1 /\*\*  
 2 \* This is the TVShowTester class that tests different sorting methods to sort then print  
 3 \* information stored in TVShow objects. It contains the following methods:  
 4 \*  
 5 \* removeReturn(String): accepts a string, removes the return (\r) at the end of the   
 6 \* string, and returns the string  
 7 \*   
 8 \* printShows(): prints all of the information for each TVShow object in the given array  
 9 \*   
 10 \* sortInsertion(TVShow array, int order, int category): takes an array of TVShow, sorts   
 11 \* it via insertion sort in 1-ascending or 2-descending order, by 1-name or 2-season   
 12 \* or 3-genre  
 13 \*   
 14 \* sortSelection(TVShow array, int order, int category): takes an array of TVShow, sorts   
 15 \* it via selection sort in 1-ascending or 2-descending order, by 1- year or 2- genre or   
 16 \* 3-male (lead last name)  
 17 \*   
 18 \* sortMerge(TVShow array, int min, int max, int order, int category): takes an array of   
 19 \* TVShow, sorts it via merge sort taking the min and max indexes from the given array,   
 20 \* in 1-ascending or 2-descending order, by 1-network or 2-episodes or 3-female (lead   
 21 \* last name)  
 22 \*   
 23 \* mergeMerge(TVShow array, int min, int max, int order, int category): takes the divided   
 24 \* arrays from sortMerge() and sorts them into a returned array  
 25 \*   
 26 \* searchSequential(TVShow array, int x, int category): takes a TVShow array and searches   
 27 \* it for TVShows that having matching information x, in 1-seasons or 2-episodes  
 28 \*   
 29 \* searchBinary(TVShow array, String seeking, int min, int max, int category): takes TVShow   
 30 \* array and finds if the information it is seeking is at the midpoint, then recursively   
 31 \* sends back each half of the array to check if there are more matching cases in the first   
 32 \* or second halves  
 33 \*   
 34 \* lastName(String name): accepts a string with first and last name, finds the letter after the   
 35 \* space (between the first and last name), and returns the substring of the last name  
 36 \*  
 37 \* @author Stephanie Gremillion  
 38 \* @version 17.0.2  
 39 \* @since 2022/07/20  
 40 \*/  
 41   
 42 import java.util.Scanner; // for use of Scanner  
 43 import java.io.IOException; // to throw exception with opening files  
 44 import java.io.File; // to be able to open a file  
 45 import java.util.ArrayList; // to use arrayLists and methods  
 46 import java.util.Arrays; // to use arrays and methods  
 47   
 48 public class TVShowTester {  
 49 public static void main(String[] args) throws IOException {  
 50 // variables  
 51 Scanner in = new Scanner(new File("tv\_shows.csv"));  
 52 in.useDelimiter(",|\n"); // getting rid of commas and new lines  
 53 TVShow[] shows = new TVShow[20];  
 54 ArrayList<Integer> temp;  
 55   
 56 // skipping first line full of labels  
 57 in.nextLine();  
 58   
 59 // reading file to create array objects  
 60 for(int i = 0; i < 20; i++) {  
 61 shows[i] = new TVShow(in.next(), Integer.parseInt(in.next()), Integer.parseInt(in.next()), Integer.parseInt(in.next()), in.next(), in.next(), in.next(), removeReturn(in));  
 62 }  
 63   
 64 // printing pre-sort/manipulation table  
 65 System.out.println("Initial List");  
 66 System.out.println("-------------");  
 67 System.out.printf("%-20s", "Name");  
 68 System.out.printf("%-20s", "Year Premiered");  
 69 System.out.printf("%-20s", "Seasons");  
 70 System.out.printf("%-20s", "Episodes");  
 71 System.out.printf("%-20s", "Network");  
 72 System.out.printf("%-20s", "Genre");  
 73 System.out.printf("%-20s", "Male Lead");  
 74 System.out.printf("%-20s", "Female Lead");  
 75 System.out.println();  
 76 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nysewgmf"); }  
 77 System.out.println();  
