Acosta Porcayo Alan Omar, Gutiérrez Grimaldo Alejandro, Medina Villa Samuel

Práctica 3. Utilerías y Clases de Uso

Resumen

Cuando se programa en un lenguaje de programación, resulta crucial utilizar las bibliotecas incorporadas en ese mismo lenguaje con el fin de ejecutar tareas habituales y repetitivas de manera más eficaz. Para lograrlo, es esencial tener un conocimiento profundo de las bibliotecas disponibles en el lenguaje y aprender a emplearlas de manera apropiada. Asimismo, es posible mejorar la eficiencia del código al utilizar las clases ofrecidas por estas bibliotecas, lo que simplifica el proceso de programación y eleva la calidad del programa

Introducción

En el mundo de la programación, es crucial aprovechar al máximo las clases preexistentes que pueden satisfacer las necesidades de un programa. Este enfoque se basa en el principio de no "reinventar la rueda". Aquí hay más detalles sobre por qué esto es importante:

- **Ahorro de tiempo**: Reutilizar clases y bibliotecas existentes permite a los desarrolladores ahorrar tiempo valioso. En lugar de escribir código desde cero para cada funcionalidad, pueden aprovechar soluciones probadas y confiables.
- **Mejora de la calidad del software**: Las clases preconstruidas suelen haber pasado por un riguroso proceso de prueba y depuración. Esto significa que son menos propensas a errores y problemas de rendimiento que el código nuevo y no probado.

Acosta Porcayo Alan Omar Ing. en Computación 320206102
Gutiérrez Grimaldo Alejandro Ing. en Computación 320282098
Medina Villa Samuel Ing. en Computación 320249538

Arreglos

Los arreglos en programación son estructuras de datos que permiten almacenar una colección de elementos del mismo tipo. Aquí hay más información sobre su uso:

- **Índices numéricos**: Cada elemento en un arreglo se identifica mediante un índice numérico. Los índices comienzan generalmente en 0 (cero) para el primer elemento y aumentan secuencialmente.
- **Declaración y asignación**: Los arreglos se pueden declarar y asignar de varias formas en diferentes lenguajes de programación. En Java, por ejemplo, puedes declarar un arreglo de enteros de la siguiente manera: *int[] numeros = new int[5];*.
- **Acceso a elementos**: Para acceder a un elemento específico en un arreglo, se utiliza su índice. Por ejemplo, *numeros[2]* accedería al tercer elemento en el arreglo (índice 2).

Argumentos por línea de comandos

La capacidad de pasar argumentos por línea de comandos a un programa es útil en muchas situaciones. Aquí hay más información sobre cómo funciona y cuándo es beneficioso:

- **Versatilidad**: Permite que un programa realice diferentes tareas o se comporte de manera diferente según los argumentos proporcionados al ejecutarlo. Esto lo hace más versátil y reutilizable.
- **Ejemplos comunes**: Algunos ejemplos comunes de uso de argumentos por línea de comandos incluyen proporcionar configuraciones personalizadas, indicar archivos de entrada/salida o activar/desactivar funciones específicas del programa.

API de Java

La API de Java es una vasta biblioteca de clases y métodos proporcionados por los desarrolladores del lenguaje Java. Aquí hay más detalles sobre su importancia:

- **Amplia funcionalidad**: La API de Java abarca una amplia gama de funcionalidades, desde manipulación de cadenas hasta operaciones matemáticas avanzadas. Esto significa que los desarrolladores pueden acceder a una gran cantidad de herramientas sin tener que escribir todo desde cero.
- **Organización en paquetes**: La API de Java está organizada en paquetes lógicos, lo que facilita la búsqueda y el uso de clases relacionadas. Por ejemplo, las clases de manipulación de archivos se encuentran en el paquete *java.io*.

Manejo de cadenas

El manejo de cadenas es una parte fundamental de la programación, ya que la mayoría de las aplicaciones deben trabajar con texto de alguna forma. Aquí hay más información sobre las operaciones comunes de manejo de cadenas:

- **Concatenación**: Unir o concatenar cadenas es una operación común. En Java, esto se hace utilizando el operador +. Por ejemplo, "Hola" + "Mundo" resultaría en "HolaMundo".
- **Longitud de cadena**: Puedes determinar la longitud de una cadena utilizando el método *length()*. Esto es útil para saber cuántos caracteres contiene una cadena.
- **Búsqueda de subcadenas**: Puedes buscar subcadenas dentro de una cadena utilizando métodos como *indexOf()* o *contains()*. Esto es útil para buscar palabras clave o patrones en texto.

