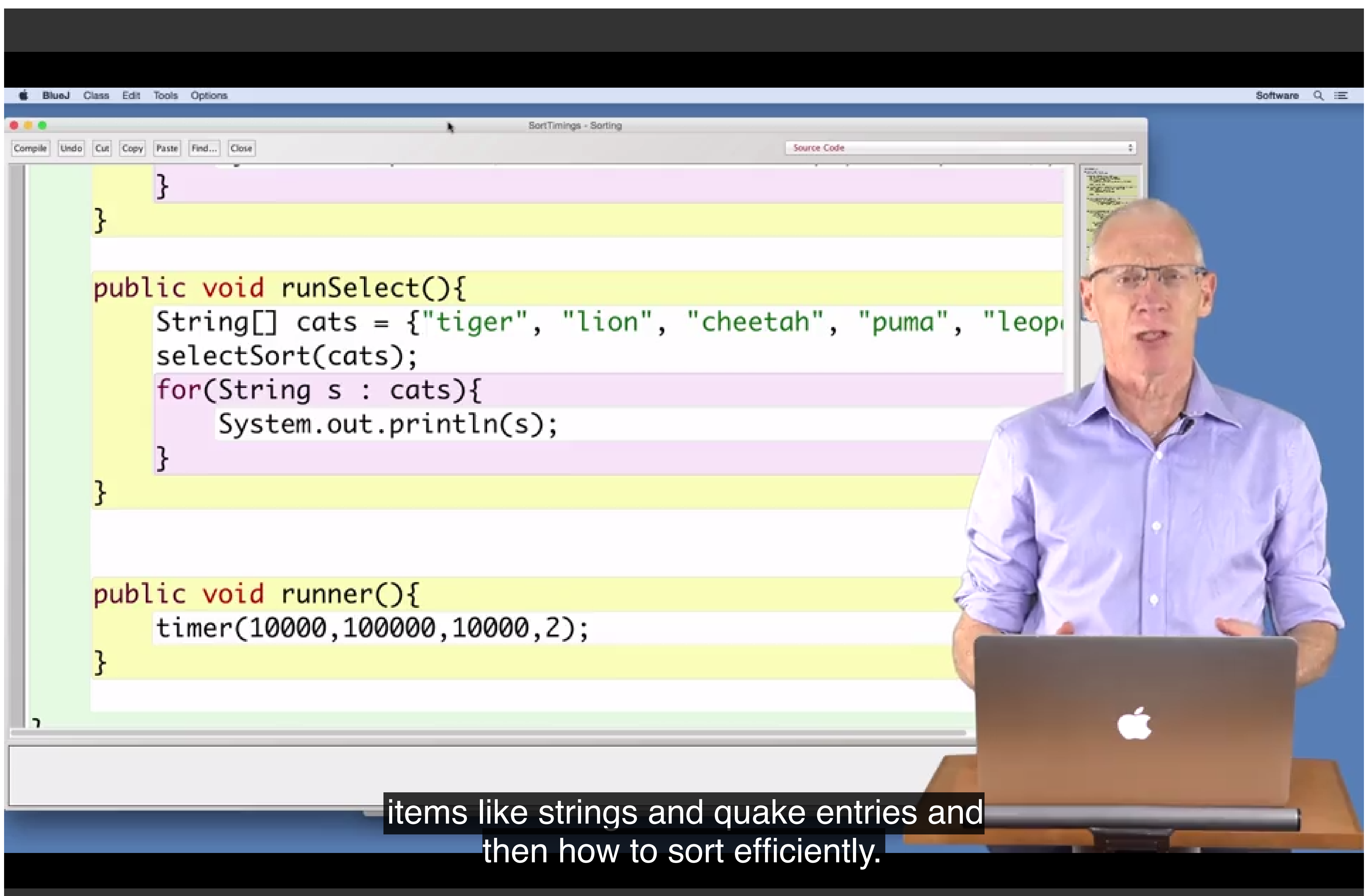
Implementing Selection Sort

Sorting at Scale



```
    }
}

public void runSelect(){
    String[] cats = {"tiger", "lion", "cheetah", "puma", "leop
    selectSort(cats);
    for(String s : cats){
        System.out.println(s);
    }
}

public void runner(){
    timer(10000,100000,10000,2);
}
```

items like strings and quake entries and then how to sort efficiently.

