|  |  |  |
| --- | --- | --- |
| **Fòrmules útils** | | |
| Factorial n! == n\*(n-1)\*(n-2)\*…\*1  0! == 1 Progressió aritmètica La suma **Sn= a1+a2+.......+an,** de los n primers termes d’una progresió aritmètica, és:  Com a casos particulars tenim:  **Suma dels N primers números:**    **Suma dels numerous entre N I M (N>M):** | Progressió geomètrica La suma **Sn= a1+a2+.......+an** dels n primers termes d’una progresió geomètrica de raó r és:    El producte **Pn= a1·a2·.......·an** dels n primers termes d’una progresió geomètrica és: |  |

|  |
| --- |
| **Eficiència de les EEDD** |
|  |

|  |  |
| --- | --- |
| **Estructures de dades en Java** | |
| **LLISTES**  Estructura seqüencial, on cada element té un índex o posició  **INTERFÍCIE**  **List<Element>**  **IMPLEMENTACIONS**  **ArrayList <Element>** 🡪 Accés ràpid  **LinkedList<Element>** 🡪 Insercions i esborrat ràpids  **Stack<Element>** 🡪 Pila  **Vector <Element>** 🡪 En desús | **CONJUNT**  Els elements no tenen un ordre i no es permeten duplicats.  **INTERFÍCIE**  **Set<Element>**  **IMPLEMENTACIONS**  **HashSet <Element>** 🡪 Utilitza una taula hash  **LinkedHashSet<Element>** 🡪 Taula hash que utilitza llistes doblement enllaçades  **TreeSet<Element>** 🡪 Utilitza arbres  **DICCIONARI (TAULA ASSOCIATIVA)**  Cada element té associada una clau que utilitzarem per recuperar-lo (en comptes de l’índex)  **INTERFÍCIE**  **Map <Clau, Valor>**  **IMPLEMENTACIONS**  **HashMap <Clau, Valor>** 🡪 Utilitza una taula hash  **LinkedHashMap <Clau, Valor**> 🡪 Taula hash que utilitza llistes doblement enllaçades  **TreeMap <Clau, Valor>** 🡪 Utilitza arbres, Claus ordenades, pero sense repeticions (interficie SortedMap)  Classe static **Collections** 🡪 Ordenacions, max, min, shuffle reverse, replace, rotate  Understanding NavigableMap  **NavigableMap**(S’implementa amb un **TreeMap**): is sub interface of SortedMap interface, so it inherits all behaviors of a sorted map like range view, endpoints and comparator access operations. In addition, the NavigableMap interface provides navigation methods and descending views similar to NavigableSet. And due to the nature of a map which stores key-value mappings, the additional APIs are designed for both keys and key-value entries in the map.  **Operations on key-value mappings (entries):**  **lowerEntry(K key):** returns a key-value mapping associated with the greatest key strictly less than the given key.  **floorEntry(K key):** returns a key-value mapping entry which is associated with the greatest key less than or equal to the given key.  **ceilingEntry(K key):** returns an entry associated with the lest key greater than or equal to the given key.  **higherEntry(K key):** returns an entry associated with the least key strictly greater than the given key.  Note that all these methods return null if there is no such key.  **firstEntry():** returns a key-value mapping associated with the least key in the map, or null if the map is empty.  **lastEntry():** returns a key-value mapping associated with the greatest key in the map, or null if the map is empty.  **descendingMap():** returns a reverse order view of the mappings contained in the map.  **pollFirstEntry():** removes and returns a key-value mapping associated with the least key in the map, or null if the map is empty.  **pollLastEntry():** removes and returns a key-value mapping associated with the greatest key in the map, or null if the map is empty.  **Operations on keys only:**  **lowerKey(K key):** returns the greatest key strictly less than the given key.  **floorKey(K key):** returns the greatest key less than or equal to the given key.  **ceilingKey(K key):** returns the least key greater than or equal to the given key.  **higherKey(K key):** returns the least key strictly greater than the given key.  **descendingKeySet():** returns a NavigableSet containing the keys in reverse order. |