Wrappers

Las clases envoltorio o "*wrappers*" en Java son fundamentales para trabajar con tipos primitivos como enteros, flotantes, etc. Aquí hay más información sobre su utilidad:

- **Conversión de tipos**: Los wrappers permiten convertir tipos primitivos en objetos y viceversa. Por ejemplo, puedes convertir un *int* en un *Integer* utilizando *wrappers*.
- **Usos comunes**: Los *wrappers* son especialmente útiles cuando se trabaja con colecciones de objetos, ya que las colecciones generalmente requieren objetos en lugar de tipos primitivos.

Colecciones

Las colecciones en Java son estructuras de datos que permiten almacenar y manipular grupos de objetos. Aquí hay más información sobre las colecciones y sus tipos comunes:

- **Dinámicas**: A diferencia de los arreglos estáticos, las colecciones pueden cambiar de tamaño dinámicamente a medida que se agregan o eliminan elementos.
- **Tipos comunes**: Algunos tipos comunes de colecciones en Java incluyen listas (por ejemplo, *ArrayList*), conjuntos (por ejemplo, *HashSet*) y mapas (por ejemplo, *HashMap*).
- **Operaciones básicas**: Las operaciones comunes en colecciones incluyen agregar elementos, eliminar elementos, buscar elementos y recorrer la colección para realizar diversas operaciones.

Clases de utilerías

Java proporciona una serie de clases de utilería que simplifican tareas comunes, como cálculos matemáticos y manipulación de fechas. Algunas de las clases más útiles son:

- ***Math***: La clase *Math* proporciona métodos para realizar operaciones matemáticas avanzadas, como exponenciación, raíz cuadrada y funciones trigonométricas.
- ***Date* y *Calendar***: Estas clases permiten trabajar con fechas y horas, realizar cálculos de tiempo y dar formato a las fechas según las necesidades del programa.

Objetivos

- Utilizar bibliotecas estándar como punto de partida para aprender y comprender conceptos

y técnicas de programación antes de explorar soluciones personalizadas.

- Utilizar bibliotecas para incorporar funcionalidades probadas y confiables, mejorando así la calidad del código.
- Aprender a adaptar y personalizar las bibliotecas para satisfacer necesidades específicas sin tener que crear soluciones desde cero.

Metodología

Ejercicio realizado por el profesor

Código

```
import java.util.Hashtable;
import java.time.LocalDate;
import java.util.Enumeration;
import java.util.ArrayList;
import java.time.LocalDate;

public class pract3 {
  public static void main(String[] args) {
    int[] nums = new int[10];
    for(int i = 0; i < nums.length; i++) {
      nums[i] = i * 2;
    }

    //foreeach
    for(int k: nums) {
      System.out.println(k);
    }

    int[][] m = new int[5][5];
    for(int i = 0; i < m.length; i++) {
      for(int j = 0; j < m[0].length; j++) {
        if(i == j) {
          m[i][j] = 1;
        }
      }
    }

    for(int i = 0; i < m.length; i++) {
      for(int j = 0; j < m[0].length; j++) {
        System.out.print(m[i][j] + " ");
      }
      System.out.println();
    }

    String a = args[0];
    int n = Integer.parseInt(args[1]); // moldeaado o casting

    System.out.println("El GOAT es: " + a + " : " + n);

    String s = "Samuel el reprobado de POO GRUPO 3";

    s = s.toUpperCase();
    System.out.println(s);

    StringBuilder sb = new StringBuilder(s);
    sb.append(" por irle a MESSI");
    System.out.println(sb);

    Hashtable<String,String> atlas = new Hashtable<String,String>();

    atlas.put("Mexico", "CDMX");
    atlas.put("Rusia ", "Moscu");
    atlas.put("Etiopia", "Addis Abeba");
    atlas.put("Marruecos", "Rabat");
    atlas.put("Sudafrica", "Pretoria");

    String pais;
    String capital;

    Enumeration<String> claves = atlas.keys();

    while(claves.hasMoreElements()) {
      pais = claves.nextElement();
      capital = atlas.get(pais);
      System.out.println("Pais: " + pais + " su capital es " + capital);
    }

    ArrayList<String> programastv = new ArrayList<String>();

    programastv.add("Tortugas Ninja");
    programastv.add("Hey Arnold!");
    programastv.add("Los chicos del barrio");
    programastv.add("Los cuentos de la calle broca");
    programastv.add("Zobomafuu y los hermanos Krat");
    programastv.add("Power Rangers");

    programastv.remove(0);

    for(String pr: programastv) {
      System.out.println(pr);
    }

    LocalDate hoy = LocalDate.now();
    System.out.println(hoy);
  }
}
```

Resultados

Problema 1

Elaboré el modelo de encriptación antiguo llamado *Scytale*. Ubicado en: https://edabit.com/challenge/KXs93N4RX6jNSsgCr.