 78   
 79 // printing Insertion Sort 1  
 80 System.out.println("Insertion Sort 1: Name Ascending");  
 81 System.out.println("--------------------------------");  
 82 System.out.printf("%-20s", "Name");  
 83 System.out.printf("%-20s", "Year");  
 84 System.out.printf("%-20s", "Male Lead");  
 85 System.out.printf("%-20s\n", "Female Lead");  
 86 sortInsertion(shows, 1, 1);  
 87 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nymf"); }  
 88 //for(int i = 0; i < shows.length; i++) { shows[i].insSortOne(); }  
 89 System.out.println();  
 90   
 91 // printing Insertion Sort 2  
 92 System.out.println("Insertion Sort 2: Seasons Descending");  
 93 System.out.println("------------------------------------");  
 94 System.out.printf("%-20s", "Name");  
 95 System.out.printf("%-20s", "Year");  
 96 System.out.printf("%-20s\n", "Seasons");  
 97 sortInsertion(shows, 2, 2);  
 98 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nys"); }  
 99 System.out.println();  
100   
101 // printing Selection Sort 1  
102 System.out.println("Selection Sort 1: Year Ascending");  
103 System.out.println("--------------------------------");  
104 System.out.printf("%-20s", "Name");  
105 System.out.printf("%-20s", "Year");  
106 System.out.printf("%-20s\n", "Seasons");  
107 sortSelection(shows, 1, 1);  
108 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nys"); }  
109 System.out.println();  
110   
111 // printing Selection Sort 2  
112 System.out.println("Selection Sort 2: Genre Descending");  
113 System.out.println("----------------------------------");  
114 System.out.printf("%-20s", "Name");  
115 System.out.printf("%-20s", "Episodes");  
116 System.out.printf("%-20s\n", "Genre");  
117 sortSelection(shows, 2, 2);  
118 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("neg"); }  
119 System.out.println();  
120   
121 // printing Merge Sort 1  
122 System.out.println("Merge Sort 1: Network Descending");  
123 System.out.println("--------------------------------");  
124 System.out.printf("%-20s", "Name");  
125 System.out.printf("%-20s", "Year");  
126 System.out.printf("%-20s", "Seasons");  
127 System.out.printf("%-20s\n", "Network");  
128 sortMerge(shows, 0, shows.length - 1, 2, 1); //1 net, 2 epi  
129 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nysw"); }  
130 System.out.println();  
131   
132 // printing Merge Sort 2  
133 System.out.println("Merge Sort 2: Episodes Ascending");  
134 System.out.println("--------------------------------");  
135 System.out.printf("%-20s", "Name");  
136 System.out.printf("%-20s", "Year");  
137 System.out.printf("%-20s", "Episodes");  
138 System.out.printf("%-20s\n", "Network");  
139 sortMerge(shows, 0, shows.length - 1, 1, 2);  
140 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nyew"); }  
141 System.out.println();  
142   
143 // printing Sequential Search 1  
144 System.out.println("Sequential Search 1: 2 Seasons");  
145 System.out.println("------------------------------");  
146 System.out.printf("%-20s", "Name");  
147 System.out.printf("%-20s", "Year");  
148 System.out.printf("%-20s", "Seasons");  
149 System.out.printf("%-20s\n", "Network");  
150 temp = searchSequential(shows, 2, 1);  
151 for(int i = 0; i < temp.size(); i++) { shows[temp.get(i)].printInfo("nysw"); }  
152 System.out.println();  
153   
154 // printing Sequential Search 2  
155 System.out.println("Sequential Search 2: 100+ Episodes");  
156 System.out.println("-----------------------------------");  
157 System.out.printf("%-20s", "Name");  
158 System.out.printf("%-20s", "Seasons");  
159 System.out.printf("%-20s\n", "Episodes");  
160 temp = searchSequential(shows, 100, 2);  
161 for(int i = 0; i < temp.size(); i++) { shows[temp.get(i)].printInfo("nse"); }  
162 System.out.println();  
163   
