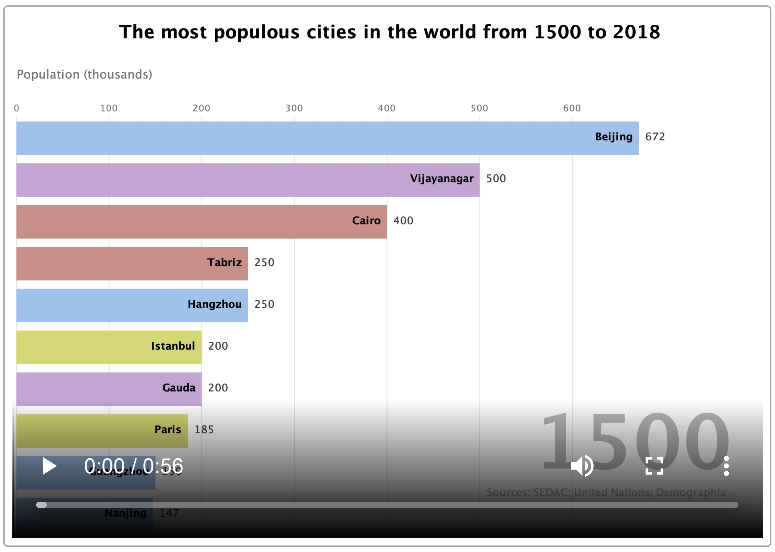
Write a program to produce animated bar charts like the one below. As this is the last programming assignment in the course, we also include two open-ended questions.



1. **Bar data type.** A *bar* aggregates related information (*name*, *value*, and *category*) for use in a bar chart. For example, the first bar drawn in the bar chart represents *name = Beijing*, *value = 672*, and *category = East Asia*. In addition to methods for accessing the individual fields, you will need to *sort* bars in order of value. Implement the following API:

```
public class Bar implements Comparable<Bar> {

    // Creates a new bar.
    public Bar(String name, int value, String category)

    // Returns the name of this bar.
    public String getName()

    // Returns the value of this bar.
    public int getValue()

    // Returns the category of this bar.
    public String getCategory()

    // Compare two bars by value.
    public int compareTo(Bar that)

    // Sample client (see below).
    public static void main(String[] args)

}
```

Sorting arrays of objects. To sort an array of objects, use Java's Arrays.sort() method. For example, the following code fragment creates an array of 10 bars and sorts them in ascending order of value.

```
// create an array of 10 bars
Bar[] bars = new Bar[10];
bars[0] = new Bar("Beijing",     22674, "East Asia");
bars[1] = new Bar("Cairo",       19850, "Middle East");
bars[2] = new Bar("Delhi",       27890, "South Asia");
bars[3] = new Bar("Dhaka",       19633, "South Asia");
bars[4] = new Bar("Mexico City", 21520, "Latin America");
bars[5] = new Bar("Mumbai",      22120, "South Asia");
bars[6] = new Bar("Osaka",       20409, "East Asia");
bars[7] = new Bar("São Paulo",   21698, "Latin America");
bars[8] = new Bar("Shanghai",    25779, "East Asia");
bars[9] = new Bar("Tokyo",       38194, "East Asia");

// sort in ascending order by weight
Arrays.sort(bars);
```

Comparable interface. In order to use a data type with `Arrays.sort()`, that data type must be *comparable*. This is Java's mechanism for specifying a *total order* among objects of a given type. To make a data type comparable, you must implement the Comparable interface, which involves doing two things:

   - Add `implements Comparable<Bar>` to the class definition to indicate that `Bar` objects can be compared (by value).
   - Include a method `compareTo()` to specify how to compare two `Bar` objects. In this case, return a { negative integer, zero, positive integer } if value of the invoking object is { less than, equal to, greater than } the value of the argument object.