Utilice el manejo de cadenas y arreglos matriciales.

Explicación

El programa funciona a partir de un mensaje y un entero positivo ingresados por el usuario. El mensaje y el entero se mandan como parámetros a la función *cipher*, en la cual primero se obtiene la longitud del mensaje y se crea una copia del mensaje instanciando la clase *StringBuilder*. Para cifrar el mensaje se necesita una matriz utilizando el valor de n como las filas y dividiendo la longitud del mensaje entre n (si esta división tiene residuo, se le suma 1 y a la copia del mensaje se le agregan espacios en blanco). A la matriz se le asignan los caracteres de la copia del mensaje fila por fila y a otra cadena se le agregan los caracteres de la matriz columna por columna. Por último se retorna el mensaje cifrado y se imprime.

Código

```
import java.util.Scanner;

public class Problema1 {
  static StringBuilder cipher(String m, int n) {
    int mLen = m.length();
```

```
    int cols;
    StringBuilder sb = new StringBuilder(m);

    if (mLen % n != 0) {
      cols = (mLen / n) + 1;
      for(int i = 0; i < (n * cols) - mLen; i ++)
        sb.append(' ');
    } else
      cols = mLen / n;

    char[][] matrix = new char[n][cols];
    for(int i = 0; i < matrix.length; i ++) {
      for(int j = 0; j < matrix[0].length; j ++)
        matrix[i][j] = sb.charAt((i * cols) + j);
    }

    StringBuilder cipher = new StringBuilder();
    for(int i = 0; i < matrix[0].length; i ++) {
      for(int j = 0; j < matrix.length; j ++)
        cipher.append(matrix[j][i]);
    }

    return cipher;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String mensaje;
    int n;

    System.out.print("Mensaje: ");
    mensaje = sc.nextLine();
    do {
      System.out.print("n: ");
      n = sc.nextInt();
      if (n <= 0)
        System.out.println("n debe ser positivo\n");
    } while(n <= 0);

    System.out.println("\n" + cipher(mensaje, n));

    sc.close();
  }
}
```

Terminal

```
Mensaje: Mubashir Scytale Code
n: 6

Ms t euhSaC biclo arved
```

Problema 3

Programe el juego de la tortuga y la liebre.

En consola de debe de mostrar siempre la pista de 100 unidades en una matriz de 10×10 de esta manera:

Donde S es *Start*, E es *End*, H es *Hare* (Liebre) y T es *Turtle*.

Se lanzará un dado por cada jugador y siempre arrancará la Tortuga. Si la Tortuga o la Liebre llegan a los siguientes números:

- 7, 14, 33, 77 y 89 se obtiene una segunda tirada.

S	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33 H	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56 T	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	E

- 6, 23, 42, 56, 82, 90 el jugador se regresa 5 casillas para atrás.
- 36 y 65 son números especiales ambos van al 71 y al 84.
- El 10 regresa al inicio del juego y 66 regresa al 40.

Utilice arreglos matriciales y un numero aleatorio para simular el dado.

Explicación

El programa funciona a partir de 4 métodos y una variable de clase que representa la pista de 10×10. El método *inicio* le asigna a las casillas el número de la posición o un carácter si es el inicio o el final de la pista. El método *pista* imprime la matriz usando dos ciclos *for*. El método *jugada* toma como parámetro la posición actual del jugador en turno y le suma un número aleatorio entre 1 y 6 simulando un dado. Las estructuras *if* comprueban si se cayo en una casilla especial para realizar una operación extra.

Por último, el método *main* inicializa la pista y dentro de un ciclo *while* sucede cada turno. Los turnos impares son de la tortuga y los pares de la liebre. En cada turno se llama al método *jugada* y se obtiene un nueva posición. Luego se elimina el carácter que representa al jugador en la casilla en la que estaba antes de la tirada de dado, para colocar el carácter en la nueva posición. El juego termina hasta que algún jugador llegue a la casilla 100.