|  |  |  |  |
| --- | --- | --- | --- |
| Valors màxims i mínims | | | |
| byte | 8 bits | -128 | 127 |
| short | 16 bits | -32768 | 32767 |
| char | 16 bits | 0 | 65535 |
| int | 32 bits | -2.147.483.648 | 2.147.483.647 ~ 2^31 ~ 10^9 |
| long | 64 bits | -9.223.372.036.854.775.808 | 9.223.372.036.854.775.807 ~ 2^63 ~ 10^18 |
| float | 32 bits | 1.4E-45 | 3.4028235E38 |
| double | 64 bits | 4.9E-324 | 1.79769313486231575E308 |

|  |
| --- |
| **Combinatòria** |
| Permutaciones  Sea un conjunto de *n* elementos. A las ordenaciones que se pueden hacer con estos *n* elementos sin repetir ningún elemento y utilizándolos todos se las denomina *permutaciones*. El número de permutaciones que se pueden realizar coincide con el factorial de *n*, y su cálculo es:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb0.gif  *Ejemplo*: ¿De cuántas maneras distintas podemos alinear a seis personas en una fila?  *Respuesta*: De 6! = 6 · 5 · 4 · 3 · 2 · 1 = 720 maneras (permutaciones de 6 elementos).   Variaciones  Sea un conjunto de *n* elementos. Supongamos que deseamos ordenar *r* elementos de entre los *n*. A cada una de estas ordenaciones la denominamos *variación*. El número de variaciones que se pueden hacer de los *n* elementos tomados de *r* en *r* es:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb3.gif  *Ejemplo*: En una carrera de velocidad compiten diez atletas. ¿De cuántas maneras distintas podría estar formado el podio? (el podio lo forman el primer, el segundo y el tercer clasificado)  *Respuesta*: Cada podio posible es una variación de diez elementos tomado de tres en tres.  Por tanto, el número posible de podios es:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb4.gif  Variaciones con repetición  Sea un conjunto de *n* elementos. Supongamos que se trata de ordenar *r* elementos que pueden estar repetidos. Cada ordenación es una *variación con repetición*. El número de variaciones con repetición para un conjunto de *n* tomados de *r* en *r* es :  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb5.gif  *Ejemplo:* En una urna tenemos cinco bolas numeradas del 1 al 5. Se extraen tres bolas sucesivamente con reposición (devolviendo cada vez la bola a la urna). ¿Cuántos resultados distintos es posible obtener?  *Respuesta*: Se trata de variaciones con repetición de un conjunto de cinco bolas tomadas de tres en tres.  En total tendremos:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb6.gif  Combinaciones  Cuando se trata de contar el número de subconjuntos de *x* elementos en un conjunto de *n* elementos tenemos lo que se denomina *combinaciones de* x *elementos en un conjunto de* n. El cálculo del contaje se hace mediante el número combinatorio, de la manera siguiente:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb1.gif  *Ejemplo*: ¿De cuántas maneras podemos elegir, en la urna anterior (recordemos que había cinco bolas), tres bolas en una única extracción?  *Respuesta*: Serán combinaciones de cinco elementos tomados de tres en tres, por tanto, tendremos:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb2.gif  Permutaciones con repetición  Sea un conjunto de *n* elementos, de entre los cuales tenemos *a* elementos indistinguibles entre sí, *b* elementos indistinguibles entre sí, *c* elementos indistinguibles entre sí, etc. Cada ordenación de estos elementos se denominará *permutación con repetición*. El número de permutaciones con repetición es:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb7.gif  *Ejemplo*: ¿Cuantas palabras con sentido o sin él pueden formarse con las letras PATATA?  *Respuesta:*Tenemos tres veces la letra A, dos veces la T y una vez la P.  Por tanto, serán:  http://www.ub.edu/stat/GrupsInnovacio/Statmedia/demo/Temas/Capitulo1/Images/comb8.gif |