164 // printing Binary Search 1  
165 System.out.println("Binary Search 1: Shows Aired on CBS");  
166 System.out.println("-----------------------------------------------");  
167 System.out.println("Sorts by Network via Merge Sort Descending");  
168 System.out.println("-----------------------------------------------");  
169 temp = searchBinary(shows, "CBS", 0, shows.length - 1, 1);  
170 if(temp.isEmpty()) { System.out.println("No shows found from CBS Network."); }  
171 else {  
172 System.out.println(temp.size() + " shows counted from CBS Network:");  
173 for(int i = 0; i < temp.size(); i++) { System.out.println(shows[temp.get(i)].getName()); }  
174 }  
175 System.out.println();  
176   
177 // printing Binary Search 2  
178 System.out.println("Binary Search 2: Shows in the Fantasy Genre");  
179 System.out.println("-------------------------------------------------");  
180 System.out.println("Sorts by Genre via Selection Sort Descending");  
181 System.out.println("-------------------------------------------------");  
182 temp = searchBinary(shows, "Fantasy", 0, shows.length - 1, 2);  
183 if(temp.isEmpty()) { System.out.println("No shows in the Fantasy Genre."); }  
184 else{  
185 System.out.println(temp.size() + " shows counted in the Fantasy Genre:");  
186 for(int i = 0; i < temp.size(); i++) { System.out.println(shows[temp.get(i)].getName()); }  
187 }  
188 System.out.println();  
189   
190 // printing Binary Search 3  
191 System.out.println("Binary Search 3: Shows in the Drama Genre");  
192 System.out.println("-------------------------------------------------");  
193 System.out.println("Sorts by Genre via Insertion Sort Descending");  
194 System.out.println("-------------------------------------------------");  
195 temp = searchBinary(shows, "Drama", 0, shows.length - 1, 2);  
196 if(temp.isEmpty()) { System.out.println("No shows in the Drama Genre."); }  
197 else{  
198 System.out.println(temp.size() + " shows counted in the Drama Genre:");  
199 for(int i = 0; i < temp.size(); i++) { System.out.println(shows[temp.get(i)].getName()); }  
200 }  
201 System.out.println();  
202   
203 // printing Bonus 1  
204 System.out.println("Bonus 1: Male Lead Last Name Ascending via Selection Sort");  
205 System.out.println("---------------------------------------------------------");  
206 sortSelection(shows, 1, 3);  
207 System.out.printf("%-20s", "Name");  
208 System.out.printf("%-20s", "Year");  
209 System.out.printf("%-20s", "Male Lead");  
210 System.out.printf("%-20s\n", "Female Lead");   
211 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nymf"); }   
212 System.out.println();  
213   
214 // printing Bonus 2  
215 System.out.println("Bonus 2: Female Lead Last Name Descending via Merge Sort");  
216 System.out.println("--------------------------------------------------------");  
217 sortMerge(shows, 0, shows.length - 1, 2, 3);  
218 System.out.printf("%-20s", "Name");  
219 System.out.printf("%-20s", "Year");  
220 System.out.printf("%-20s", "Male Lead");  
221 System.out.printf("%-20s\n", "Female Lead");   
222 for(int i = 0; i < shows.length; i++) { shows[i].printInfo("nymf"); }   
223 System.out.println();  
224   
225 // close file  
226 in.close();  
227 }   
228 /\*~~~~~~~~~~~~~~~~~\*/  
229 /\*~~ END OF MAIN ~~\*/  
230 /\*~~~~~~~~~~~~~~~~~\*/  
231   
232   
233 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
234 \* removeReturn() \*  
235 \* this method removes the return after the end of each row from the \*  
236 \* CSV file \*  
237 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
238 public static String removeReturn(Scanner in) {  
239 String temp = in.next();  
240 temp = temp.replace("\r", "");  
241 return temp;  
242 }  
243   
244   
245 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