Código

```
import java.util.Random;

public class Problema3 {
  static StringBuilder[][] p = new StringBuilder[10][10];

  static void inicio() {
    int k = 2;

    p[0][0] = new StringBuilder("S");
    p[9][9] = new StringBuilder("E");

    for(int i = 0; i < p.length; i ++) {
      for(int j = 0; j < p[0].length; j ++){
        if((i == 0 && j == 0) || (i == 9 && j == 9))
          continue;
        else {
          p[i][j] = new StringBuilder(Integer.toString(k));
          k ++;
```

```
      }
    }
  }

  p[0][0].append("HT");
}

static void pista() {
  for(int i = 0; i < p.length; i ++) {
    for(int j = 0; j < p[0].length; j ++)
      System.out.print(p[i][j] + "\t");
    System.out.println();
  }
}

static int jugada(int nPos) {
  Random rand = new Random();
  int d, n = nPos;

  d = rand.nextInt(6) + 1;
  n += d;

  if (n > 100)
    n = 100;

  System.out.println("Dado: " + d);
  System.out.println("Avanza de " + nPos +" a " + n);

  if(n == 7 || n == 14 || n == 33 || n == 77 || n == 89) {
    System.out.println("Casilla Especial!\tTira de nuevo");
    n = jugada(n);
  } else if(n == 6 || n == 23 || n == 42 || n == 56 || n == 82 || n == 90) {
    System.out.println("Trampa\tRetrocede de " + n + " a " + (n - 5));
    n -= 5;
  } else if(n == 36) {
    System.out.println("Casilla Especial!\tAvanza de " + n + " a 71");
    n = 71;
  } else if(n == 65) {
    System.out.println("Casilla Especial!\ttAvanza de " + n + " a 84");
    n = 84;
  } else if(n == 10) {
    System.out.println("Trampa\tRegresa al inicio");
    n = 1;
  } else if(n == 66) {
    System.out.println("Trampa\tRegresa de " + n + " a 40");
    n = 40;
  }

  return n;
}

public static void main(String[] args) {
  int posIH = 0, posJH = 0, posIT = 0, posJT = 0, nH = 1, nT = 1, turno = 0;

  inicio();

  System.out.println("Pista inicial");
  pista();

  while(true) {
    turno ++;

    System.out.print("\nTurno " + turno);
    if(turno % 2 != 0){
      System.out.println(". Tira la Tortuga\n");

      nT = jugada(nT);

      if(p[posIT][posJT].charAt(p[posIT][posJT].length() - 1) == 'T')
        p[posIT][posJT].deleteCharAt(p[posIT][posJT].length() - 1);
      else
        p[posIT][posJT].deleteCharAt(p[posIT][posJT].length() - 2);

      if(nT == 0) {
        posIT = 0;
        posJT = 0;
      } else {
        posIT = (nT - 1) / 10;
        posJT = (nT - 1) % 10;
      }

      p[posIT][posJT].append("T");
    } else {
      System.out.println(". Tira la Liebre\n");

      nH = jugada(nH);

      if(p[posIH][posJH].charAt(p[posIH][posJH].length() - 1) == 'H')
        p[posIH][posJH].deleteCharAt(p[posIH][posJH].length() - 1);
      else
        p[posIH][posJH].deleteCharAt(p[posIH][posJH].length() - 2);

      if(nH == 0) {
        posIH = 0;
        posJH = 0;
      } else {
        posIH = (nH - 1) / 10;
        posJH = (nH - 1) % 10;
      }

      p[posIH][posJH].append("H");
    }

    pista();

    if(nH == 100 || nT == 100)
      break;

    System.out.println("Fin del turno\n");
  }

  System.out.println("\nGanador: " + (turno % 2 != 0 ? "Tortuga!" : "Liebre!"));
  System.out.println("Fin del juego");
 }
}
```