|  |
| --- |
| **Plantilla** |
| **public** **static** **void** main(String[] args) {  Scanner s = **new** Scanner(System.***in***);  StringBuilder resposta = **new** StringBuilder();    **int** nCasos = s.nextInt();  **for**(**int** cas=0; cas<nCasos;cas++)  {  }    **int** num = s.nextInt();  s.nextLine();  **while** (num!=0)  {  System.***out***.println(total);  resposta.append(total);  resposta.append(‘\n’);  **if** (resposta.length()>1000)  {  System.***out***.print(resposta);  }  num=s.nextInt();  s.nextLine();  }  s.close();  System.***out***.print(resposta);  } |

|  |
| --- |
| **Descomposició en primers** |
| Map<Integer,Integer> factorsPrimers = **new** HashMap<>();    factorsPrimers.clear();  num = s.nextInt();  divisor = 2;  **while** ((num != 1) && (divisor <= Math.*ceil*(Math.*sqrt*(num))))  {  **while** (num % divisor == 0)  {  num/= divisor;  factorsPrimers.put(divisor,0);  }  divisor ++;  }  factorsPrimers.put(num, 0);  producte = 1;  **for** (**int** factor:factorsPrimers.keySet())  producte\*=factor;    //producte == num; |

|  |
| --- |
| **Lectura amb BufferedReader** |
| **public** **static** **class** FastScanner {  BufferedReader br;  StringTokenizer st;  **public** FastScanner(String s) {  **try** {  br = **new** BufferedReader(**new** FileReader(s));  } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  **public** FastScanner() {  br = **new** BufferedReader(**new** InputStreamReader(System.***in***));  }  String nextToken() {  **while** (st == **null** || !st.hasMoreElements()) {  **try** {  st = **new** StringTokenizer(br.readLine());  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  **return** st.nextToken();  }  String nextLine()  {  String cadena="";  **try** {  cadena= br.readLine();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  **return** cadena;  }    **int** nextInt() {  **return** Integer.*parseInt*(nextToken());  }  **long** nextLong() {  **return** Long.*parseLong*(nextToken());  }  **double** nextDouble() {  **return** Double.*parseDouble*(nextToken());  }  } |