246 \* printShows() \*  
247 \* this method prints all of the information for each show \*  
248 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
249 public static void printShows(TVShow[] shows) {  
250 for(int i = 0; i < (shows.length - 1); i++) {  
251 System.out.println(shows[i].toString());  
252 }  
253 }  
254   
255   
256 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
257 \* sortInsertion() \*  
258 \* this method uses insertion sort to sort show category in ascending\*  
259 \* or descending order: \*  
260 \* order: 1 = ascending; 2 = descending \*  
261 \* category: 1 = name; 2 = season; 3 = genre \*  
262 \* hold considered show in temp \*  
263 \* while next index is bigger than previous, switch places \*  
264 \* place temp at next free index \*  
265 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
266 public static void sortInsertion(TVShow[] shows, int order, int category) {  
267 for(int i = 0; i < shows.length; i++) {  
268 TVShow next = shows[i];  
269 int index = i;  
270   
271 if((order == 1) && (category == 1)) { // ascending, name  
272 while((index > 0) && ((shows[index - 1].getName().compareTo(next.getName()) > 0))) { // while name is bigger  
273 shows[index] = shows[index - 1]; // move smaller name closer to front  
274 index--;  
275 }  
276 }  
277 else if((order == 2) && (category == 1)) { // descending, name  
278 while((index > 0) && ((shows[index - 1].getName().compareTo(next.getName()) < 0))) { // while name is smaller  
279 shows[index] = shows[index - 1]; // move bigger name closer to front  
280 index--;  
281 }  
282 }  
283 else if((order == 1) && (category == 2)) { // ascending, season  
284 while((index > 0) && ((shows[index - 1].getSeasons() > next.getSeasons()))) { // while seasons is bigger  
285 shows[index] = shows[index - 1]; // move smaller name closer to front  
286 index--;  
287 }  
288 }  
289 else if((order == 2) && (category == 2)) { // descending, season  
290 while((index > 0) && ((shows[index - 1].getSeasons() < next.getSeasons()))) { // while seasons is smaller  
291 shows[index] = shows[index - 1]; // move bigger name closer to front  
292 index--;  
293 }  
294 }  
295 else if((order == 1) && (category == 3)) { // ascending, genre  
296 while((index > 0) && ((shows[index - 1].getGenre().compareTo(next.getGenre()) > 0))) { // while genre is bigger  
297 shows[index] = shows[index - 1]; // move smaller name closer to front  
298 index--;  
299 }  
300 }  
301 else if((order == 1) && (category == 3)) { // descending, genre  
302 while((index > 0) && ((shows[index - 1].getGenre().compareTo(next.getGenre()) < 0))) { // while genre is smaller  
303 shows[index] = shows[index - 1]; // move bigger name closer to front  
304 index--;  
305 }  
306 }  
307 shows[index] = next;  
308 }  
309 }  
310   
311   
312 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
313 \* sortSelection() \*  
314 \* this method uses selection sort to sort show category in ascending\*  
315 \* or descending order: \*  
316 \* order: 1 = ascending; 2 = descending \*  
317 \* category: 1 = year; 2 = genre; 3 = male \*  
318 \* determine which one is larger, set that index to max \*  
319 \* store show at considered index in temp \*  
320 \* set show at index i to show at max index \*  
321 \* replace show at max index with show stored in temp \*  
322 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
323 public static void sortSelection(TVShow[] shows, int order, int category) {  
324 int i, ii, index;  
325 TVShow temp;  
326   
327 for(i = (shows.length - 1); i >= 0; i--) {  
328 index = 0;  
329 for(ii = 0; ii <= i; ii++) {  
330 if((order == 1) && (category == 1)) { // ascending, year  
331 if(shows[ii].getYear() > shows[index].getYear()) { index = ii; }  
332 }  
333 else if((order == 2) && (category == 1)) { // descending, year  