Terminal

```
Turno 31. Tira la Tortuga

Dado: 3
Avanza de 95 a 98
S       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20
21      22      23      24      25      26      27      28      29      30
31      32      33      34      35      36      37H     38      39      40
41      42      43      44      45      46      47      48      49      50
51      52      53      54      55      56      57      58      59      60
61      62      63      64      65      66      67      68      69      70
71      72      73      74      75      76      77      78      79      80
81      82      83      84      85      86      87      88      89      90
91      92      93      94      95      96      97      98T     99      E
Fin del turno


Turno 32. Tira la Liebre

Dado: 3
Avanza de 37 a 40
S       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20
21      22      23      24      25      26      27      28      29      30
31      32      33      34      35      36      37      38      39      40H
41      42      43      44      45      46      47      48      49      50
51      52      53      54      55      56      57      58      59      60
61      62      63      64      65      66      67      68      69      70
71      72      73      74      75      76      77      78      79      80
81      82      83      84      85      86      87      88      89      90
91      92      93      94      95      96      97      98T     99      E
Fin del turno


Turno 33. Tira la Tortuga

Dado: 6
Avanza de 98 a 100
S       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20
21      22      23      24      25      26      27      28      29      30
31      32      33      34      35      36      37      38      39      40H
41      42      43      44      45      46      47      48      49      50
51      52      53      54      55      56      57      58      59      60
61      62      63      64      65      66      67      68      69      70
71      72      73      74      75      76      77      78      79      80
81      82      83      84      85      86      87      88      89      90
91      92      93      94      95      96      97      98      99      ET

Ganador: Tortuga!
Fin del juego
```

Problema 4

Cree 4 conjuntos A, B, C, D, llene los conjuntos de manera aleatoria de tamaño 10, 20, 30 y 40 respectivamente de números aleatorios entre 0 y 200. Encuentre los siguientes valores:

a. $A \cap B \cap C \cap D$

b. $(A \cap B) \cup (C \cap D)$

c. Calcule el $U = A \cup B \cup C \cup D$ sin repetición de elementos.

d. $B - U = B^C$

Utilice Sets y sus operaciones.

Explicación

Para la generación de los conjuntos A, B, C y D se utiliza el método *generarConjuntoAleatorio()* que se encarga de generar conjuntos de números enteros aleatorios y devuelve un conjunto *Set<Integer>*.

Para el inciso a) ee crea un nuevo conjunto *interseccionABCD*. Se calcula la intersección de *interseccionABCD* con los conjuntos A, B, C y D utilizando el método *retainAll()*. El resultado es un conjunto que contiene solo los elementos que están presentes en todos los conjuntos A, B, C y D.

Para el inciso b) ee calcula la intersección de los conjuntos A y B en *interseccionAB*. Se calcula la intersección de los conjuntos C y D en *interseccionCD*. Se crea un nuevo conjunto *unionABCD* que contiene todos los elementos de *interseccionAB* y *interseccionCD* combinados. Esto representa la unión de dos intersecciones.

Para el inciso c) se crea un nuevo conjunto *unionSinRepetir*. Se agregan todos los elementos de los conjuntos A, B, C y D a *unionSinRepetir*. La particularidad de usar *HashSet* es que se asegura de que no haya elementos duplicados, por lo que al final, *unionSinRepetir* contiene la unión de todos los conjuntos sin elementos repetidos.

Pot ultimo, para el inciso d) se crea un nuevo conjunto *complementoBU* que comienza como una copia del conjunto B utilizando *new HashSet<>(B)*. Luego, se utiliza el método *removeAll()* en *complementoBU* para eliminar todos los elementos que están en el conjunto *unionSinRepetir*. El conjunto *complementoBU* resultante contiene los elementos que están no estan en B.

Código

```
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

public class Problema4 {
  public static void main(String[] args) {
    Set<Integer> A = generarConjuntoAleatorio(10);
    Set<Integer> B = generarConjuntoAleatorio(20);
    Set<Integer> C = generarConjuntoAleatorio(30);
    Set<Integer> D = generarConjuntoAleatorio(40);

    System.out.println("Conjunto A: " + A);
    System.out.println("Conjunto B: " + B);
    System.out.println("Conjunto C: " + C);
    System.out.println("Conjunto D: " + D);

    Set<Integer> interseccionABCD = new HashSet<>(A);
    interseccionABCD.retainAll(B);
    interseccionABCD.retainAll(C);
    interseccionABCD.retainAll(D);

    Set<Integer> interseccionAB = new HashSet<>(A);
    interseccionAB.retainAll(B);
    Set<Integer> interseccionCD = new HashSet<>(C);
    interseccionCD.retainAll(D);
    Set<Integer> unionABCD = new HashSet<>(interseccionAB);
    unionABCD.addAll(interseccionCD);

    Set<Integer> unionSinRepetir = new HashSet<>(A);
    unionSinRepetir.addAll(B);
    unionSinRepetir.addAll(C);
    unionSinRepetir.addAll(D);

    Set<Integer> complementoBU = new HashSet<>(unionSinRepetir);
    complementoBU.removeAll(B);

    System.out.println("\n\na. A interseccion B interseccion C interseccion
     D: " + interseccionABCD);
    System.out.println("\nb. (A interseccion B) union (C interseccion D):
    " + unionABCD);
    System.out.println("\nc. U = A union B union C union D sin repeticion de
     elementos: " + unionSinRepetir);
    System.out.println("\nd. Complemento de B con respecto a U: " +
    complementoBU);
  }

  public static Set<Integer> generarConjuntoAleatorio(int tamano) {
    Set<Integer> conjunto = new HashSet<>();
    Random random = new Random();
    while (conjunto.size() < tamano) {
      conjunto.add(random.nextInt(201));
    }
    return conjunto;
  }
}
```

