***Fecha de entrega máxima***: 14 de julio de 2021

***Hora***: 11: 50 pm, envíos después de esta hora NO SERÁN RECIBIDOS, por ser extemporáneos.

***Cantidad de integrantes por grupo***: máximo 2, SÓLO UNO ENVÍA EL ARCHIVO, pero ambos son responsables por el envío, por ende, si el internet falla, el otro integrante asume la responsabilidad. Los grupos de trabajo podrán conformarse con estudiantes del grupo 1 o grupo 3 de programación I, franja diurna.

***Envío:*** ÚNICAMENTE por la plataforma Moodle, luego de la hora indicada NO PODRÁ SUBIR el archivo, el enlace se inactivará. No se aceptan otros medios de envío. Se sugiere que inicie el envío, a más a tardar, 20 minutos antes de la hora indicada.

***Formato de entrega***: ÚNICAMENTE Archivo **WORD.** NO SE ACEPTAN, entre otros, PDFs, fotos de código, proyectos en Eclipse, texto o imágenes con fondo negro, porque dificulta la impresión. Trabajar sobre el mismo archivo que descargan, para conservar el formato. No sacar el código de las tablas ni cambiar el orden de las funciones. En caso de encontrar un error en el código debe corregirlo. No podrá cambiar la signatura de las funciones, a menos que el docente autorice mediante correo electrónico.

***Nombre del archivo Word:*** debe contener el nombre de los integrantes PedroPerez\_MarioRodriguezParcial.docx e internamente (en el cuerpo del parcial) debe también tener el nombre de los integrantes.

**El no cumplir con estos requisitos resta automáticamente 0.5 a la nota obtenida en el parcial**.

Luego de enviar el parcial, por favor verifiquen que si puede descargarse y descomprimirse (ojo porque a veces me mandan archivos corruptos y en tal caso no podría calificarles ese archivo). Deben enviar un archivo comprimido, no un link.

Si se presentan funciones repetidas en diferentes grupos, esto dará lugar a la anulación del parcial

de dichos grupos.

NOTA: queda pendiente la sustentación de parcial. La nota definitiva es nota de código (que se

califica de 0 a 5) por la nota de la sustentación (se califica de 0 a 1).

Nombre: Camilo Esteban Davila Sierra Doc. ID: 1004917304

Nombre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Doc. ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PARCIAL 2**

Programación I

1. Dado un arreglo que contiene las cédulas de varias personas, devolver otro arreglo únicamente con las cédulas que contienen al menos un digito seguidos por sus 2 siguientes dígitos consecutivos (deben ir en posiciones consecutivas). No usar métodos de la clase String.

Ejemplo de cédulas:

“12333389”  Cumple

“2334589”  cumple

“142389” No cumple

Estas son las instrucciones para pasar un String a vector de caracteres

String id = "12456699";

char[] caracteres= id.toCharArray();

(1,6 ptos)

|  |
| --- |
| public static boolean verificar3DigitosConsecutivos(String numero)  {  boolean respuesta = false;  char charArreglo[] = numero.toCharArray();  for (int i = 0; i < numero.length(); i++) {  char charActual = charArreglo[i];  if (i < 8) {  char nextChar = charArreglo[i+1];  char nextChar2 = charArreglo[i+2];  if (nextChar-'0' == (charActual-'0')+1) {  if (nextChar2-'0' == nextChar-'0'+1) {  respuesta = true;  }  }  }  }  return respuesta;      } |
| **public static** String[] devolverCedulas (String cedulas[])  {  String cedulasAprobadas[]= new String[cedulas.length];    for (int i = 0; i < cedulas.length; i++) {  boolean esAprobado = verificar(cedulas[i]);  cedulasAprobadas[i] = (esAprobado) ? cedulas[i] : "";  }  return cedulasAprobadas;    } |

2. Dado un vector de números informar el menor número apocalíptico. Un número n es apocalíptico si cumple que 2n tiene la secuencia 666.