   Corner cases. Handle invalid argument in the following manner:

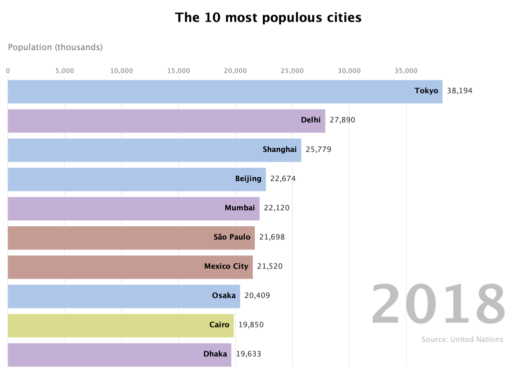
   - Throw an `IllegalArgumentException` in the constructor if `name` is `null`, `value` is negative, or `category` is `null`.
   - Throw a `NullPointerException` if the argument to `compareTo()` is `null`.

2. **Bar chart data type.** We provide a simple data type BarChart.java that supports drawing static bar charts to standard draw. The purpose of this step is solely to familiarize yourself with its API:

```
public class BarChart {

    // Creates a bar chart with the given title, x-axis label, and data source.
    public BarChart(String title, String xAxisLabel, String dataSource)

    // Sets the caption of this bar chart.
    public void setCaption(String caption)

    // Adds a bar (name, value, category) to this bar chart.
    public void add(String name, int value, String category)

    // Remove all of the bars from this bar chart.
    public void reset()

    // Draws this bar chart to standard draw.
    public void draw()
}
```

   The following code fragment illustrates the API by drawing a bar chart with 10 bars. Note that the bars are drawn in the same order in which they are added to the bar chart (from top to bottom).

```
// create the bar chart
String title = "The 10 most populous cities";
String xAxis = "Population (thousands)";
String source = "Source: United Nations";
BarChart chart = new BarChart(title, xAxis, source);
chart.setCaption("2018");

// add the bars to the bar chart
chart.add("Tokyo",       38194, "East Asia");
chart.add("Delhi",       27890, "South Asia");
chart.add("Shanghai",    25779, "East Asia");
chart.add("Beijing",     22674, "East Asia");
chart.add("Mumbai",      22120, "South India");
chart.add("São Paulo",   21698, "Latin America");
chart.add("Mexico City", 21520, "Latin America");
chart.add("Osaka",       20409, "East Asia");
chart.add("Cairo",       19850, "Middle East");
chart.add("Dhaka",       19633, "South Asia");

// draw the bar chart
StdDraw.setCanvasSize(1000, 700);
StdDraw.enableDoubleBuffering();
chart.draw();
StdDraw.show();
```



3. **Bar chart racer.** Write a program to produce animated bar charts, using BarChart.java to draw static bar charts.

   As a canonical example, consider an animated bar chart of the 10 most populous cities in the world, from 1500 to 2018. To produce the visualization, you will successively draw 519 individual bar charts (one per year of data), with a short pause between each drawing. Each bar chart contains the 10 most populous cities in that year, arranged in descending order of population.

   File format. A bar-chart-racer data file is organized as a sequence of lines. The first three lines comprise the *header*:

   - The *title*.
   - The *x-axis label*.
   - The *source* of the data.

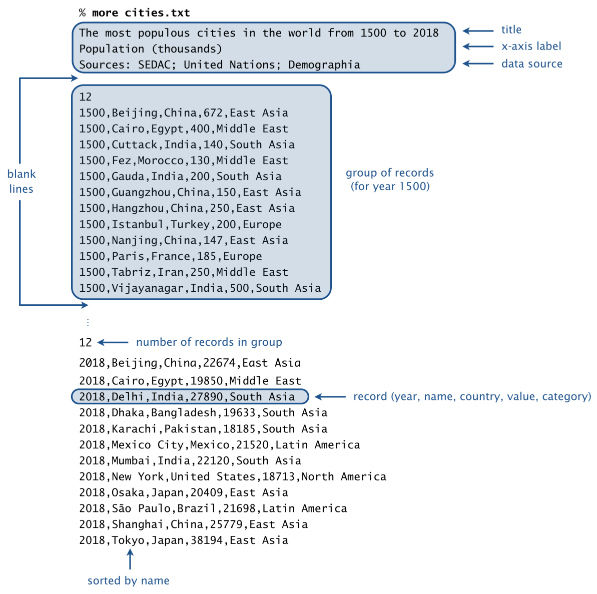
   Following the header is a blank line, followed by the raw data. Each line (or *record*) consists of 5 fields, separated by commas:

   - The *year* or *date* (e.g, 2018).
   - The *name* (e.g., Mumbai).