Terminal

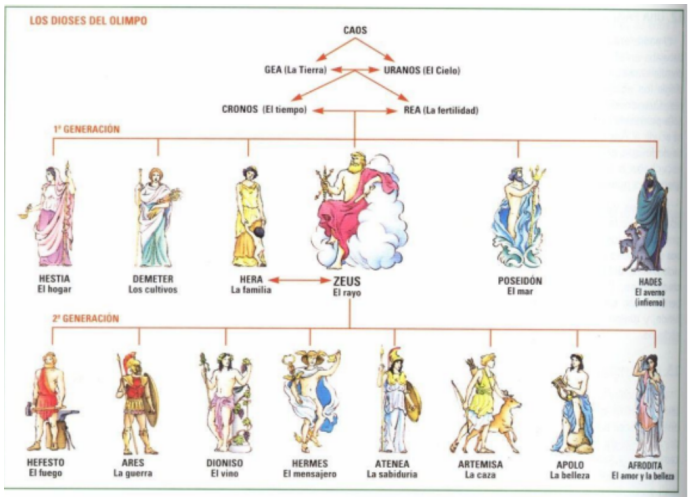
```
Conjunto A: [115, 100, 133, 104, 8, 24, 74, 91, 61, 190]
Conjunto B: [64, 36, 68, 164, 69, 40, 136, 46, 144, 112, 145, 178, 19, 149, 117, 119,
90, 154, 63, 31]
Conjunto C: [64, 1, 66, 133, 134, 199, 140, 13, 15, 145, 146, 19, 83, 148, 149, 91, 95
, 100, 166, 104, 41, 46, 175, 54, 120, 121, 58, 59, 188, 127]
Conjunto D: [0, 129, 2, 196, 134, 198, 70, 10, 13, 143, 15, 144, 145, 83, 147, 20, 88,
 90, 27, 31, 33, 98, 34, 164, 101, 38, 42, 107, 171, 108, 172, 109, 111, 47, 48, 180,
120, 57, 186, 59]


a. A interseccion B interseccion C interseccion D: []

b. (A interseccion B) union (C interseccion D): [145, 83, 134, 120, 59, 13, 15]

c. U = A union B union C union D sin repeticion de elementos: [0, 1, 129, 2, 133, 134,
 8, 136, 10, 140, 13, 15, 143, 144, 145, 146, 19, 147, 148, 20, 149, 24, 154, 27, 31,
33, 34, 36, 164, 166, 38, 40, 41, 42, 171, 172, 46, 175, 47, 48, 178, 180, 54, 57, 58,
 186, 59, 188, 61, 190, 63, 64, 66, 68, 196, 69, 198, 70, 199, 74, 83, 88, 90, 91, 95,
 98, 100, 101, 104, 107, 108, 109, 111, 112, 115, 117, 119, 120, 121, 127]

d. Complemento de B con respecto a U: [0, 1, 129, 2, 133, 134, 8, 10, 140, 13, 15, 143
, 146, 147, 148, 20, 24, 27, 33, 34, 166, 38, 41, 42, 171, 172, 175, 47, 48, 180, 54,
57, 58, 186, 59, 188, 61, 190, 66, 196, 198, 70, 199, 74, 83, 88, 91, 95, 98, 100, 101
, 104, 107, 108, 109, 111, 115, 120, 121, 127]
```

Problema 5

Construya un árbol genealógico **sencillo** de cualquier panteón mitológico (griego, romano, nórdico, etc) seleccione cualquier tipo de estructura tipo Tree que java ofrece. De manera tal que una estructura de tipo Tree es una persona y sobre el estarán sus hij@s.