Ejemplos de números apocalípticos son 157 y 192. Por ejemplo 157 es apocalíptico, porque 2^157 = 182687704666362864775460604089535377456991567872 ”\*

Dados los números tan grandes a los que puede dar lugar la potencia se sugiere usar la clase BigInteger. Visitar la página https://codingornot.com/enteros-grandes-big-integers-en-java para mayor información (1.2 ptos)

|  |
| --- |
| public static boolean isApocaliptico(Bigdecimal numero)  {  boolean changer = false;  String cadenaNumeros = serie.toString();  char arrayNumeros[] = cadenaNumeros.toCharArray();  for (int i = 0; i < arrayNumeros.length; i++) {  if (i < arrayNumeros.length-2) {  int firstPosition = arrayNumeros[i]-'0';  int secondPosition = arrayNumeros[i+1]-'0';  int thirdPosition = arrayNumeros[i+2]-'0';  if (firstPosition == 6 && secondPosition == 6 && thirdPosition == 6) {  changer = true;  }  }  }  return changer;  } |
| **public static** BigInteger devolverMenorApocaliptico(int exponentes[])  {  int index = 0;  String resultado = "Vacio";  BigDecimal numeroMenor;  BigDecimal apocalipticos[] = new BigDecimal[exponentes.length];  for (int i = 0; i < exponentes.length; i++) {  BigDecimal serieActual = new BigDecimal(2).pow(exponentes[i]);  boolean esApocaliptico = isApocaliptico(serieActual);  if (esApocaliptico) {  apocalipticos[index] = serieActual;  index++;  }  }  if (apocalipticos.length >= 2) {  numeroMenor = apocalipticos[0];  for (int i = 0; i < apocalipticos.length; i++) {  if(apocalipticos[i] != null){  int comparacion = apocalipticos[i].compareTo(numeroMenor);  if (comparacion == -1) {  numeroMenor = apocalipticos[i];  }  }  }  resultado = numeroMenor.toString();  }  return resultado;    } |

3. Dado un vector con números identificar si en él hay números primos y evaluar si hay al menos 2 primos consecutivos. Si se sabe por ejemplo que los primeros primos son 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 …. Y dentro del vector están por ejemplo el 29 y el 31 se debe retornar true, porque hay 2 primos consecutivos, si no se cumple esta regla se retorna false. El código no puede ser estático, debe ser generalizable. (2.2 ptos)

|  |
| --- |
| public static boolean isPrimo(int numero)  {  boolean retorno = true;  if (numero == 0 || numero == 1 || numero == 4) {  retorno = false;  }  for (int i = 2; i < numero/2; i++) {  retorno = (numero%i == 0)?false:true;  }  return retorno;  } |
| public static int devolverPrimoConsecutivo (int primo)  {  int retorno = 0;  int i = primo+2;  boolean breaker = false;    while (breaker == false) {  breaker = isPrimo(i);  if (breaker) {  retorno = i;  }else{  i +=2;  }  }  return retorno;  } |
| // recibe un vector con números y devuelve solo los primos  public static int[] devolverPrimos (int números[])  {//invocar a isPrimo  //sin usar  } |
| public static boolean verificarSiNumeroPrimoTieneSuConsecutivoEnVector(int primos[])  {//invocar a devolverPrimoConsecutivo  boolean retornoF = false;  boolean retornoV = false;  for (int i = 0; i < primos.length; i++) {  int caracterActual = primos[i];  if (isPrimo(caracterActual) && i <= primos.length-1) {  int primoConsecutivo = devolverPrimoConsecutivo(caracterActual);  if(primos[i+1] == primoConsecutivo){  retornoV = true;  }  }  }  retornoF = (retornoV == true)?true:false;  return retornoF;  } |
| public static boolean evaluarExistePrimosConsecutivos (int primos[])  {//invocar a verificarSiNumeroPrimoTieneSuConsecutivoEnVector  boolean retorno = false;  boolean verificando= verificarSiNumeroPrimoTieneSuConsecutivoEnVector(primos);  if(verificando){  retorno = true;  }  return retorno;  } |