| **String.format** | |
| --- | --- |
| **Integers, d format.** This is a good first example. We add the values 10, 20 and 30 to a format string. We specify the format string as the first argument to format.  // Call String.format with three integer codes.  String result = **String.format**("One: %1$d Two: %2$d Three: %3$d",10, 20, 30);  System.out.println(result); | **Output**  One: 10 Two: 20 Three: 30 |
| **Pad with zeros.** Many format codes can be used with String.format. Here we pad a number with zeros on the left side. The first 0 means "pad with zeros" and the 5 means "use five digits."  for (int i = 0; i < 5; i++) {  // Pad with zeros and a width of 5 chars.  String result = **String.format**("%1$05d %2$05d", i, i + 10);  System.out.println(result);  } | **Output**  00000 00010  00001 00011  00002 00012  00003 00013  00004 00014 |
| **Precision, double.** We can specify the precision of a double or float. We use the "f" to indicate a floating-point. Before this we specify ".3" to mean "three numbers after the decimal."  **Note:**As this example shows, the "f" format will round the number up, not truncate digits.  double number = 1.23456;  // Specify precision with format.  String value = **String.format**("Three numbers after decimal: %1$.3f",number);  System.out.println(value); | **Output**  Three numbers after decimal: 1.235 |
| **Insert strings.** We can insert a string with the "s" format. This example shows three different syntax forms. We can specify just "%s," use the percentage sign "%," or use a "$" in between.  **Index:**It helps to use an index to reference the argument (like "%1") if we reuse the argument, or the references are different in order.  **Opinion:**I think just using "%s" when possible yields the clearest, easiest-to-read code. Keep it simple.  **Java program that inserts string with format**  String first = "Marcus";  String last = "Aurelius";  // Use simple string format.  String value = **String.format**("%s %s", first, last);  System.out.println(value);  // Use indexes before simple string format.  value = **String.format**("%1s %2s", first, last);  System.out.println(value);  // Use $ symbol before string character.  value = **String.format**("%1$s %2$s", first, last);  System.out.println(value); | **Output**  Marcus Aurelius  Marcus Aurelius  Marcus Aurelius |
| **Argument order.** This example uses both a string and an integer in one format string. It also inserts the second argument first—it uses the "%2" syntax for this.  **Java program that reorders arguments**  String id = "ART1";  int number = 100;  // Place the last argument first.  // ... Use integer and string in same format string.  String value = **String.format**("%2$d %1$s", id, number);  System.out.println(value); | **Output**  100 ART1 |
| **Padding, right and left.** Here we use right-padding and left-padding in a format string. We can apply padding to strings, numbers or other values.[**Padding**](https://www.dotnetperls.com/padding-java)  **Right:**To specify padding on the right (ten spaces on the right) we use -10s. The result string is 10 chars.  **Left:**To add padding to the left (right justify text) to 10 chars, we use 10s. No plus sign is used.  **Java program that uses padding, right and left**  String[] lefts = { "cat", "dog", "bird", "elephant" };  String[] rights = { "orange", "black", "blue", "grey" };  // Loop over both arrays and display paired strings.  for (int i = 0; i < lefts.length; i++) {  String left = lefts[i];  String right = rights[i];  // Add padding to the right.  // ... Add padding to the left.  String value = **String.format**("%1$-10s %2$10s", left, right);  System.out.println(value); | **Output**  cat orange  dog black  bird blue  elephant grey |
| IMPORTANT!! el %n fa salt de línia, de tal manera que:  System.out.println(“%d”,n) == System.out.format(“%d%n”,n) |  |
| **long n = 461012;**  System.out.format("%d%n", n);  System.out.format("%08d%n", n);  System.out.format("%+8d%n", n);  System.out.format("%,8d%n", n);  System.out.format("%+,8d%n%n", n); | "461012"  "00461012"  " +461012"  " 461,012"  "+461,012" |
| **double pi = Math.PI;**  System.out.format("%f%n", pi);  System.out.format("%.3f%n", pi);  System.out.format("%10.3f%n", pi);  System.out.format("%-10.3f%n", pi); | "3.141593"  "3.142"  " 3.142"  "3.142" |
| System.out.format(Locale.FRANCE,"%-10.4f%n%n", pi);  Calendar c = Calendar.getInstance();  System.out.format("%tB %te, %tY%n", c, c, c);  System.out.format("%tl:%tM %tp%n", c, c, c);  System.out.format("%tD%n", c); | "3,1416"  "May 29, 2006"  "2:34 am"  "05/29/06" |

|  |
| --- |
| **Entrada/Sortida** |
| Lectura:  Scanner s=new Scanner(System.in);  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));  Mètodes Scanner:  s.hasNext() – Retorna true si hi ha més dades per llegir s.next() – Retorna el següent String  s.nextDouble() – Retorna el següent double s.nextInt() – Retorna el següent int s.nextLong() – Retorna el següent long  s.nextLine() – Retorna un String amb tota la línia (Important per descartar casos si no s’ha llegit lúltim)  Mètodes BufferedReader:  br.read() – Retorna el codi ASCII del següent caràcter  br.readLine() – Retorna un String amb tota la línia (convinar amb String.split())  String to Other Integer.parseInt(String s) Double.parseInt(String s)  Escriptura:  System.out.println(String s); Per imprimir un string amb salt de línia  System.out.print(String s); Per imprimir un string sense salt de línia  System.out.printf(String format, String s); Per imprimir un string amb format sense salt de línia |