Ejemplo:

Cronos → Zeus, Poseidón, Hades



Zeus → Hermes, Atenea, Artemisa, Apolo, Afrodita

Y podemos buscar sus descendientes, borrar, o agregar. Y buscar sus antecesores.

Explicación

Es una implementación de un árbol genealógico ficticio basado en la mitología griega. A través de una serie de nodos, representados como objetos DefaultMutableTreeNode, se establecen relaciones jerárquicas entre diversos personajes mitológicos griegos, como Urano, Cronos, Zeus y otros dioses y diosas importantes.

Ofrece funciones para explorar, modificar y visualizar las relaciones familiares entre los personajes.

Código

```
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeNode;
import java.util.Scanner;

public class Problema5 {
  private static DefaultMutableTreeNode uranos;
  private static DefaultMutableTreeNode zeus;
  private static Scanner sc;

  public Problema5() {
    // Crear los nodos para los personajes
    uranos = new DefaultMutableTreeNode("Uranos");
    DefaultMutableTreeNode cronos = new DefaultMutableTreeNode("Cronos");

    zeus = new DefaultMutableTreeNode("Zeus");
    DefaultMutableTreeNode ades = new DefaultMutableTreeNode("Ades");
    DefaultMutableTreeNode hestia = new DefaultMutableTreeNode("Hestia");
    DefaultMutableTreeNode demeter = new DefaultMutableTreeNode("Demeter");
    DefaultMutableTreeNode hera = new DefaultMutableTreeNode("Hera");
    DefaultMutableTreeNode poseidon = new DefaultMutableTreeNode("Poseidon");

    DefaultMutableTreeNode hefesto = new DefaultMutableTreeNode("Hefesto");
    DefaultMutableTreeNode ares = new DefaultMutableTreeNode("Ares");
    DefaultMutableTreeNode dioniso = new DefaultMutableTreeNode("Dioniso");
    DefaultMutableTreeNode hermes = new DefaultMutableTreeNode("Hermes");
    DefaultMutableTreeNode atenea = new DefaultMutableTreeNode("Atenea");
    DefaultMutableTreeNode artemisa = new DefaultMutableTreeNode("Artemisa");
    DefaultMutableTreeNode apolo = new DefaultMutableTreeNode("Apolo");
    DefaultMutableTreeNode afrodita = new DefaultMutableTreeNode("Afrodita");

    // Construir el arbol genealogico
    uranos.add(cronos);
    cronos.add(zeus);
    cronos.add(poseidon);
    cronos.add(ades);
    cronos.add(hestia);
    cronos.add(demeter);
    cronos.add(hera);
    cronos.add(poseidon);

    zeus.add(hefesto);
    zeus.add(ares);
    zeus.add(dioniso);
    zeus.add(hermes);
    zeus.add(atenea);
    zeus.add(artemisa);
    zeus.add(apolo);
    zeus.add(afrodita);

    sc = new Scanner(System.in);
  }

  public TreeNode buscarDescendiente(TreeNode root, String nombre) {
    if (root.toString().equalsIgnoreCase(nombre)) {
      return root;
    }
    for (int i = 0; i < root.getChildCount(); i++) {
      TreeNode resultado = buscarDescendiente(root.getChildAt(i), nombre);
      if (resultado != null) {
        return resultado;
      }
    }
    return null;
  }

  public void printTree(TreeNode root, String indent) {
    System.out.println(indent + root);
    for (int i = 0; i < root.getChildCount(); i++) {
      printTree(root.getChildAt(i), indent + " | ");
    }
  }

  public void buscarUnDescendiente() {
    // Buscar un descendiente
    System.out.print("\nIngresa el nombre que deseas buscar: ");
    String nombreABuscar = sc.nextLine();
    TreeNode nodoEncontrado = buscarDescendiente(uranos, nombreABuscar);
    if (nodoEncontrado != null) {
      System.out.println("Descendiente encontrado: " + nodoEncontrado);
      if (nodoEncontrado.getParent() != null)
        System.out.println("\nPadre: " + nodoEncontrado.getParent());
      else
        System.out.println("\nPadre: No tiene");
      // desendientes
      if (nodoEncontrado.getChildCount() == 0)
        System.out.println("\nHijos: No tiene");
      else {
        System.out.println("\nHijos: ");
        for (int i = 0; i < nodoEncontrado.getChildCount(); i++) {
          System.out.println(" - " + nodoEncontrado.getChildAt(i));
        }
      }
    } else {
      System.out.println("Descendiente no encontrado: " + nombreABuscar);
    }
  }

  public void agregarUnDescendiente() {
    System.out.println("\nAgregar un descendiente a Zeus:");
    System.out.print("Ingresa el nombre del nuevo descendiente: ");
    String descendiente = sc.nextLine();
    DefaultMutableTreeNode nuevoDescendiente = new
                        DefaultMutableTreeNode(descendiente);
    zeus.add(nuevoDescendiente);
    System.out.println("Descendiente agregado a Zeus: " + descendiente);
  }

  public void eliminarUnDescendiente() {
    System.out.println("\nEliminar un descendiente de Zeus:");
    System.out.print("Ingresa el nombre del descendiente a eliminar: ");
    String descendiente = sc.nextLine();
    DefaultMutableTreeNode nodoAEliminar = (DefaultMutableTreeNode)
                        buscarDescendiente(zeus, descendiente);
    if (nodoAEliminar != null) {
      zeus.remove(nodoAEliminar);
      System.out.println("Descendiente eliminado de Zeus: " + nodoAEliminar);
    } else {
      System.out.println("Descendiente no encontrado en Zeus: " + descendiente);
    }
  }

  public static void main(String[] args) {
    Problema5 programa = new Problema5();

    // Menu
    int opcion = 0;
```

```
    do {
      System.out.println("\nMenu:");
      System.out.println("1. Imprimir arbol genealogico");
      System.out.println("2. Buscar un descendiente");
      System.out.println("3. Agregar un descendiente a Zeus");
      System.out.println("4. Eliminar un descendiente de Zeus");
      System.out.println("5. Salir");
      System.out.print("Ingresa una opcion: ");
      opcion = Integer.parseInt(sc.nextLine());
      switch (opcion) {
        case 1:
          System.out.println();
          programa.printTree(uranos, "");
          break;
        case 2:
          programa.buscarUnDescendiente();
          break;
        case 3:
          programa.agregarUnDescendiente();
          break;
        case 4:
          programa.eliminarUnDescendiente();
          break;
        case 5:
          System.out.println("Adios!");
          break;
        default:
          System.out.println("Opcion no valida");
          break;
      }
    } while (opcion != 5);

    sc.close();
  }
}
```