- The associated *country* (e.g., India).
- The *value* (e.g, 22120).
- The *category* (e.g., South Asia).

The value field is an integer; the other fields can be arbitrary strings (except that they can’t contain commas or newlines).

Records corresponding to the same year (or time period) are grouped together. A *group* of records consists of an integer n, followed by n records. Within a group, the records are sorted by name. A blank line separates each group.



```
% more cities.txt
The most populous cities in the world from 1500 to 2018
Population (thousands)
Sources: SEDAC; United Nations; Demographia

12
1500,Beijing,China,672,East Asia
1500,Cairo,Egypt,400,Middle East
1500,Cuttack,India,140,South Asia
1500,Fez,Morocco,130,Middle East
1500,Gauda,India,200,South Asia
1500,Guangzhou,China,150,East Asia
1500,Hangzhou,China,250,East Asia
1500,Istanbul,Turkey,200,Europe
1500,Nanjing,China,147,East Asia
1500,Paris,France,185,Europe
1500,Tabriz,Iran,250,Middle East
1500,Vijayanagar,India,500,South Asia

⋮
12
2018,Beijing,China,22674,East Asia
2018,Cairo,Egypt,19850,Middle East
2018,Delhi,India,27890,South Asia
2018,Dhaka,Bangladesh,19633,South Asia
2018,Karachi,Pakistan,18185,South Asia
2018,Mexico City,Mexico,21520,Latin America
2018,Mumbai,India,22120,South Asia
2018,New York,United States,18713,North America
2018,Osaka,Japan,20409,East Asia
2018,São Paulo,Brazil,21698,Latin America
2018,Shanghai,China,25779,East Asia
2018,Tokyo,Japan,38194,East Asia
```

title
x-axis label
data source
blank lines
group of records (for year 1500)
number of records in group
record (year, name, country, value, category)
sorted by name

Data files. We supply a number of fascinating data files in the specified format, curated from various sources.

input file	description	period	data source
cities.txt	*most populous cities in the world*	1500–2018	John Burn-Murdoch
countries.txt	*most populous countries in the world*	1950–2100	United Nations
cities-usa.txt	*most populous cities in the U.S.*	1790-2018	U.S. Census Bureau
brands.txt	*most valuable brands in the world*	2000–2018	Interbrand
movies.txt	*highest-grossing movies in the U.S.*	1982–2019	Box Office Mojo
baby-names.txt	*most popular baby names in the U.S.*	1880–2018	U.S. Social Security
football.txt	*the best football clubs in Europe*	1960–2019	clubelo.com
endgame.txt	*characters in Endgame by screen time*	Minute 1–170	Prashant
infinity-war.txt	*characters in Infinity War by screen time*	Minute 1–132	Prashant
trademarks.txt	*trademarks granted by country*	1980–2018	WIPO
patents.txt	*patents granted by country*	1980–2018	WIPO

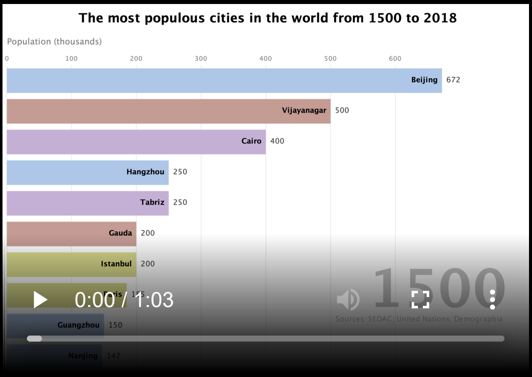
Soundtrack (optional). If you would like a musical accompaniment to your animated bar chart, add the following line at the beginning of your program:

```
StdAudio.loop("soundtrackA.wav");
```