334 if(shows[ii].getYear() < shows[index].getYear()) { index = ii; }  
335 }  
336 else if((order == 1) && (category == 2)) { // ascending, genre  
337 if(shows[ii].getGenre().compareTo(shows[index].getGenre()) > 0) { index = ii; }  
338 }  
339 else if((order ==2) && (category == 2)) { // descending, genre  
340 if(shows[ii].getGenre().compareTo(shows[index].getGenre()) < 0) { index = ii; }  
341 }  
342 else if((order == 1) && (category == 3)) { // ascending, male  
343 if(lastName(shows[ii].getMale()).compareTo(lastName(shows[index].getMale())) > 0) { index = ii; }  
344 }  
345 else if ((order == 2) && (category == 3)) { // descending, male  
346 if(lastName(shows[ii].getMale()).compareTo(lastName(shows[index].getMale())) < 0) { index = ii; }  
347 }  
348 }  
349   
350 temp = shows[i]; // temporarily stores shows at index i  
351 shows[i] = shows[index]; // switches shows at index index to shows at index i  
352 shows[index] = temp; // sets shows at index to the temporary  
353 }  
354 }  
355   
356   
357 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
358 \* sortMerge() \*  
359 \* this method uses merge sort to sort show category in ascending or \*  
360 \* descending order: \*  
361 \* order: 1 = ascending; 2 = descending \*  
362 \* category: 1 = network; 2 = episodes; 3 = female \*  
363 \* determines mid point of array \*  
364 \* sorts array into two sections at mid: first and second half \*  
365 \* recursion: calls sortMerge() to sort \*  
366 \* when finished sorting (min == max), calls mergeMerge helper \*  
367 \* \*  
368 \* mergeMerge () \*  
369 \* this is the helper method for sortMerge() that merges the first \*  
370 \* and second halves of sorted arrays: \*  
371 \* if the next first half index exceeds the middle number (too \*  
372 \* big), start adding the second half array \*  
373 \* if the next second half index exceeds the max number (too \*  
374 \* big), start adding the first half array \*  
375 \* if the next first half index considered is greater than the \*  
376 \* next second half index considered, add the first half \*  
377 \* index to the array \*  
378 \* otherwise, add the next second half index to the array \*  
379 \* set shows to temp array (which is sorted) \*  
380 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
381 public static void sortMerge(TVShow[] shows, int min, int max, int order, int category) {  
382 if(min == max) { return; } // base case  
383 int mid = (min + max) / 2;  
384   
385 sortMerge(shows, min, mid, order, category); // sorting first half  
386 sortMerge(shows, mid + 1, max, order, category); // sorting second half  
387   
388 mergeMerge(shows, min, mid, max, order, category); // merging first and second half  
389 }  
390   
391 // helper method to merge first and second half of network sorting  
392 public static void mergeMerge(TVShow[] shows, int min, int mid, int max, int order, int category) {  
393 TVShow[] temp = new TVShow[max - min + 1];  
394 int fNext = min; // first half next index  
395 int sNext = mid + 1; // second half next index  
396 int next = 0; // next shows index  
397   
398 while((fNext <= mid) || (sNext <= max)) {   
399 if(fNext > mid) { // if the first half is larger than the middle  
400 temp[next] = shows[sNext]; // start adding remaining second half to array  
401 sNext++;  
402 }  
403 else if(sNext > max) { // if the second half is larger than max  
404 temp[next] = shows[fNext]; // start adding remaining first half to array  
405 fNext++;  
406 }  
407 else if((order == 1) && (category == 1) && (shows[fNext].getNetwork().compareTo(shows[sNext].getNetwork()) < 0)) { // ascending, network  
408 temp[next] = shows[fNext]; // add next first to array  
409 fNext++;  
410 }  