Introduction

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript English

0:03
Hi. In this coding demonstration we're going to see both how to sort items like strings and quake entries and then how to sort efficiently. You've already seen some information about using SelectionSort and preview if you're using the built in system collections.sort so we're going to take a look at those more closely. What you've already seen with Quake entries is how to sort by magnitude using the SelectionSort algorithm. So what I'd like to do first is see how can I sort strings using the same algorithm. So I've got my code here that I'm going to try to run SelectionSort on. I've created a very small array just for demonstration purposes that has the name of some cats. Tiger, lion, cheetah Puma and leopard. I'm going to sort them using SelectionSort. I just copied that code from the quake entries where we sorted by magnitude. And I'm going to sort the strings by just replacing the array of quake entries with an array of strings. So I've got a very simple code here. I'm ready to compile it and test it. Uh-oh. Here's my SelectionSort algorithm which is just like the one that we had in one of the previous lessons and it says bad operand types for binary operator less than. String and string. And it's highlighted this entry where I'm comparing the current element I'm iterating over with the minimal element. And I did that with the quake entries to compare them by magnitude. Here it's saying I can't use less than to compare strings. And that's what happens with Java in anything other than the primitive types. Ints, doubles, char. I can compare those using the less than operator as I have here. But for object or quake entry or other more complicated types like string, I'm not allowed to use the less than operator. Java won't let me. So I'm going to show you what's gonna be the topic of this next group of lessons. I'm going to use the compare to method. And I replace the less than sign with compare to and ask if it's less than zero. That may seem a little mysterious right now but we're gonna explain that in the next sequence of lessons that you see. So list[j].compareTo(list[mindex]) < 0. I'm gonna compile this. And now it compiled. And no problems running it. I'm going to come in and make an object on my object work bench And now there are many methods in here. I wanna do runSelect. That's the one I just had that sorted cats.

2:39
And I can see cheetah, leopard, lion, puma, and tiger which is exactly what I wanted. The strings were printed in alphabetical order. So that CompareTo method Did what the less than method did for ints. But compared to works with strings and quake entries as we'll see in the next group of lessons. That shows me that I need this other method, compare to in order to compare strings or quake entries and any other things besides int and doubles. What I'd like to do now is look at the efficiency of the sort. Now that I've seen I can compare strings with compare to. I'm going to look at efficiency of methods. What we've done here in the sort timings class that you heard a little bit about of before, is create some timings. I'm going to sort 10,000 random strings. And then 20,000 random strings. All the way up to 100,000. I've created a method to do that and I'm gonna run that method now.

3:37
I'm going to select my object and I'm going to call runner which calls my sort methods for several different array lists. For 10,000 strings it took 0.35 seconds to sort them with SelectionSort. But the built-in collection that sort as we'll see in a minute took a hundredth of a second.

3:59
For 30,000 strings I'm almost at 3 seconds. A hundredth of a second for selections.sort. 40,000 more. Still, hundredths of a second.

4:12
I could keep this running but it's going to take a while. So I'm going to stop running by shutting down my Java Virtual Machine. And I'm going to show you the output of this program that I ran before and let it run to completion. Here's a screenshot of me running that sort timings.java class, and for 10 thousand you can see SelectionSort was fine. It took 0.35 seconds. By the time I got up to 100 thousand strings it took nearly a minute for SelectionsSort to run. And four hundredths of a second for collections.sort to run. Let me make sure I've pointed that out. 100 thousand. I could take a minute or .04 seconds. And as you saw in one of the previous lessons that timing gets worse and worse the more elements you have.

5:01
To see what I did. I've taken the call to selectSort and run it. And I printed out the time it took using System.nanoTime. A convenient method that Java provides for timing code segments. And then I called Collections.sort. You'll be able to study this code on your own if you look at this sorttimings.java class that we provide. But Collections.sort works with array list of strings precisely because I have the compareTo method. Here's my select sort again. Let me remind you. CompareTo. That's the method we use when sorting. We'll see more of that in this lesson. Happy sorting. Efficiently.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?