|  |  |  |
| --- | --- | --- |
| Converter | Flag | Explanation |
| d |  | A decimal integer. |
| f |  | A float. |
| n |  | A new line character appropriate to the platform running the application. You should always use %n, rather than \n. |
| tB |  | A date & time conversion—locale-specific full name of month. |
| td, te |  | A date & time conversion—2-digit day of month. td has leading zeroes as needed, te does not. |
| ty, tY |  | A date & time conversion—ty = 2-digit year, tY = 4-digit year. |
| tl |  | A date & time conversion—hour in 12-hour clock. |
| tM |  | A date & time conversion—minutes in 2 digits, with leading zeroes as necessary. |
| tp |  | A date & time conversion—locale-specific am/pm (lower case). |
| tm |  | A date & time conversion—months in 2 digits, with leading zeroes as necessary. |
| tD |  | A date & time conversion—date as %tm%td%ty |
|  | 08 | Eight characters in width, with leading zeroes as necessary. |
|  | + | Includes sign, whether positive or negative. |
|  | , | Includes locale-specific grouping characters. |
|  | - | Left-justified.. |
|  | .3 | Three places after decimal point. |
|  | 10.3 | Ten characters in width, right justified, with three places after decimal point. |

|  |
| --- |
| **Interfície comparable en Java** |
| **public** **class** Ordenacions {  **public** **class** Numero **implements** Comparable<Numero> {  **private** **int** valor;  **public** Numero(**int** valor) { **this**.valor = valor; }  **public** **int** compareTo(Numero altreNumero) {  **if**(valor<altreNumero.valor)  **return** -1; //Retornem un negatiu si valor<altreNumero.valor  **else** **if**(valor==altreNumero.valor)  **return** 0; //Retornem zero si valor==altreNumero.valor  **else** //if(valor>altreNumero.valor)  **return** 1; //Retornem un positiu si valor>altreNumero.valor    //Generalment s'abreuja amb un càlcul:  **return** **this**.valor - altreNumero.valor;  }  }  **public** **class** Cadira {  **private** **int** pes;  **private** **int** alçada;  **public** Cadira(**int** pes, **int** alçada) {  **this**.pes = pes;  **this**.alçada = alçada;  }  /\* Ometem els getters i els setters \*/  }  **public** **class** ComparadorDeCadiresPerPes **implements** Comparator<Cadira> {  **public** **int** compare(Cadira cadira1, Cadira cadira2) {  **return** cadira1.getPes() - cadira2.getPes();  }  }  **public** **class** ComparadorDeCadiresPerAlçada **implements** Comparator<Cadira> {  **public** **int** compare(Cadira cadira1, Cadira cadira2) {  **return** cadira1.getAlçada() - cadira2.getAlçada();  }  }    **public** **static** **void** main(String[] args)  {  List<Numero> numeros = **new** ArrayList<Numero>();  ...  Collections.sort(numeros);  List<Cadira> cadires = **new** ArrayList<Cadira>();  // Ordenem per pes:  Collections.sort(cadires, **new** ChairWeightComparator());  // Ordenem per alçada:  Collections.sort(cadires, **new** ChairHeightComparator());  // Podem crear comparadors anònims;  // Ordenem per relació pes/alçada:  Collections.sort(cadires, **new** Comparator<Cadira>() {  **public** **int** compare(Cadira cadira1, Cadira cadira2) {  **return** (**int**)((**double**)cadira1.getPes()/cadira1.getAlçada()  - (**double**)cadira2.getPes()/cadira2.getAlçada());  }  });  // Ordenem per relació per Pes de manera decreixent:  Collections.sort(cadires, **new** Comparator<Cadira>() {  **public** **int** compare(Cadira cadira1, Cadira cadira2) {  **return** **new** ComparadorDeCadiresPerPes().compare(cadira2, cadira1)  }  });  }} |

|  |  |
| --- | --- |
|  | |
|  |  |