411 else if((order == 2) && (category == 1) && (shows[fNext].getNetwork().compareTo(shows[sNext].getNetwork()) > 0)) { // descending, network  
412 temp[next] = shows[fNext]; // add next first to array  
413 fNext++;  
414 }  
415 else if((order == 1) && (category == 2) && (shows[fNext].getEpisodes() < shows[sNext].getEpisodes())) { // ascending, episodes  
416 temp[next] = shows[fNext]; // add next first to array  
417 fNext++;  
418 }  
419 else if((order == 2) && (category == 2) && (shows[fNext].getEpisodes() > shows[sNext].getEpisodes())) { // descending, episodes  
420 temp[next] = shows[fNext]; // add next first to array  
421 fNext++;  
422 }  
423 else if ((order == 1) && (category == 3) && (lastName(shows[fNext].getFemale()).compareTo(lastName(shows[sNext].getFemale())) < 0)) { // ascending, female  
424 temp[next] = shows[fNext]; // add next first to array  
425 fNext++;  
426 }  
427 else if ((order == 2) && (category == 3) && (lastName(shows[fNext].getFemale()).compareTo(lastName(shows[sNext].getFemale())) > 0)) { // descending, female  
428 temp[next] = shows[fNext]; // add next first to array  
429 fNext++;  
430 }  
431 else { // if next in first half is too big, add next in second half  
432 temp[next] = shows[sNext];  
433 sNext++;  
434 }  
435 next++;  
436 }  
437   
438 for(int ii = min; ii <= max; ii++) { shows[ii] = temp[ii - min]; } // setting array to temp array  
439 }  
440   
441   
442 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
443 \* searchSequential() \*  
444 \* this method uses sequential search to find number x in category: \*  
445 \* category: 1 = seasons; 2 = episodes \*  
446 \* loops through each element in array \*   
447 \* if matches, adds to arrayList \*  
448 \* returns arrayList \*  
449 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
450 public static ArrayList<Integer> searchSequential(TVShow[] shows, int x, int category){  
451 ArrayList<Integer> temp = new ArrayList<>();  
452 for(int i = 0; i < shows.length; i++) {  
453 if(category == 1) { // seasons  
454 if(shows[i].getSeasons() == x) { temp.add(i); }  
455 }  
456 else { // episodes  
457 if(shows[i].getEpisodes() > x) { temp.add(i); }  
458 }  
459 }  
460 return temp;  
461 }  
462   
463   
464 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
465 \* searchBinary() \*  
466 \* this method uses binary search to find all elements that match \*  
467 \* category given min and max: \*  
468 \* category: 1 = network; 2 = genre \*  
469 \* determine mid point \*  
470 \* determine if category at mid point matches search \*  
471 \* if so, recursion: send first and second half back to \*  
472 \* searchBinary() \*  
473 \* if not, add -1 to array (indicates no return) \*  
474 \* determine if mid point is lower than category \*  
475 \* if so, search second half of array \*  
476 \* otherwise, search first half of array \*  
477 \* \*  
478 \* if nothing matches, return -1 (indicates no return) \*  
479 \* remove all -1: if no match, will return empty array, otherwise\*  
480 \* return all indexes of matches \*  
481 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
482 public static ArrayList<Integer> searchBinary(TVShow[] shows, String seeking, int min, int max, int category) {  
483 ArrayList<Integer> temp = new ArrayList<>(); // all matching elements' indexes  
484 ArrayList<Integer> recTemp = new ArrayList<>(); // matching elements in recursion calls  
485   
486 if(seeking.compareTo("CBS") == 0) { sortMerge(shows, 0, shows.length - 1, 2, 1); } // merge sort  
487 if(seeking.compareTo("Fantasy") == 0) { sortSelection(shows, 2, 2); } // selection sort  
488 if(seeking.compareTo("Drama") == 0) { sortInsertion(shows, 3, 3); } // insertion sort  
489   
490 if(min <= max) {  
491 int mid = (min + max) / 2;  
492   
493 if(((category == 1) && (seeking.compareTo(shows[mid].getNetwork()) == 0)) ||   