Terminal

```
Menu:
1. Imprimir arbol genealogico
2. Buscar un descendiente
3. Agregar un descendiente a Zeus
4. Eliminar un descendiente de Zeus
5. Salir
Ingresa una opcion: 4

Eliminar un descendiente de Zeus:
Ingresa el nombre del descendiente a eliminar: Thor
Descendiente no encontrado en Zeus: Thor

Menu:
1. Imprimir arbol genealogico
2. Buscar un descendiente
3. Agregar un descendiente a Zeus
4. Eliminar un descendiente de Zeus
5. Salir
Ingresa una opcion: 1

Uranos
 | Cronos
 |  | Zeus
 |  |  | Hefesto
 |  |  | Ares
 |  |  | Dioniso
 |  |  | Hermes
 |  |  | Atenea
 |  |  | Artemisa
 |  |  | Apolo
 |  |  | Afrodita
 |  | Ades
 |  | Hestia
 |  | Deméter
 |  | Hera
 |  | Poseidón

Menu:
1. Imprimir arbol genealogico
2. Buscar un descendiente
3. Agregar un descendiente a Zeus
4. Eliminar un descendiente de Zeus
5. Salir
Ingresa una opcion: 5
Adios!
```

Conclusiones

La incorporación de bibliotecas en la programación se convierte en un recurso crucial para potenciar la eficacia, la calidad y la cooperación en el proceso de desarrollo de software. Estas bibliotecas ofrecen soluciones previamente diseñadas para tareas comunes, lo que conlleva ahorro de tiempo y minimiza la probabilidad de cometer errores. Asimismo, fomentan la capacidad de reutilizar código. En última instancia, adquirir destrezas en la utilización de bibliotecas se convierte en una habilidad esencial y contribuye al logro de éxitos en el ámbito de la programación.

Referencias

Solano, J. (2017, 20 enero). *Manual de prácticas de Programación Orientada a Objetos*. Laboratorio de Computación Salas A y B. http://lcp02.fi-b.unam.mx/

Deep Xavier. *Spartans Cipher*. Edabit. https://edabit.com/challenge/KXs93N4RX6jNSsgCr