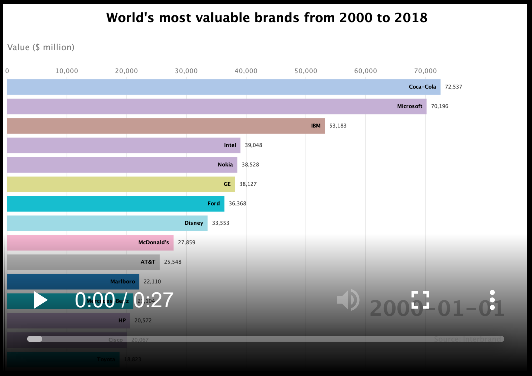
Feel free to substitute a different .wav file.

Command-line arguments. The program takes two command-line arguments: the name of a bar-chart-racer file and an integer k that specifies how many bars to display in each bar chart. Here are some sample executions:

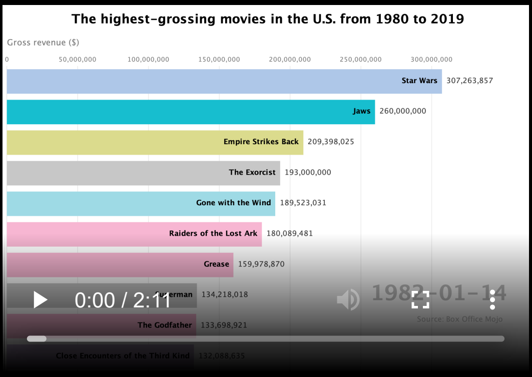
```
~/Desktop/barchart> java-introcs BarChartRacer cities.txt 10
```



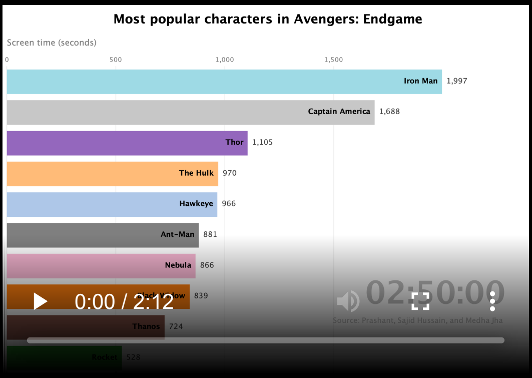
```
~/Desktop/barchart> java-introcs BarChartRacer brands.txt 15
```



```
~/Desktop/barchart> java-introcs BarChartRacer movies.txt
```



```
~/Desktop/barchart> java-introcs BarChartRacer endgame.txt 10
```



4. **Curate a bar chart racer data file (ungraded).** Create a data file in the given format from a subject of interest to you. The data can be international or specific to your country or region. Curating a data file typically involves several steps:

   - Identifying an appropriate data source, typically one that is available freely on the web.
   - Downloading the raw data. Depending on the format of the data, this may involve *web scraping*, where you write a program to download the raw HTML containing the data and then parse that HTML to extract the relevant information.
   - Reorganizing the data into the specified format.

    - Cleaning the data by detecting and correcting any missing, inaccurate, or inconsistent parts.

    Create a plan to update and maintain your data file when new data becomes available from the data source.

5. **Bar chart racer II (ungraded).** Learn a new programming language (such as Python, C++, C#, C, Javascript, Swift, Go, Scala, Kotlin, Ruby, Rust, R, or Matlab) and create an animated bar chart in that language.

Submission. Submit a `.zip` file containing `Bar.java` and `BarChartRacer.java`.

This assignment was developed by Kevin Wayne, inspired by tweets from Matt Navarra and John Burn-Murdoch.