494 ((category == 2) && (seeking.compareTo(shows[mid].getGenre()) == 0))) { // if category at mid matches  
495 temp.add(mid);  
496 recTemp = searchBinary(shows, seeking, min, mid - 1, category); // search again in first half  
497 for(int i = 0; i < recTemp.size(); i++) { temp.add(recTemp.get(i)); } // add recursion results  
498 recTemp = searchBinary(shows, seeking, mid + 1, max, category); // search again in second half  
499 for(int i = 0; i < recTemp.size(); i++) { temp.add(recTemp.get(i)); } // add recursion results  
500 }  
501 else if(((category == 1) && (seeking.compareTo(shows[mid].getNetwork()) < 0)) ||   
502 ((category == 2) && (seeking.compareTo(shows[mid].getGenre()) < 0))) { // if category is less than mid  
503 return searchBinary(shows, seeking, mid + 1, max, category); // search second half  
504 }  
505 else { return searchBinary(shows, seeking, min, mid - 1, category); } // else search first half  
506 }  
507   
508 if(temp.isEmpty()) { // returns if no matches  
509 temp.add(-1);  
510 return temp;  
511 }  
512   
513 temp.removeAll(Arrays.asList(-1)); // removes all all unnecessary no matches  
514 return temp;  
515 }  
516   
517   
518 /\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
519 \* lastName() \*  
520 \* this method finds last name (substring starting after space) \*  
521 \*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/  
522 public static String lastName(String name) {  
523 return name.substring(name.lastIndexOf(" ") + 1);  
524 }  
525 }

1 /\*\*  
 2 \* This is the TVShow class that holds instance variables for each TVShow object:  
 3 \* name, year premiered, seasons, episodes, network, genre, male lead, and   
 4 \* female lead  
 5 \* This class contains a constructor and getters for each instance variable. It  
 6 \* also contains the following method:  
 7 \*  
 8 \* printInfo(String options): accepts a string of letters corresponding to  
 9 \* the information that is requested to be gotten from the object:  
10 \* n = name  
11 \* y = year  
12 \* s = seasons  
13 \* e = episodes  
14 \* w = network  
15 \* g = genre  
16 \* m = male lead  
17 \* f = female lead  
18 \*  
19 \* @author Stephanie Gremillion  
20 \* @version 17.0.2  
21 \* @since 2022/07/20  
22 \*/  
23   
24 public class TVShow {  
25 // variables  
26 public String name, network, genre, male, female;  
27 public int year, seasons, episodes;  
28   
29 // constructor  
30 public TVShow(String na, int y, int s, int e, String ne, String g, String m, String f) {  
31 name = na;  
32 year = y;  
33 seasons = s;  
34 episodes = e;  
35 network = ne;  
36 genre = g;  
37 male = m;  
38 female = f;  
39 }  
40   
41 // getters  
42 public String getName() { return name; }  
43 public int getYear() { return year; }  
44 public int getSeasons() { return seasons; }  
45 public int getEpisodes() { return episodes; }  
46 public String getNetwork() { return network; }  
47 public String getGenre() { return genre; }  
48 public String getMale() { return male; }  
49 public String getFemale() { return female; }  
50   
51 // print info  
52 public void printInfo(String options) {  
53 if(options.contains("n")) { System.out.printf("%-20s", this.name); }  
54 if(options.contains("y")) { System.out.printf("%-20d", this.year); }  
55 if(options.contains("s")) { System.out.printf("%-20d", this.seasons); }  
56 if(options.contains("e")) { System.out.printf("%-20d", this.episodes); }  
57 if(options.contains("w")) { System.out.printf("%-20s", this.network); }  
58 if(options.contains("g")) { System.out.printf("%-20s", this.genre); }  
59 if(options.contains("m")) { System.out.printf("%-20s", this.male); }  
60 if(options.contains("f")) { System.out.printf("%-20s", this.female); }  
61 System.out.println();  
62 }  
63 }